

Data Processing with Databricks Spark

Analyzing Trips of NYC Taxi Cab Data

BIG DATA ENGINEERING - II

PRESENTED BY

Akshaya Parthasarathy
14081133



1. Overview

Provided by technology providers authorised under the Taxicab & Livery Passenger Enhancement Programs, to the NYC Taxi and Limousine Commission, the TLC Trip Record Data is an extensive dataset containing trip details of NYC cabs.

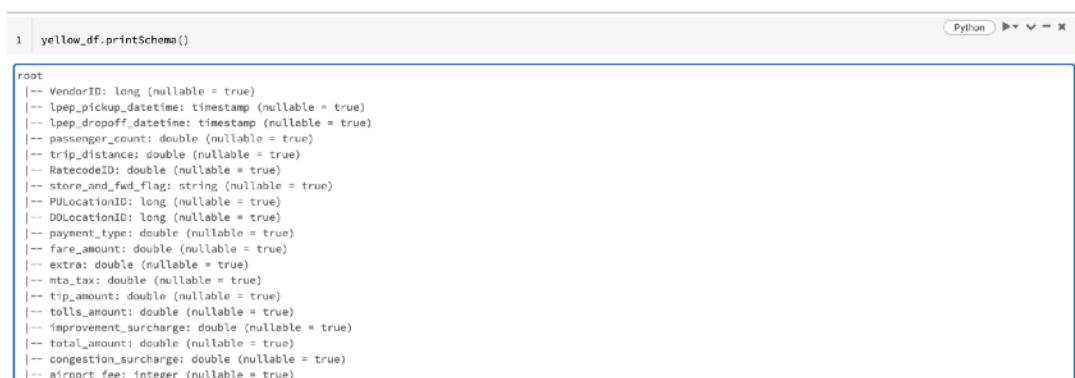
Dating all the way back to **2009**, the dataset contains the trip records for each month which includes details such as the number of passengers in the cab, pickup and drop-off times and locations and fares. Initially the dataset only contained details for the classic NYC Yellow taxi cabs, but in 2013 trip records for Green cabs were also added. Additionally, For-Hire Vehicle trip records are included but will not be used for the purpose of this project.

The aim of this project will be to analyse the trip record data of Yellow and Green taxi cabs from January 2019 to April 2022 using **Databricks Spark**. The datasets are initially uploaded into a **cloud storage platform** (eg: Azure) and integrated into Databricks for further analysis using SparkSQL. PySpark will be used to build two ML models in order to predict the total amount column. The ML models will be tested for accuracy on a slice of the dataset.

2. Dataset Exploration

The yellow taxi dataset consists of **19 columns** and a total of **152 million rows** prior to any cleaning. The green taxi cab has a total of 20 columns, ehail_fee which allows passengers to electronically hail cabs via TLC approved applications is the additional column. The green taxi dataset contains just over 9 million rows prior to data cleaning. **There are a total of 80 files**. The schema for both datasets can be found below.

Each trip record file is in Parquet File Format (.parquet). Apache Parquet has been made use as it provides efficient data compression on large databases, which is necessary for the dataset we are working with.



```
1 yellow_df.printSchema()

root
|-- VendorID: long (nullable = true)
|-- lpep_pickup_datetime: timestamp (nullable = true)
|-- lpep_dropoff_datetime: timestamp (nullable = true)
|-- passenger_count: double (nullable = true)
|-- trip_distance: double (nullable = true)
|-- RatecodeID: double (nullable = true)
|-- store_and_fwd_flag: string (nullable = true)
|-- PULocationID: long (nullable = true)
|-- DOLocationID: long (nullable = true)
|-- payment_type: double (nullable = true)
|-- fare_amount: double (nullable = true)
|-- extra: double (nullable = true)
|-- mta_tax: double (nullable = true)
|-- tip_amount: double (nullable = true)
|-- tolls_amount: double (nullable = true)
|-- improvement_surcharge: double (nullable = true)
|-- total_amount: double (nullable = true)
|-- congestion_surcharge: double (nullable = true)
|-- airport_fee: integer (nullable = true)
```

Fig 1: Schema of Yellow Taxi Dataset

```

Cmd 40
1 green_df.printSchema()

root
 |-- VendorID: long (nullable = true)
 |-- lpep_pickup_datetime: timestamp (nullable = true)
 |-- lpep_dropoff_datetime: timestamp (nullable = true)
 |-- store_and_fwd_flag: string (nullable = true)
 |-- RatecodeID: double (nullable = true)
 |-- PULocationID: long (nullable = true)
 |-- DOLocationID: long (nullable = true)
 |-- passenger_count: double (nullable = true)
 |-- trip_distance: double (nullable = true)
 |-- fare_amount: double (nullable = true)
 |-- extra: double (nullable = true)
 |-- mta_tax: double (nullable = true)
 |-- tip_amount: double (nullable = true)
 |-- tolls_amount: double (nullable = true)
 |-- ehail_fee: double (nullable = true)
 |-- improvement_surcharge: double (nullable = true)
 |-- total_amount: double (nullable = true)
 |-- payment_type: double (nullable = true)
 |-- trip_type: double (nullable = true)

```

Fig 2: Schema of Green Taxi Dataset

3. System Architecture

Databricks is a managed platform for running Apache Spark. By providing a one click solution to create, clone and delete clusters one does not have to learn complex cluster management nor perform tedious maintenance tasks to take advantage of Spark.

Databricks is structured to enable secure cross-functional team collaboration while keeping a significant amount of backend services managed by Databricks so you can stay focused on your data science, data analytics, and data engineering tasks.

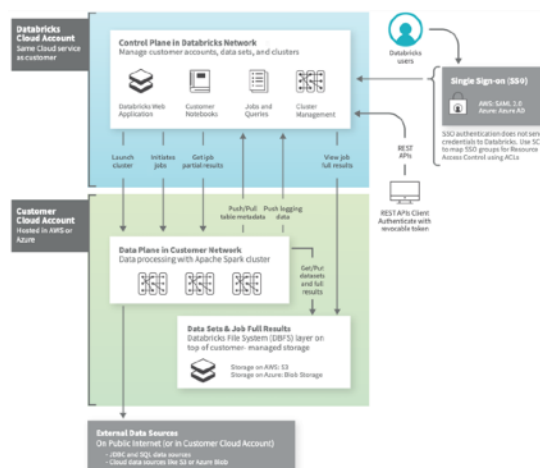


Fig 3: Databricks Architecture

4. Initial Configurations

Assuming that all the necessary data files have been downloaded, and an account on both Azure and Databricks has been verified, the data ingestion process can begin.

4.1 Microsoft Azure Setup

- Create a new container for the purpose of this project, eg: bde-at-2
- Upload each individual .parquet file into the container
- Retrieve STORAGE ACCOUNT NAME and STORAGE ACCOUNT ACCESS KEY under Security + Networking menu and the access key option
- Save this for establishing a connection to Databricks

4.2 Establishing connection with Databricks

Step 1: From your Databricks workspace, create a new cluster with a custom name, eg: bde_at_2

- *Note:* For community edition of Databricks, a clone of the cluster must be created if you're starting to code on a new day or after 2 hours of inactivity.

Step 2: In the Admin Console option in your workspace, click on the **Workspace Settings** then under **Advanced** toggle on **DBFS File Browser** to make it enabled.

Step 3: Create a new notebook, it will automatically attach itself to the active cluster.

Step 4: Create new variables STORAGE ACCOUNT NAME and STORAGE ACCOUNT ACCESS KEY and set it to the previously retrieved values.

Eg: `storage_account_name = "{STORAGE ACCOUNT NAME}"`

Step 5: Create a variable called blob_container_name and set it to the container created before.

Step 6: Mount the blob container into the Databricks File System using the following 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}  
)
```

Step 7: Verify that all files have been uploaded into DBFS by listing them.

Eg: `dbutils.fs.ls("/mnt/blob_container_name/")`

5. Part 1: Data Ingestion and Preparation

Load the Green & Yellow trip datasets into two different data frames. Since the schema of both datasets are different, there will be errors if you try to read all files together.

Eg: `green_df = spark.read.parquet("/mnt/bde-at-2/green_tripdata_*.parquet")`

Count the number of rows for both datasets by calling the `count()` function.

Yellow Taxi Dataframe should contain **152,823,008 rows**. Green Taxi Dataframe should contain **9,390,483 rows**.

5.1 Conversion to CSV

Create a new data frame “yellow_april” and read only the trip data from April, 2022.

Upon converting the CSV file and writing it back into Azure storage, we can notice a huge difference in file size.

<input type="checkbox"/>	 yellow_tripdata_2022-04.parquet	9/14/2022, 5:44:10 PM	Hot (Inferred)	Block blob	52.66 MiB	Available	***
--------------------------	---	-----------------------	----------------	------------	-----------	-----------	-----

Fig 4: Original Parquet File

Name	Modified	Access tier	Archive status	Blob type	Size	Lease state	
<input type="checkbox"/>  [-]							***
<input type="checkbox"/>  _committed_38037223398415945...	9/14/2022, 6:55:25 PM	Hot (Inferred)		Block blob	113 B	Available	***
<input type="checkbox"/>  _started_3803722339841594549	9/14/2022, 6:53:50 PM	Hot (Inferred)		Block blob	0 B	Available	***
<input type="checkbox"/>  _SUCCESS	9/14/2022, 6:55:29 PM	Hot (Inferred)		Block blob	0 B	Available	***
<input type="checkbox"/>  part-00000-tid-380372233984159...	9/14/2022, 6:55:22 PM	Hot (Inferred)		Block blob	394.96 MiB	Available	***

Fig 5: Converted CSV File

While the parquet file has compressed the file size to 55MB, the converted CSV file is almost 394MB. Considering that there are around 80 files, the amount of space occupied by them in CSV format would definitely exceed limits of Azure storage. The main difference in size can be attributed to the fact that CSV data is row-oriented, where each record is a row, but Parquet is columnar oriented, where each column is stored separately, thus allowing for better data processing on huge files.

5.2 Data Cleaning

With over a 160 million data in both datasets, there is a possibility that not all records are legitimate trips. Factors such as incorrect readings and data manipulation can contribute to dirty data that must be dealt with prior to building a good model. The following steps were taken to clean both the yellow and green datasets.

a) Based on Passenger Count

Trips where negative passengers, 0 passengers, and more than 5 passengers were removed. An empty trip cannot qualify as a valid record and the maximum legal capacity of a NYC taxi is 5, assuming that the 5th person is a small child.

b) Based on Distance Traveled

Trips with negative distance have been removed as they are not valid records. Trips where the distance traveled is less than 500m (or 0.310 miles) and trips where distance traveled is more than 240km (or 150 miles) have also been removed. The lower limit was set on the assumption that it's less than a block in NYC and the upper limit is considering the boundaries in which the cab can operate within NYC.

c) Based on Speed

While there is no minimum speed limit inside of NYC, the maximum limit is 55 mph, in the outskirts/highway. Considering these factors, any records where speed is less than 1mph has been removed assuming that the car is atleast in movement. An upper limit of 55.95 mph for speed is also set based on inspection of records.

287	2	2019-02-17T07:33:54.000+0000	2019-02-17T08:07:31.000+0000	33	0.55	30.77	55.9455
-----	---	------------------------------	------------------------------	----	------	-------	---------

Fig 6: Sample speed record

c) Based on Time

Records where the pickup time is after the dropoff time have been removed due to invalidity.

Trips that are greater than 2.5 hours or less than 2 minutes have also been removed on the account that they are either too long of a duration for a cab ride or too short (assuming rush hour, minimum speed limits etc.)

d) Based on fare

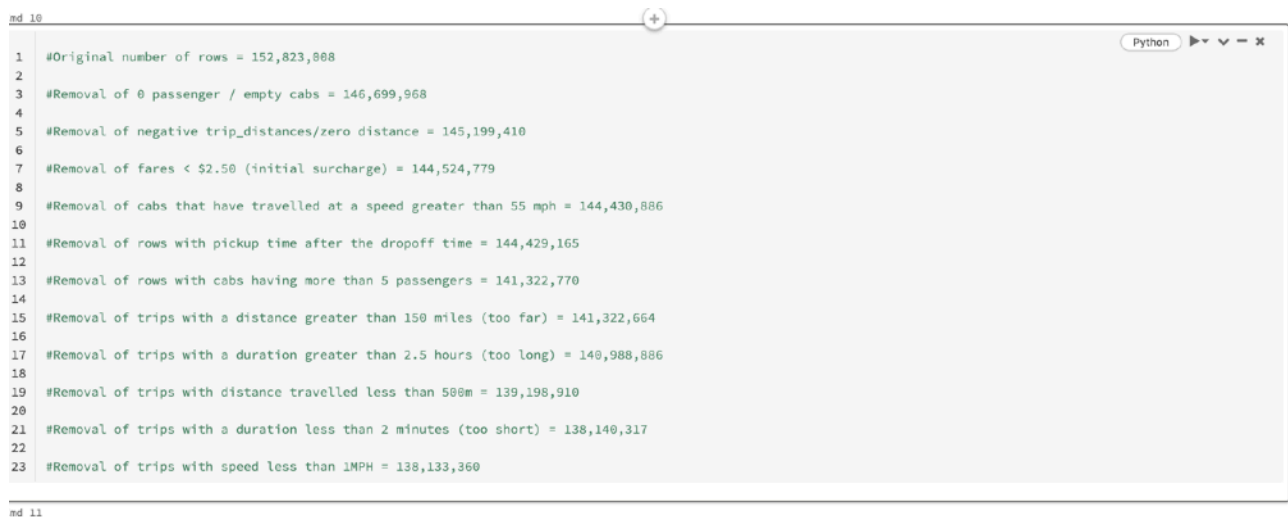
The base fare for hailing a cab is \$2.50, thus columns like fare_amount and total_amount have been automatically filtered to contain records greater than 2.50. The same is not applied to fields like tip, extra, and toll_amount since there is a possibility that these can be zero.

Once the cleaning has been completed, the final record count for both datasets are:

Yellow -> 138,133,360 rows

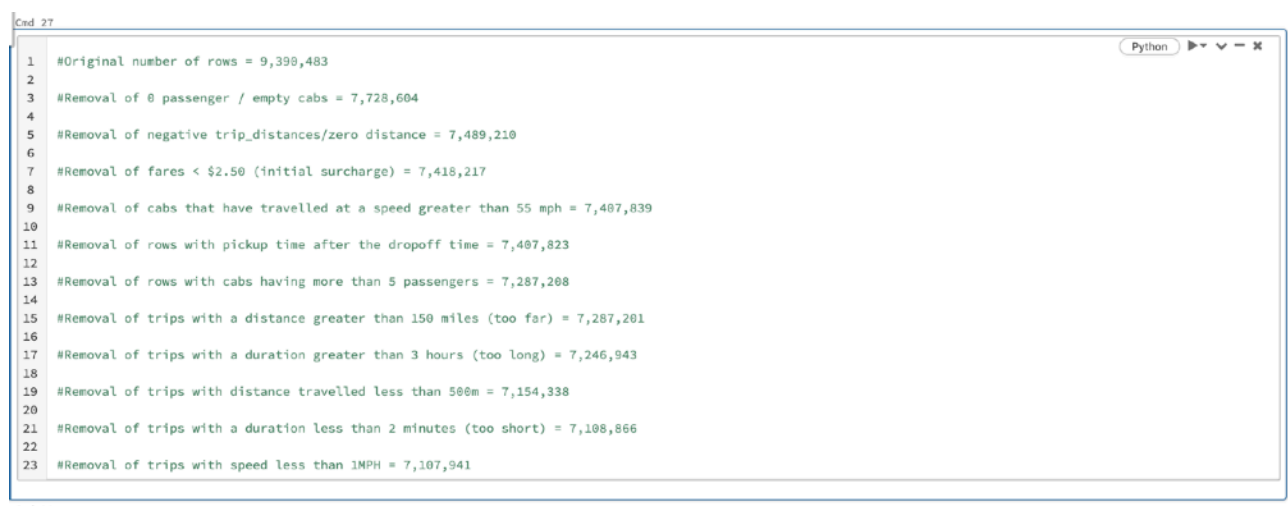
Green -> 7,107,941 rows

A stepwise sanity check for both data frames at each step of the cleaning process can be found below.



```
md 10
1 #Original number of rows = 152,823,888
2
3 #Removal of 0 passenger / empty cabs = 146,699,968
4
5 #Removal of negative trip_distances/zero distance = 145,199,410
6
7 #Removal of fares < $2.50 (initial surcharge) = 144,524,779
8
9 #Removal of cabs that have travelled at a speed greater than 55 mph = 144,430,886
10
11 #Removal of rows with pickup time after the dropoff time = 144,429,165
12
13 #Removal of rows with cabs having more than 5 passengers = 141,322,770
14
15 #Removal of trips with a distance greater than 150 miles (too far) = 141,322,664
16
17 #Removal of trips with a duration greater than 2.5 hours (too long) = 140,988,886
18
19 #Removal of trips with distance travelled less than 500m = 139,198,910
20
21 #Removal of trips with a duration less than 2 minutes (too short) = 138,140,317
22
23 #Removal of trips with speed less than 1MPH = 138,133,360
md 11
```

Fig 7: Sanity check on Yellow data frame



```
Cnd 27
1 #Original number of rows = 9,398,483
2
3 #Removal of 0 passenger / empty cabs = 7,728,604
4
5 #Removal of negative trip_distances/zero distance = 7,489,210
6
7 #Removal of fares < $2.50 (initial surcharge) = 7,418,217
8
9 #Removal of cabs that have travelled at a speed greater than 55 mph = 7,407,839
10
11 #Removal of rows with pickup time after the dropoff time = 7,407,823
12
13 #Removal of rows with cabs having more than 5 passengers = 7,287,208
14
15 #Removal of trips with a distance greater than 150 miles (too far) = 7,287,201
16
17 #Removal of trips with a duration greater than 3 hours (too long) = 7,246,943
18
19 #Removal of trips with distance travelled less than 500m = 7,154,338
20
21 #Removal of trips with a duration less than 2 minutes (too short) = 7,108,866
22
23 #Removal of trips with speed less than 1MPH = 7,107,941
md 28
```

Fig 8: Sanity check on Green data frame

5.3 Data Merge

Prior to merging the two data frames, a few additional data processing steps need to be taken to ensure that there aren't errors or processing issues during analysis.

Step 1: Conversion of payment_type column in the yellow data frame from String to Double type.

Step 2: Creating a new column called “taxi_colour” and appending the values “Yellow” and “Green” to each data frame to help identify the records easily after merging.

Step 3: Converting pickup time and drop-off time column names in the yellow data frame to match the column names in green so as to avoid errors in processing.

For the merge, utilise the unionByName function, setting the allowMissingColumns argument to True. Convert the data frame into a temporary View to perform data analysis.

Step 4 (Optional): Once the merge is complete, for ease of data analysis, extrapolate the year, month, day of the week, and hour using pyspark.sql.functions import and attach them as columns to the merged data frame.

5.4 Possible Issues and Errors

The merged dataset contains weird values in specific columns that were not previously identified in the Data Cleaning step, hence we must filter those records before proceeding.

- a) Keep only the records between the specified timeline of January 2019 to April 2022 and filtering weird record values where the years are outside of this limit.
- b) Filter out records where the value of VendorID is different from the ones provided in the data dictionary (i.e. besides 1 & 2).
- c) Filter out records with weird RatecodeID. According to the data dictionary the valid RatecodeID's are between 1 - 6.
- d) Filter out PULocationID and DOLocationID's that are not between 1 & 264, as defined in the data dictionary.

The merged data frame will now have a total of 143,403,154 rows.

Tip: For easier analysis and modeling from this point, you can take a random sample size (using sample()) from the merged data frame to work on. This cuts down computation time and allows for more experimentation.

6. Part 2: Business Questions

Q1) While the popular days of the week vary for the months from January 2019, a majority of the busy hours are either 6 P.M. or 3 P.M. which is the general rush hour traffic. Most cabs consist of either 1 or 2 passengers with the amount paid not going below \$15 or above \$20. For most drivers, these specific hours can turn out to be beneficial as there is always a need of cab service.

(6) Spark Jobs

year_month	total_trips	dayofweek	day_trip_count	hourofday	hour_trip_count	avg_number_passengers	avg_amount_per_trip	avg_amount_per_passenger
2019-1	7431115	Thursday	1310918	18	502927	1.46	15.69	13.34
2019-10	6914535	Thursday	1204472	18	467483	1.46	19.34	16.31
2019-11	6575222	Friday	1163620	18	437727	1.46	19.02	16.02
2019-12	6579228	Tuesday	1037709	18	422326	1.47	19.17	16.03
2019-2	6893767	Friday	1086131	18	467784	1.47	18.52	15.69
2019-3	7692606	Friday	1357190	18	512487	1.47	18.92	15.97
2019-4	7238410	Tuesday	1227163	18	489846	1.47	18.96	15.94
2019-5	7345850	Thursday	1287433	18	487835	1.47	19.3	16.24
2019-6	6744077	Saturday	1105705	18	426963	1.47	19.41	16.31
2019-7	6102591	Wednesday	1079871	18	402647	1.48	19.15	16.04
2019-8	5856288	Thursday	1057697	18	386118	1.48	19.2	16.06
2019-9	6314147	Thursday	950684	18	421469	1.46	19.5	16.44
2020-1	6108377	Friday	1075573	18	424504	1.44	18.22	15.49
2020-10	1481563	Thursday	273195	15	111385	1.37	17.34	15.19
2020-11	1327572	Monday	225256	15	106220	1.36	16.54	14.47
2020-12	1279611	Tuesday	239585	15	109519	1.35	16.51	14.45
2020-2	6017220	Saturday	1029182	18	425281	1.43	18.25	15.55
2020-3	2866878	Tuesday	451397	18	201813	1.4	18.25	15.62
2020-4	210161	Wednesday	38948	15	15814	1.24	15.73	14.43
2020-5	277254	Friday	54273	15	21562	1.27	16.13	14.62

only showing top 20 rows

Command took 16.28 minutes -- by akshaya.parthasarathy@student.uts.edu.au at 29/09/2022, 18:50:11 on bde-at-2 (clone)

Fig 9: Answer to Business Question 1

Q2) The average, mean, and median trip duration, distance in km, and speed in km/hr for each taxi colour are shown as below.

It is interesting to note that both the Yellow and Green taxi cabs have nearly similar values for each calculation. It would be ideal for the taxi drivers to choose trips based on the average duration and distance as a majority of the trips fall in that range.

▶ (3) Spark Jobs

	taxi_colour	avg_trip_duration	min_trip_duration	max_trip_duration	median_trip_duration	avg_distance_km	min_distance_km	max_distance_km	median_distance_km	avg_speed_km	min_speed_km	max_speed_km	median_speed_km
57	Green	14.37	2	149	11	5.1	0.51	169.86	3.12	20			
87	Yellow	13.76	2	149	11	4.87	0.51	205.31	2.75	19			

Command took 7.49 minutes -- by akshaya.parthasarathy@student.uts.edu.au at 29/09/2022, 15:22:16 on bde-at-2

Fig 10: Answer to Business Question 2

Q3) Out of all valid paid trips, nearly 70.5% of them are tipped. This is an encouraging statistic but further analysis into the amount paid in tips for each trip would provide more insight into customer behaviour.

End 65

```

1 #Question 3
2 spark.sql("""
3
4 WITH tip_calc_cte AS(
5 SELECT
6     SUM (
7         CASE
8             WHEN tip_amount > 0 THEN 1
9             ELSE 0
10        ) AS paid_tips,
11     COUNT(1) AS total_trips
12 FROM merged_table
13 )
14
15
16 SELECT ROUND(paid_tips/total_trips*100,2) AS percent_tipped_trips
17 FROM tip_calc_cte
18
19
20
21
22
23
24
25 ).show()

```

▶ (2) Spark Jobs

	percent_tipped_trips
1	70.48

Fig 11: Answer to Business Question 3

Q4) Out of all trips where drivers have received tips, only 3.41% of them have received a tip of over \$10. The customary tipping rule in NYC to taxi driver is 15-20% of the total fare. To receive a tip of \$10, the minimum fare should be around \$60-\$70, which is quite rare considering the rise of alternative Taxi services like Uber, Lyft etc.

```

1 #Question 4
2 spark.sql('''
3
4 WITH tip_calc_cte AS(
5 SELECT
6     SUM (
7         CASE
8             WHEN tip_amount >= 10 THEN 1
9             ELSE 0
10        END) AS paid_tips,
11     COUNT(1) AS total_trips
12 FROM merged_table
13 WHERE tip_amount > 0
14 )
15
16 SELECT ROUND(paid_tips/total_trips*100,2) AS percent_tipped_trips_over_10
17 FROM tip_calc_cte
18
19
20
21
22
23
24
25
26 ).show()

```

▶ (2) Spark Jobs

percent_tipped_trips_over_10
3.41

Command took 3.95 minutes -- by akshaya.parthasarathy@student.uts.edu.au at 29/09/2022, 15:22:25 on bde-at-2

Fig 12: Answer to Business Question 4

Q5) & Q6) Looking at the time buckets, below, it is safe to assume that trips under 5 minutes are the most profitable for the driver, considering that the duration traveled is less and income is maximised.

▶ (2) Spark Jobs

time_buckets	avg_speed_kmh	avg_distance_per_dollar
Under 5	22.85	0.13
Between 5 & 10	18.05	0.18
Between 10 & 20	18.36	0.23
Between 20 & 30	22.44	0.29
Between 30 & 60	27.56	0.35
Atleast 60	23.09	0.4

Command took 5.03 minutes -- by akshaya.parthasarathy@student.uts.edu.au at 29/09/2022, 15:22:29 on bde-at-2

Fig 13: Answer to Business Question 5

However upon further analysis, we can see that a majority of the trips fall under the bracket of 5 to 10 minutes. Even though the average amount earned per km is \$5 lesser than the Under 5 minute bracket, the frequency of these short 5-10 minute trips can produce considerably more income. Although completely unavoidable, drivers can be advised to refuse trips that are greater than 30 minutes or 60 minutes as these would not be as beneficial.

► (2) Spark Jobs

time_buckets	trips_per_time_bucket	avg_speed_km	avg_amount_per_km
Atleast 60	1060946	23.89	7.03
Between 30 & 60	9398571	27.56	8.34
Between 20 & 30	15991285	22.44	10.4
Under 5	18138065	22.85	21.93
Between 10 & 20	46871564	18.36	12.79
Between 5 & 10	51942883	18.85	16.37

Command took 5.06 minutes -- by akshaya.parthasarathy@student.uts.edu.au at 29/09/2022, 15:22:33 on bde-at-2

Fig 14: Answer to Business Question 6

7. Part 3: Modeling

7.1 Feature Transformation

An underlying principle for ML models is “garbage in, garbage out”, which attributes the poor working of a model to the poor cleaning and transformation of data. Since we have extensively cleaned the entire dataset, our next priority is to ensure we select only certain important columns so as to prevent overfitting. Thus, only few columns have been selected due to their impact on the “total_amount”. These include:

VendorID	Passenger Count
RatecodeID	PULocationID
Trip Distance	DOLocationID
Tip Amount	Extra
MTA Tax	Taxi Colour
Tolls Amount	

An easier approach to check the efficacy of the models is to take only a sample size population of the large dataset. This approach allows for experimentation with the features and faster computation. The sample size data can then be split randomly using a 80/20 proportion.

Prior to training the data, the “Taxi_Colour” column has been transformed from categorical to nominal using the StringIndexer() function, where the Yellow cabs are identified as 1 and Green as 0. The dataset was then split into train and test datasets, with the train dataset excluding rows from April 2022.

Using the VectorAssembler() function, the columns of interest or our features were transformed such as that all values in a row of the dataset was contained within one vector, under a column

called “features.” The same process must also be applied for the test dataset which includes the trip data from April 2022.

A new dataset, “features_data”, is created using the combined “features” column and the “total_amount” column.

7.2 Linear Regression Model

The linear regression model is first assembled by identifying the target column as “total_amount” and the features column as “features.” We fit the “features_data” into the model. Using the `RegressionEvaluator()` function, we can evaluate the model’s performance against the test dataset using RMSE as the evaluation metric.

The following observations were made on both a sample size of the data and the entire dataset, with and without using “Taxi_Colour” as one of the features.

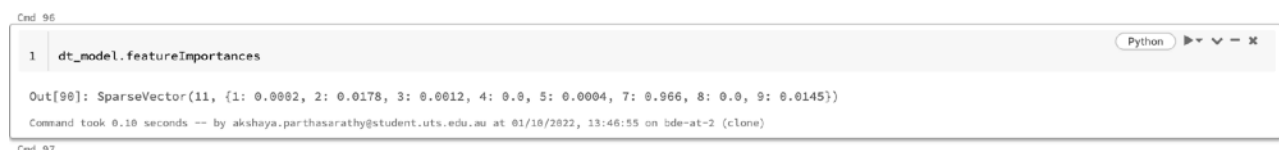
Dataset Type	R^2	RMSE
Sample Size With Taxi Colour	0.950293459145525	2.90975604749161
Sample Size Without Taxi Colour	0.00426084582109232	4.40497822466138
Entire Dataset With Taxi Colour	0.00632995318373431	4.82565214246251
Entire Dataset Without Taxi Colour	0.00632721978275774	4.83128644546773

While the encoded values of taxi colour seem to make a huge impact on a smaller dataset, it fails to create such a significant effect on the larger dataset.

7.3 Decision Tree Regressor

Following the same steps as the previous model, the Decision Tree regressor was fit with the vector transformed features data frame.

The dataset was used with the encoded taxi_colour column. The resulting model yielded an RMSE score of 5.2983. The feature importance vector yielded the following result.



```
Cnd 96
1 dt_model.featureImportances

Out[90]: SparseVector(11, {1: 0.0002, 2: 0.0178, 3: 0.0012, 4: 0.0, 5: 0.0004, 7: 0.966, 8: 0.0, 9: 0.0145})
Command took 0.10 seconds -- by akshaya.parthasarathy@student.uts.edu.au at 01/10/2022, 13:46:55 on bde-at-2 (clone)
Fnd 97
```

Fig 15: Feature Importance of Decision Tree Regressor

8. Conclusion

Databricks Spark provided an integrated approach to perform cleaning, analysis and modelling of the dataset. Analysis and Modeling on a large dataset becomes significantly easier when the cleaning is done with great diligence. An easier approach to verify the accuracy of analysis is to always work on a chunk of data rather than the whole, which proved highly useful in various parts of this project. As for modelling, the Linear Regressor model works better than the Decision Tree Regressor on the given dataset.