Assignment no.2

Student Name: Shaqran Bin Saleh

Student ID: 25010238

94693 Big Data Engineering

TD School

University of Technology Sydney

Contents

Introduction

This project focuses on analyzing a large dataset using Spark. Here the project enabled us to load all the provided data, perform data transformation and analysis on them and finally train it using machine learning models for predicting outcome.

Now the dataset provided is of The New York City Taxi and Limousine Commission (TLC) which is responsible for licensing and regulating New York City's taxi cabs since 1971. The data provided were in parquet files from the years 2015 to 2022 And they were classified based on the taxi color green and yellow.

The project involved several steps starting from uploading all the data to the Azure container, then mounting Azure to the DataBricks environment, then reading the data , cleaning it, analyzing it and finally training it on two machine learning models (Linear Regression and Random Forest Regression) to predict the outcome of the total fare amount.

Methodology

The Data and Cloud Upload

For this project the dataset from the New York City Taxi and Limousine Commission (TLC) was used. There were 16 data files in parquet. The data files were classified according to the taxi cab color and the year of the ride. We had two taxi colors yellow and green and the year of ride ranged from 2015 to 2022. Apart from the 16 parquet data files there was another file in csv format which held the location details and the IDs.

Name	Date modified	Type	Size
green_taxi_2015.parquet	9/24/2024 3:05 PM	PARQUET File	395,075 KB
green_taxi_2016.parquet	9/24/2024 3:05 PM	PARQUET File	338,606 KB
green_taxi_2017.parquet	9/24/2024 3:05 PM	PARQUET File	245,611 KB
green_taxi_2018.parquet	9/24/2024 3:05 PM	PARQUET File	189,331 KB
green_taxi_2019.parquet	9/24/2024 3:05 PM	PARQUET File	138,832 KB
green_taxi_2020.parquet	9/24/2024 3:05 PM	PARQUET File	36,208 KB
green_taxi_2021.parquet	9/24/2024 3:05 PM	PARQUET File	22,930 KB
green_taxi_2022.parquet	9/24/2024 3:05 PM	PARQUET File	19,615 KB
taxi_zone_lookup.csv	9/24/2024 9:58 AM	Microsoft Excel C...	13 KB
yellow_taxi_2015.parquet	9/24/2024 10:00 AM	PARQUET File	2,828,701 KB
yellow_taxi_2016.parquet	9/24/2024 10:07 AM	PARQUET File	2,555,508 KB
yellow_taxi_2017.parquet	9/24/2024 10:15 AM	PARQUET File	2,023,705 KB
yellow_taxi_2018.parquet	9/24/2024 10:15 AM	PARQUET File	2,048,896 KB
yellow_taxi_2019.parquet	9/24/2024 10:15 AM	PARQUET File	1,746,457 KB
yellow_taxi_2020.parquet	9/24/2024 12:58 PM	PARQUET File	521,250 KB
yellow_taxi_2021.parquet	9/24/2024 1:00 PM	PARQUET File	663,030 KB
yellow_taxi_2022.parquet	9/24/2024 1:00 PM	PARQUET File	854,790 KB

Figure 1: The .parquet and .csv data files

So, for this project we had to upload all 17 files to our Microsoft Azure Cloud container for future use of mounting it to the Databricks application.

Name	Modified	Access tier	Archive status	Blob type	Size	Lease state
green_taxi_2015.parquet	9/24/2024, 3:30:15 PM	Hot (Inferred)		Block blob	385.81 MiB	Available
green_taxi_2016.parquet	9/24/2024, 3:33:03 PM	Hot (Inferred)		Block blob	330.67 MiB	Available
green_taxi_2017.parquet	9/24/2024, 3:35:10 PM	Hot (Inferred)		Block blob	239.85 MiB	Available
green_taxi_2018.parquet	9/24/2024, 3:45:26 PM	Hot (Inferred)		Block blob	184.89 MiB	Available
green_taxi_2019.parquet	9/24/2024, 3:45:07 PM	Hot (Inferred)		Block blob	135.58 MiB	Available
green_taxi_2020.parquet	9/24/2024, 3:24:52 PM	Hot (Inferred)		Block blob	35.36 MiB	Available
green_taxi_2021.parquet	9/24/2024, 3:24:05 PM	Hot (Inferred)		Block blob	22.39 MiB	Available
green_taxi_2022.parquet	9/24/2024, 3:23:52 PM	Hot (Inferred)		Block blob	19.16 MiB	Available
taxi_zone_lookup.csv	9/24/2024, 6:31:28 PM	Hot (Inferred)		Block blob	12.03 KiB	Available
yellow_taxi_2015.parquet	9/24/2024, 4:07:32 PM	Hot (Inferred)		Block blob	2.7 GiB	Available
yellow_taxi_2016.parquet	9/24/2024, 4:26:25 PM	Hot (Inferred)		Block blob	2.44 GiB	Available
yellow_taxi_2017.parquet	9/24/2024, 4:41:37 PM	Hot (Inferred)		Block blob	1.93 GiB	Available
yellow_taxi_2018.parquet	9/24/2024, 4:58:10 PM	Hot (Inferred)		Block blob	1.95 GiB	Available
yellow_taxi_2019.parquet	9/24/2024, 5:18:02 PM	Hot (Inferred)		Block blob	1.67 GiB	Available
yellow_taxi_2020.parquet	9/24/2024, 5:30:30 PM	Hot (Inferred)		Block blob	509.03 MiB	Available
yellow_taxi_2021.parquet	9/24/2024, 5:32:12 PM	Hot (Inferred)		Block blob	647.49 MiB	Available
yellow_taxi_2022.parquet	9/24/2024, 5:33:34 PM	Hot (Inferred)		Block blob	834.75 MiB	Available

Figure 2: The .parquet and .csv data files uploaded to Azure Cloud container

All these files were uploaded into the Microsoft Azure container for future use in the time of data ingestion into Databricks The Storage Account Access key is the important credential needed for use in time of mounting the data in Databricks. This was found in the “Access Keys” tab under the storage account.

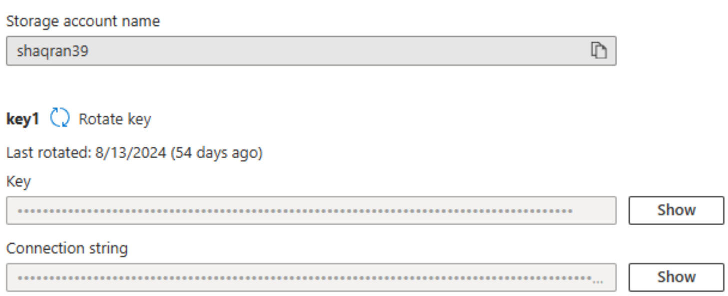


Figure 3: Access Key to Microsoft Azure Container

Azure mounting and Data Ingestion

As we sign in to our Community Edition of Databricks, we create a new notebook under Worksheet. This is the place where all of our codes and queries were written and ran.

The purpose of Microsoft Azure's mounting in the Databricks is to be able to access the data as if they were a part of Databrick's file system (DBFS).

```
storage_account_name = "shaqran39"
storage_account_access_key = "yQTb+AMZogT9ie2qeFSUIyKtrt/iPg7ZRPOiNFvwdboubyr4+cyf2uem9q57uUMO1x7W8O3MNPDR+AStfpVP4g=="
blob_container_name = "bde-assignment2"

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}
)
```

Figure 4: Code to mount Microsoft Azure Container to the Databricks File System (DBFS)

After we have mounted, we read the files and put them in dataframes according to the colors and the csv is loaded to a separate dataframe.

Just so that our processing time is quicker we copy all the files from our mounted directory and copy them into Databrick's File System.

9/28/2024 (45m) 16

```
# Dynamically copy all files that start with 'green_taxi'
files = dbutils.fs.ls("/mnt/bde-assignment2/")
for f in files:
    if f.name.startswith("green_taxi"):
        dbutils.fs.cp(f"/mnt/bde-assignment2/{f.name}", f"/dbfs/FileStore/Assignment2/{f.name}")

# Dynamically copy all files that start with 'yellow_taxi'
for f in files:
    if f.name.startswith("yellow_taxi"):
        dbutils.fs.cp(f"/mnt/bde-assignment2/{f.name}", f"/dbfs/FileStore/Assignment2/{f.name}")
```

9/28/2024 (2s) 17

```
# Copy CSV file from Azure Blob to DBFS
dbutils.fs.cp("/mnt/bde-assignment2/taxi_zone_lookup.csv", "/dbfs/FileStore/Assignment2/taxi_zone_lookup.csv")
```

```
Out[19]: True
```

Figure 5: Code to copy all the files from the mounted directory to a newly made directory inside DBFS

Reading the DBFS data

After the copying is done, we proceed to reading the files and putting them on dataframes and from this step onwards we will be using these dataframes for all the next parts of joining data, cleaning data, performing analysis and finally training the models.

Figure 6: Reading the files to dataframes from the Databrick's File System (DBFS)

After that we count the number of records for the green and yellow taxi dataframes. We find that

- Green taxi dataframe has 66,200,401 records and 20 features
- Yellow taxi dataframe has 663,055,251 records and 19 features

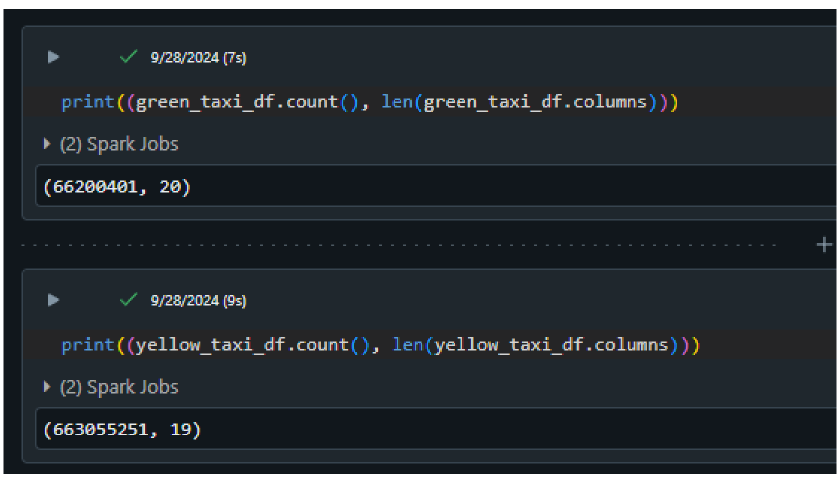


Figure 7: Counting the rows and columns of the tallow and green taxi dataframes

Converting a parquet file to a csv file

According to the project instruction, we were asked to convert the parquet file of green taxi data from the year 2015 and convert that into a csv file.

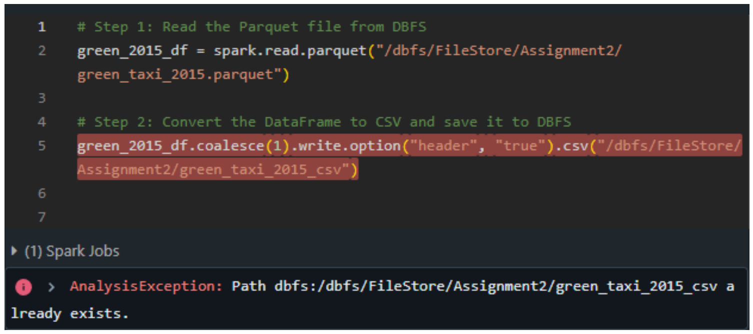


Figure 8: Reading the file and converting it to a csv file

Here, we are reading the saved file from the DBFS and then we are converting it to a csv file. As we convert the parquet file to a csv, the actual csv is found in the directory.

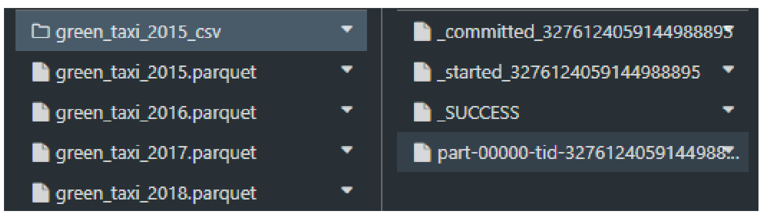


Figure 9: Browsing file directory to find the resultant csv file

After converting it we now check the file size and compare. We see that the newly made csv file has a size of 2078.17 MB where as the parquet file has a size of 385.81 MB.

Figure 10: Finding out the size of the newly made csv file

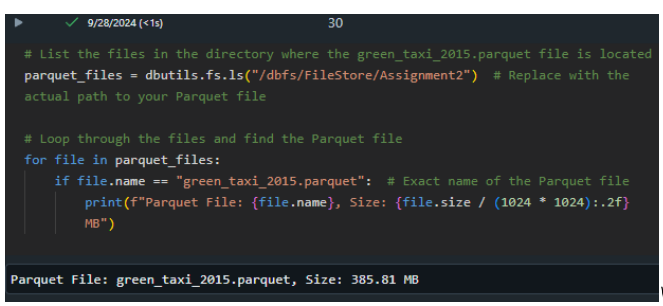


Figure 11: Finding out the size of the newly made csv file

Combining the Dataframes

After we have successfully copied the files in the DBFS and read to the dataframes, the next job is to combine all the data from the green and yellow dataframes. We combine the two green and yellow taxi dataframes using their common columns. Also we added a new column "Taxi_color", depending upon from where the record is from.

```
# Step 1: Rename columns in green_taxi_df to match yellow_taxi_df
green_taxi_df = green_taxi_df.withColumnRenamed("lpep_pickup_datetime",
"tpep_pickup_datetime") \
                             .withColumnRenamed("lpep_dropoff_datetime",
                             "tpep_dropoff_datetime")

# Step 2: Select common columns from both DataFrames to ensure compatibility for
union
common_columns = ['VendorID', 'tpep_pickup_datetime', 'tpep_dropoff_datetime',
'passenger_count',
                  'trip_distance', 'RatecodeID', 'store_and_fwd_flag',
                  'PULocationID', 'DOLocationID',
                  'payment_type', 'fare_amount', 'extra', 'mta_tax', 'tip_amount',
                  'tolls_amount',
                  'improvement_surcharge', 'total_amount', 'congestion_surcharge']

# Step 3: Add the 'Taxi_color' column for yellow_taxi_df and green_taxi_df
yellow_taxi_df_selected = yellow_taxi_df.select(*common_columns).withColumn
("Taxi_color", F.lit("yellow"))
green_taxi_df_selected = green_taxi_df.select(*common_columns).withColumn
("Taxi_color", F.lit("green"))

# Step 4: Union the two DataFrames
combined_taxi_df = yellow_taxi_df_selected.union(green_taxi_df_selected)
```

```
green_taxi_df: pyspark.sql.dataframe.DataFrame = [VendorID: long, tpep_pickup_datetime:
timestamp ... 18 more fields]
yellow_taxi_df_selected: pyspark.sql.dataframe.DataFrame = [VendorID: long,
tpep_pickup_datetime: timestamp ... 17 more fields]
green_taxi_df_selected: pyspark.sql.dataframe.DataFrame = [VendorID: long, tpep_pickup_datetime:
timestamp ... 17 more fields]
combined_taxi_df: pyspark.sql.dataframe.DataFrame = [VendorID: long, tpep_pickup_datetime:
timestamp ... 17 more fields]
```

Figure 12: Combining two dataframes together

After that we are to combine the resultant dataframe to the location dataframe.

```
from pyspark.sql import functions as F

# Step 1: Alias the location_df for pickup (PU) and dropoff (DO) joins
location_df_PU = taxi_zone_lookup_df.alias("PU")
location_df_DO = taxi_zone_lookup_df.alias("DO")

# Step 2: Join the location_df_PU with the combined_taxi_df to add pickup location
information
combined_taxi_df = combined_taxi_df.join(location_df_PU, combined_taxi_df.
PULocationID == location_df_PU.LocationID, how='left') \
                                   .withColumn("PU_Borough", F.col("PU.Borough")) \
                                   .withColumn("PU_Zone", F.col("PU.Zone")) \
                                   .withColumn("PU_service_zone", F.col("PU.
                                   service_zone")) \
                                   .drop("PU.location_ID")

# Step 3: Join the location_df_DO with the combined_taxi_df to add dropoff location
information
combined_taxi_df = combined_taxi_df.join(location_df_DO, combined_taxi_df.
DOLocationID == location_df_DO.LocationID, how='left') \
                                   .withColumn("DO_Borough", F.col("DO.Borough")) \
                                   .withColumn("DO_Zone", F.col("DO.Zone")) \
                                   .withColumn("DO_service_zone", F.col("DO.
                                   service_zone")) \
                                   .drop("DO.location_ID")
```

```
location_df_PU: pyspark.sql.dataframe.DataFrame = [LocationID: integer, Borough: string ... 2 more
fields]
location_df_DO: pyspark.sql.dataframe.DataFrame = [LocationID: integer, Borough: string ... 2 more
fields]
combined_taxi_df: pyspark.sql.dataframe.DataFrame = [VendorID: long, tpep_pickup_datetime:
timestamp ... 31 more fields]
```

Figure 13: Combining the above dataframe to the taxi zone lookup dataframe

The above code combines the combined_taxi_df by joining it with a location lookup table (taxi_zone_lookup_df) twice—once for pickup and once for dropoff locations. It adds details such as borough, zone, and service zone for both the pickup (PU_) and dropoff (DO_) locations, using their respective IDs.

Finally, we drop the redundant columns coming from the location lookup table.

```
4 hours ago (<1s)                36                Python

# Drop unwanted columns that were duplicated after the join
combined_taxi_df = combined_taxi_df.drop("Borough", "Zone", "service_zone",
"LocationID")

combined_taxi_df: pyspark.sql.dataframe.DataFrame = [VendorID: long, tpep_pickup_datetime:
timestamp ... 23 more fields]
```

Figure 14: Dropping the redundant columns from the final dataframe.

Exporting the Combined Dataframe to a parquet file

For quicker computation time and easy data retrieval we are to export the combined dataframe to the Databrick's File System and as a parquet file.

```
combined_taxi_df.write.mode("overwrite").parquet("/dbfs/FileStore/tables/combined_taxi_data.parquet")


(2) Spark Jobs
```

Figure 15: Converting the combined dataframe as a parquet file and putting it in the DBFS

Data Cleaning

The data cleaning is one the most important parts of any data project. The large dataset we had involving the taxi ride records, we needed to make sure all the data were meaningful. For the entire data cleaning part, 6 steps of cleaning were performed:

a. We made sure that no records were there where the trip start time and end time were wrong. We made sure, that there were no records with trip finish time was earlier than trip start time.

```
5 hours ago (<1s)                58

# Step 1: Remove trips where dropoff is before pickup
combined_taxi_df = combined_taxi_df.filter(combined_taxi_df.tpep_dropoff_datetime >= combined_taxi_df.tpep_pickup_datetime)

combined_taxi_df: pyspark.sql.dataframe.DataFrame = [VendorID: long, tpep_pickup_datetime: timestamp ... 23 more fields]
```

Figure 16: Filtering out only the records that have trip drop-off time greater than pickup time

b. We made sure that the pickup and drop-off datetime were within the range. Here we have data from 2015 to 2022. So, all of the data we have needs to be within that time.

```
5 hours ago (<1s)                                                    60
from pyspark.sql.functions import year

# Filter the DataFrame to keep only rows where the pickup and dropoff dates are between 2015 and 2022
# Filter the DataFrame to keep only rows where the pickup and dropoff dates are between 2015 and 2022
combined_taxi_df = combined_taxi_df.filter(
    (year(combined_taxi_df['tpep_pickup_datetime']) >= 2015) &
    (year(combined_taxi_df['tpep_pickup_datetime']) <= 2022) &
    (year(combined_taxi_df['tpep_dropoff_datetime']) >= 2015) &
    (year(combined_taxi_df['tpep_dropoff_datetime']) <= 2022)
)
```

Figure 16: Filtering out only the records that are within the years 2015 to 2022

c. We made sure that there were no records of trips with negative speed. For this we first made a column “speed_mph”, we chose miles as unit of distance as it was an American based data source. So, after making the column we filtered it and removed trip with zero or negative speed.

```
5 hours ago (<1s)                                                    63
combined_taxi_df = combined_taxi_df.withColumn(
    "speed_mph",
    F.col("trip_distance") / F.col("trip_duration_hours")
)
combined_taxi_df: pyspark.sql.dataframe.DataFrame = [VendorID: long, tpep_pickup_datetime: timestamp ... 25 more fields]
```

```
5 hours ago (<1s)                                                    64
# Step 4: Remove trips with negative or zero speed
combined_taxi_df = combined_taxi_df.filter(F.col("speed_mph") > 0)
combined_taxi_df: pyspark.sql.dataframe.DataFrame = [VendorID: long, tpep_pickup_datetime: timestamp ... 25 more fields]
```

Figure 16: Creating a new column speed_mph and then filtering out values which are zero or negative

d. We filtered out trips with very high speed. We first calculated the record with the highest speed. And found that it was 3079127441.7391305 mph.

```
7 hours ago (9m)
max_speed = combined_taxi_df.agg(F.max("speed_mph")).collect()[0][0]

# Display the maximum speed
print(f"The highest speed in the dataset is: {max_speed} mph")
(2) Spark Jobs
The highest speed in the dataset is: 3079127441.7391305 mph
```

Figure 17: Finding out trip with the highest speed

After this we found out that NYC highways has the highest speed of 65mph. So, we filtered out records above 65mph.

```
6 hours ago (<1s)                                                    67
combined_taxi_df = combined_taxi_df.filter(F.col("speed_mph") <= 65)
combined_taxi_df: pyspark.sql.dataframe.DataFrame = [VendorID: long, tpep_pickup_datetime: timestamp ... 25 more fields]
```

Figure 18: Filtering the dataset and keeping only the records with speed equal or below 65mph

e. We filtered out trip which are too short and which are too long(distance wise). Here we will be keeping records which are at least of 0.1 miles and equal or below 100 miles.

```
5 hours ago (<1s)                                   71

# Step 8: Remove trips that are too short or too long (distance-wise)
# Assume trips less than 0.1 miles or greater than 100 miles are unrealistic
combined_taxi_df = combined_taxi_df.filter((F.col("trip_distance") >= 0.1) & (F.col("trip_distance") <= 100))

combined_taxi_df: pyspark.sql.dataframe.DataFrame = [VendorID: long, tpep_pickup_datetime: timestamp ... 25 more fields]
```

Figure 19: Filtering the dataset and keeping only the records with trip distance from 0.1 miles up to 100miles

f. We filtered out trip which are too short and which are too long(duration wise). Here we will be keeping the data which are equal or above 1 min in trip duration and ranges till 2 hours.

```
5 hours ago (<1s)                                   69

# Step 6: Remove trips that are too short or too long (duration-wise)
# Assume trips less than 1 minute or greater than 2 hours are unrealistic
combined_taxi_df = combined_taxi_df.filter((F.col("trip_duration_hours") >= 1/60) & (F.col("trip_duration_hours") < 2))

combined_taxi_df: pyspark.sql.dataframe.DataFrame = [VendorID: long, tpep_pickup_datetime: timestamp ... 25 more fields]
```

Figure 20: Filtering the dataset and keeping only the records with trip duration from 1 minute to up to 2 hours

After cleaning we find that the column had a total of 719568021 rows and 21 features. We will be analyzing this dataset in the next steps and training machine learning models on it.

```
9/28/2024 (4m)

num_rows = combined_taxi_df.count()

# Get the number of columns
num_columns = len(combined_taxi_df.columns)

# Show the dimensions
print(f"Number of rows: {num_rows}")
print(f"Number of columns: {num_columns}")

(2) Spark Jobs

Number of rows: 719568021
Number of columns: 21
```

Figure 21: Counting the total number of rows and columns of the final dataset

Business Questions

Trip Analysis by Year and Month

In this query we had to go through the entire dataset and calculate the total number of trips per month providing insights. The query also had important rows like day of the week, hour of the day with most trips, average number of passengers per trip, average amount paid per trip, average amount paid per passenger.

	year_month	day_of_week	hour_of_day	total_trips	avg_passengers	avg_amount_per_trip	avg_amount_per_passenger
1	2015-01	6	19	172288	1.673157735884101	14.615631616836838	11.854003420636632
2	2015-02	6	19	142926	1.6418986048724515	15.548873053195589	12.732923634145436
3	2015-03	1	0	162497	1.7227087269303434	15.41992166010582	12.202952520009474
4	2015-04	5	19	163641	1.6365825190508492	15.86519723053486	13.056187348124004
5	2015-05	6	19	162412	1.6667118193236954	15.7002132231708	12.707644189884475
6	2015-06	3	19	143043	1.6305376704906915	15.364608684118844	12.666560170061032
7	2015-07	4	19	143727	1.6386204401399875	15.392784862978523	12.626946730032607
8	2015-08	7	23	128046	1.7340565109413804	15.665450072644296	12.3400087325455
9	2015-09	4	19	138661	1.6189844296521734	15.869379998712695	13.100318731781224
10	2015-10	6	19	151167	1.6381022313070974	16.149042052839572	13.147052225595067

Figure 22: Table from business question no1

The table above shows the total number of trips for each month. It indicates the day of the week which had the highest number of trips in each month. The first day 1= Monday and last day of the week is consider 7=Sunday. And the hour of the day shows the hour with the most trips. This is set in a 24hour format.

Trip Duration, Distance, and Speed by Taxi Color (Yellow and Green)

In this query we calculate the average, median, minimum and maximum trip duration, trip distance and speed in kmph for the two colored taxi cabs.

Table

	Taxi_color	avg_trip_duration_min	median_trip_duration_min	min_trip_duration_min	max_trip_duration_min	avg_trip_distance_km	median_trip_distance_km	min_trip_distance_km	max_trip_distance_km	avg_speed_kmph	median_speed_kmph	min_speed_kmph	max_speed_kmph
1	green	13.96	10.72	1	119.98	3.06	1.94	0.1	99.94	20.27	18.36	0.08	104.61
2	yellow	14.41	11.28	1	119.98	3.03	1.7	0.1	99.96	18.73	16.44	0.08	104.61

Figure 23: Table from the business question no2

The analysis of journey length, distance traveled, and speed for yellow and green cabs is shown in this table:

- Yellow taxis had slightly longer trips—14.41 minutes—than green taxis, which have an average travel duration of 13.96 minutes.
- Average Travel Time (min): The average travel time for yellow and green taxis is 11.28 and 10.72 minutes, respectively.
- Maximum Trip Time (min): For both kinds of trips, the maximum time is 119.98 minutes.
- The average trip distance (kilometers) traveled by green taxis is 3.06, which is somewhat greater than the 3.03 kilometers traveled by yellow taxis.

- Minimum and Maximum Trip Distance (km): The range of excursions for both taxi kinds is from 0.1 km to around 99.9 km.
- Yellow taxis average 18.74 km/h, slightly slower than green taxis' average of 20.27 km/h.
- Median Speed (km/h): The median speed for green taxis is greater (18.36 km/h) than for yellow taxis (16.44 km/h).
- Maximum Speed (km/h): The top speed attained by both varieties of taxis was 104.61 km/h.

This information offers a thorough analysis of the variations in journey duration and speed between different colored taxi. Although the maximum trip lengths and distances for both types of taxis are quite comparable, green taxis appear to run at slightly quicker rates on average.

Trips by Pickup and Dropoff Locations (Boroughs)

Through this query, for each pair of pickup and dropoff locations we found out the total number of trips, average distance, average amount paid per trip, total amount paid. Here we used the Borough information for the pickup and dropoff location.

	Taxi_color	PU_Borough	DO_Borough	year_month	day_of_week	hour_of_day	total_trips	avg_distance_km	avg_amount_paid_per_trip	total_amount_paid
1	yellow	Manhattan	Manhattan	2015-01	5	3	22042	2.48	12.75	280999.17
2	yellow	Bronx	Queens	2015-01	5	4	5	11.2	37.46	187.32
3	yellow	Unknown	Manhattan	2015-01	5	6	146	2.41	11.48	1675.96
4	yellow	Brooklyn	Brooklyn	2015-01	5	10	351	2.26	11.99	4208.57
5	yellow	Bronx	Manhattan	2015-01	5	21	22	3.44	16.73	367.96
6	yellow	Brooklyn	EWR	2015-01	6	4	2	19.2	100.6	201.2
7	yellow	Queens	EWR	2015-01	6	12	4	27.96	108.18	432.73
8	yellow	Manhattan	Brooklyn	2015-01	6	21	6089	5.84	26.17	159376.32
9	yellow	Brooklyn	Unknown	2015-01	7	4	4	9.58	30.74	122.96
10	yellow	Unknown	Unknown	2015-01	7	4	714	3.76	18.34	13097.84

Figure 24: Table outcome for business question no 3

This table analyzes yellow and green taxi journeys between several boroughs, providing financial data and trip details:

The starting and finishing boroughs for each trip are indicated by the boroughs PU_Borough (Pickup Borough) and DO_Borough (Dropoff Borough).
Month and Year: Monthly grouping of the data begins in January 2015 (2015-01).
Day of the Week: Shows which day of the week is busiest for each combination of boroughs (1 = Monday, 7 = Sunday).
Hour of the Day: Displays the hour when every pair of pickup and drop-off locations had the most journeys.
Total Trips: The quantity of travels made for every combination of boroughs. For instance, in January 2015, 22,042 journeys were place inside Manhattan.

Percentage of Trips with Tips

Through this query we calculated the percentage of trip where the tips were given. From the final dataset as we analyze we find that the percentage is 63.54%

Table

	.00 percentage_tips
1	63.54

Figure 25: Table outcome for business question no4

Percentage of Trips with Tips of at Least $5

For this query we go through the trips with tips again and calculate the percentage of those where the tip amount was at least 5$.

After going through the data we find that percentage was 12.23%.

Table

	.00 percentage_tips_at_least_5
1	12.23

Figure 26: Table outcome for business question no5

Classifying Trips by Duration and Calculating Metrics

Through this query, trips were classified into bins. The following bins:

- Under 5 minutes
- From 5 to 10 minutes
- From 10 to 20 minutes
- From 20 to 30 minutes
- From 30 to 60 minutes
- At least 60 minutes

For each bin, the following were calculated:

- Average speed (km/h): Helps assess traffic conditions.
- Average distance per dollar (km/$): Indicates fare efficiency.

	duration_bin	avg_speed_kmph	avg_distance_per_dollar_km
1	From 30 to 60 Mins	25.68	0.37
2	At least 60 Mins	22.85	0.53
3	From 10 to 20 Mins	17.72	0.26
4	Under 5 Mins	19.57	0.16
5	From 5 to 10 Mins	17.02	0.21
6	From 20 to 30 Mins	21.25	0.31

Figure 27: Table outcome for business question no6

- Duration Bin: Trips were grouped into 6 duration categories (e.g., "From 30 to 60 Mins," "From 10 to 20 Mins").
- Average Speed (km/h): The average speed of trips within each duration bin. Trips lasting from 30 to 60 minutes have the highest average speed (25.68 km/h), while shorter trips (e.g., under 5 minutes) tend to have lower speeds.
- Average Distance per Dollar (km/$): This measures the fare efficiency by calculating how far the taxi travels per dollar. Longer trips (e.g., trips lasting at least 60 minutes) tend to offer more distance per dollar (0.53 km/$), while shorter trips (e.g., under 5 minutes) have lower fare efficiency (0.16 km/$).

This analysis suggests that longer trips are more cost-effective in terms of distance covered per dollar, while shorter trips are less efficient due to higher base fares. The optimal trip duration for balancing speed and fare efficiency is between 30 to 60 minutes.

Recommended Duration Bin for Maximizing Income

According to the previous business question we find that the optimal trip duration is between 30 to 60 minutes. But as we analyze to find the most rewarding trips, we see that the trips which are at least 60 minutes are the most rewarding.

	duration_bin	median_total_amount
1	From 30 to 60 Mins	42.35
2	At least 60 Mins	64.34
3	From 10 to 20 Mins	14.76
4	Under 5 Mins	6.8
5	From 5 to 10 Mins	9.8
6	From 20 to 30 Mins	23.16

Figure 28: Table outcome for business question no7

Machine Learning

Baseline Model

For this project, the baseline model is a simple predictive model that uses historical averages to predict the total fare (total_amount) for each taxi trip. Instead of considering specific features like trip distance or duration to make predictions, the baseline model simply uses the average fare for similar trips based on factors such as:

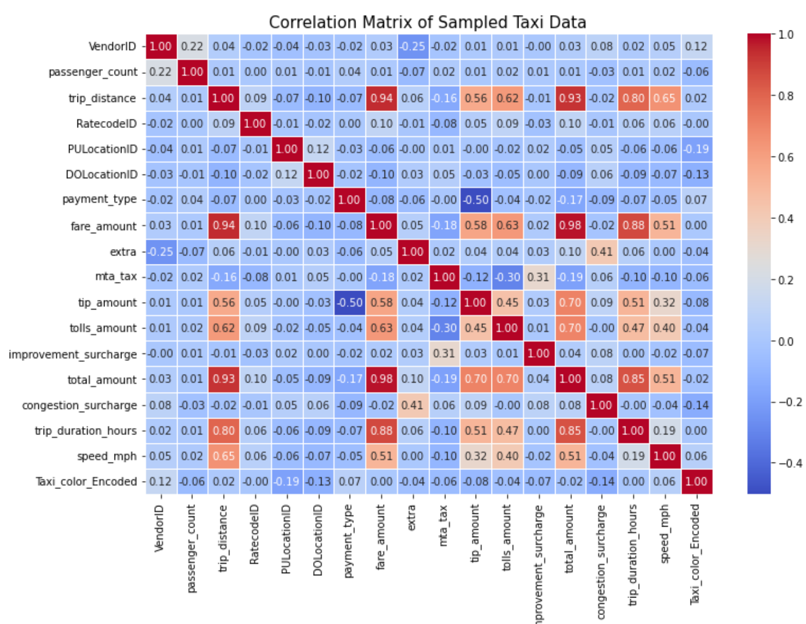
- Taxi color (yellow or green)
- Pickup and dropoff boroughs
- Month of the trip
- Day of the week
- Hour of the day

For every trip in the dataset, the baseline model assigns the average fare from trips with the same taxi color, boroughs, and time factors as the predicted fare (baseline_prediction).If no historical data is available for a specific combination of taxi color, borough, and time, a global average fare is used as the prediction.

The Root Mean Squared Error (RMSE) is computed by comparing the baseline predictions with the actual fare values (total_amount). RMSE provides a measure of the prediction error, where lower RMSE values indicate better performance. We got a RMSE score of 189.5.

Linear and Random Forest Regression

Before we ran the two models into the data. A lot of steps were taken. Since computational time was excessively high and as clusters collapsed and terminated many times, we chose 10000 random records to train the data. And before we trained our data we performed correlation matrix to find the relevant features for training. The following is the correlation matrix:



After that we chose the following as features and target:

```
# Columns to use (excluding 'fare_amount' and 'tolls_amount')
features = ['trip_distance', 'trip_duration_hours', 'tip_amount', 'speed_mph', 'extra',
            'RatecodeID', 'congestion_surcharge', 'improvement_surcharge', 'VendorID', 'passenger_count']

# Target variable
target = 'total_amount'
```

The features were selected based upon the correlation scores with the target variable.

For the training data and testing data we make sure we follow the assignment requirement.

```
train_df = sampdf.filter(~(col('tpep_pickup_datetime').between('2022-10-01', '2022-12-31')))
```

train_df: pyspark.sql.dataframe.DataFrame = [VendorID: long, tpep_pickup_datetime: timestamp ... 26 more fields]

```
# Filter the data for October, November, and December 2022
test_df = combined_taxi_df.filter(col('tpep_pickup_datetime').between('2022-10-01', '2022-12-31'))
```

test_df: pyspark.sql.dataframe.DataFrame = [VendorID: long, tpep_pickup_datetime: timestamp ... 26 more fields]

That is we take all the data for the training data except those from October, November, December from 2022. And for testing data we only take the data from October, November, December of 2022.

After that we tried running the data through spark ML but failed so we tried using sklearn and implemented our models.

- For Linear Regression we got an RMSE score of 5.86
- For Random Forest Regression we got an RMSE score of 5.63

The Root Mean Squared Error (RMSE) values of 5.86 for Linear Regression and 5.6 for Random Forest suggest that both models are performing somewhat similarly, with Random Forest slightly outperforming Linear Regression.

Challenges and Resolutions

In this Big Data project, several technical and operational challenges arose during the analysis of a large taxi trip dataset, which consisted of millions of records. These challenges were primarily due to platform limitations, data size, and infrastructure constraints.

Platform Limitations Due to Free Subscriptions

Challenge: One of the primary challenges was the use of non-paid subscriptions for both Azure and Databricks. As a result, many of the platform features, such as higher compute resources, larger storage capacity, and more advanced analytics tools, were not fully accessible. This significantly limited the scalability of the project and the speed of processing.

Resolution: With the limitations of a free-tier subscription, we attempted to optimize queries and processes to make the best use of the available resources. For example, we carefully chose which portions of the data to analyze first and focused on sampling data where possible to reduce computational strain. However, without access to the full platform capabilities, performance remained a bottleneck.

Long Query Processing Times Due to Large Dataset

Challenge: The dataset contained millions of rows, which led to extremely long query execution times. Basic operations such as filtering, aggregating, and joining the data required significant amounts of processing power and time. The scale of the data overwhelmed the available compute resources, leading to delays in analysis.

Resolution: To mitigate the long processing times, we experimented with data sampling techniques, working with smaller chunks of data initially to test code and logic. This allowed for quicker iterations on queries before running the full dataset. Additionally, we leveraged PySpark optimizations where possible, including caching frequently used DataFrames to minimize repeated computations.

Frequent Cluster Termination and Short Idle Time

Challenge: The Databricks clusters frequently terminated due to inactivity or exceeded execution time limits. The idle time before the clusters automatically terminated was very short, which disrupted long-running queries and batch processes. This frequent downtime significantly affected productivity.

Resolution: We attempted to minimize the occurrence of idle time by scheduling jobs more frequently and closely monitoring cluster usage. Unfortunately, the frequent termination of clusters due to resource constraints limited our ability to efficiently manage long-running queries and processes.

Failure to Write Parquet File to CSV in Azure

Challenge: When attempting to write back the final dataset (which was in Parquet format) to CSV format in the Azure Blob Storage, multiple attempts resulted in write failures. These failures were likely caused by storage limitations or access restrictions in the free-tier Azure account.

Resolution: After several attempts, we explored alternative approaches, such as exporting smaller data chunks instead of the full dataset at once. Unfortunately, with the limited storage capacity and permissions, we were unable to successfully export the complete dataset as CSV. This highlighted the need for an upgraded Azure subscription with more robust storage options and permissions.

Failure to Copy the Final Dataset to DBFS (Databricks File System)

Challenge: A significant issue occurred when trying to copy the final dataset to DBFS (Databricks File System). Despite several attempts, the operation failed, most likely due to the large file size and storage restrictions in Databricks' free-tier environment.

Resolution: To overcome this, we performed the tasks by reading the datafiles again and again, combining and cleaning them on each resetting of the cluster.

Conclusion

This project utilized a large New York City taxi dataset with over 720 million rows to predict taxi fare amounts based on trip-related features such as trip distance, duration, and passenger count. After filtering the data and handling missing values, we developed and evaluated two models: Linear Regression and Random Forest Regression.

The Random Forest model outperformed Linear Regression, achieving an RMSE of 5.6 compared to 5.86 for Linear Regression, indicating that non-linear relationships between variables played a significant role in fare prediction. Despite the slight improvement, further optimizations such as hyperparameter tuning and feature engineering could enhance model performance.

Overall, the project demonstrated effective handling of big data, preprocessing techniques, and model building, providing useful insights into taxi fare prediction in a real-world context.

References

- Katal, M. Wazid, and R. Goudar, "Big Data: Issues, Challenges, Tools and Good Practices," in Proceedings of 2013 Sixth International Conference on Contemporary Computing (IC3), Noida, India, 2013, pp. 404-409. doi: 10.1109/IC3.2013.6612229.
- P. Russom, "Big Data Analytics," TDWI Best Practices Report, Fourth Quarter 2011. Available: https://tdwi.org/research/2011/09/best-practices-report-q4-big-data-analytics.aspx
- X. Wu, X. Zhu, G.-Q. Wu, and W. Ding, "Data Mining with Big Data," IEEE Transactions on Knowledge and Data Engineering, vol. 26, no. 1, pp. 97-107, Jan. 2014. doi: 10.1109/TKDE.2013.109

- H. Hu, Y. Wen, T.-S. Chua, and X. Li, "Toward Scalable Systems for Big Data Analytics: A Technology Tutorial," IEEE Access, vol. 2, pp. 652-687, 2014. doi: 10.1109/ACCESS.2014.2332453
- K. Kambatla, G. Kollias, V. Kumar, and A. Grama, "Trends in Big Data Analytics," Journal of Parallel and Distributed Computing, vol. 74, no. 7, pp. 2561-2573, Jul. 2014. doi: 10.1016/j.jpdc.2014.01.003