

# Big Data Engineering

Data processing on a big dataset with Databricks Spark



Master of Data Science and Innovation  
University of Technology Sydney

Date: 02-10-2022

## Project Overview

**Big data** refers to data sets that are too large or complex to be dealt with by traditional data-processing application software. Data with many attributes offer greater statistical power and may lead to higher complexity. Big data analysis challenges include capturing, storing, analysing, searching, sharing, transferring, visualising, querying, etc.

This project's data is the NYC Taxi data, including the yellow and green cabs. The data will be first stored in Microsoft Azure, cloud storage. Then, it will be analysed, preprocessed and cleaned using Pyspark on Databricks. Finally, two machine learning models will run on that cleaned dataset also using Pyspark on Databricks.

DataBricks is a big data processing platform. DataBricks was founded to provide an alternative to the MapReduce system and provides a just-in-time cloud-based platform for big data processing clients.

## Setup

Here, the entire project works on the connection of Microsoft Azure Blob Storage and Databricks. As a result, a few steps must be completed beforehand to start working with data. In Microsoft Azure,

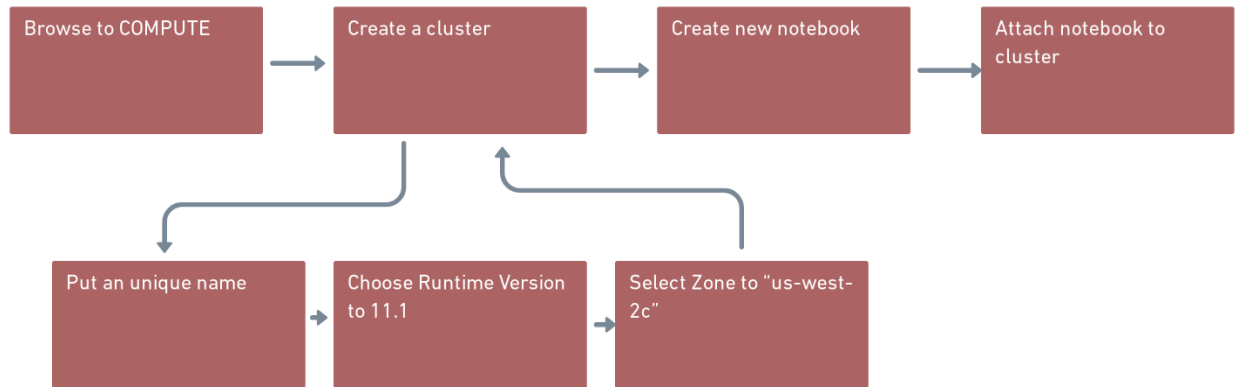
1. Creating a storage account (if not previously made).
2. Creating a container.
3. Uploading all the parquet files in several subfolders.
4. From azure portal home>Storage account>preferred storage account>access keys>key1>show>copy the key.



*Flowchart 1: Setup Steps in Microsoft Azure*

In the Databricks, after the sign in process,

1. Browse to compute from options on the left-hand side.
2. Create a cluster
3. Create a new notebook
4. Attach cluster to notebook from the top-left corner.



Flowchart 2: Setup Steps in Databricks

## Dataset Exploration

For this project, we have parquet type data that describes the taxi fare of NYC (New York City). Both of these are time-series data. They differ in colour (yellow and green). As yellow and green taxis operate in distinct areas of NYC.

In the yellow data, there are 19 columns. It shows the records from the southern part of Manhattan. The data dictionary is as follows:

Column Name	Description
VendorID	A code indicating the TPEP provider that provided the record. <b>1= Creative Mobile Technologies, LLC; 2= VeriFone Inc.</b>
tpep_pickup_datetime	The date and time when the meter was engaged.
tpep_dropoff_datetime	The date and time when the meter was disengaged.
Passenger_count	The number of passengers in the vehicle. This is a driver-entered value.
Trip_distance	The elapsed trip distance in miles reported by the taximeter.
PULocationID	TLC Taxi Zone in which the taximeter was engaged.
DOLocationID	TLC Taxi Zone in which the taximeter was disengaged.
RateCodeID	The final rate code in effect at the end of the trip. <b>1= Standard rate, 2=JFK, 3=Newark, 4=Nassau or Westchester, 5=Negotiated fare, 6=Group ride.</b>
Store_and_fwd_flag	This flag indicates whether the trip record was held in vehicle memory before sending to the vendor, aka "store and forward," because the vehicle did not have a connection to the server. <b>Y= store and forward trip, N= not a store and forward trip.</b>
Payment_type	A numeric code signifying how the passenger paid for the trip. <b>1= Credit card, 2= Cash, 3= No charge, 4= Dispute, 5= Unknown, 6= Voided trip.</b>
Fare_amount	The time-and-distance fare calculated by the meter.
Extra	Miscellaneous extras and surcharges. Currently, this only includes the \$0.50 and \$1 rush hour and overnight charges.
MTA_tax	\$0.50 MTA tax that is automatically triggered based on the metered rate in use

Improvement_surcharge	\$0.30 improvement surcharge assessed trips at the flag drop. The improvement surcharge began being levied in 2015.
Tip_amount	Tip amount – This field is automatically populated for credit card tips. Cash tips are not included.
Tolls_amount	Total amount of all tolls paid in trip.
Total_amount	The total amount charged to passengers. Does not include cash tips.
Congestion_Surcharge	Total amount collected in trip for NYS congestion surcharge.
Airport_fee	\$1.25 for pick up only at LaGuardia and John F. Kennedy Airports

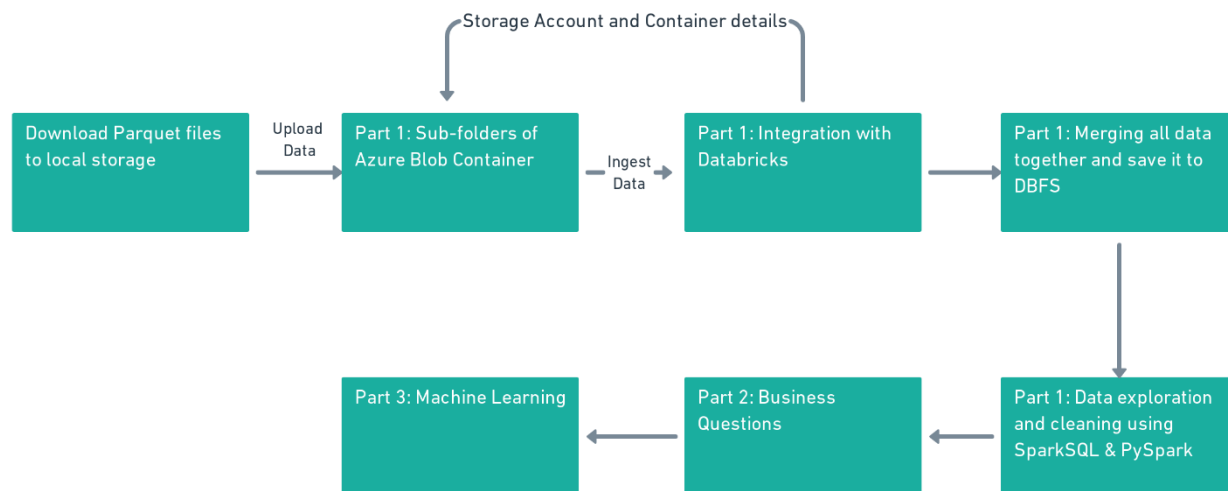
*Table 1: Data Dictionary of yellow taxis.*

The green dataset keeps the records from all the places other than the southern Manhattan. There are 20 columns in this dataset. The data dictionary is shown below.

Column Name	Description
VendorID	A code indicating the LPEP provider that provided the record. <b>1= Creative Mobile Technologies, LLC; 2= VeriFone Inc.</b>
lpep_pickup_datetime	The date and time when the meter was engaged.
lpep_dropoff_datetime	The date and time when the meter was disengaged.
Passenger_count	The number of passengers in the vehicle. This is a driver-entered value.
Trip_distance	The elapsed trip distance in miles reported by the taximeter.
PULocationID	TLC Taxi Zone in which the taximeter was engaged.
DOLocationID	TLC Taxi Zone in which the taximeter was disengaged.
RateCodeID	The final rate code in effect at the end of the trip. <b>1= Standard rate, 2=JFK, 3=Newark, 4=Nassau or Westchester, 5=Negotiated fare, 6=Group ride.</b>
Store_and_fwd_flag	This flag indicates whether the trip record was held in vehicle memory before sending to the vendor, aka “store and forward,” because the vehicle did not have a connection to the server. <b>Y= store and forward trip, N= not a store and forward trip.</b>
Payment_type	A numeric code signifying how the passenger paid for the trip. <b>1= Credit card, 2= Cash, 3= No charge, 4= Dispute, 5= Unknown, 6= Voided trip.</b>
Fare_amount	The time-and-distance fare calculated by the meter.
Extra	Miscellaneous extras and surcharges. Currently, this only includes the \$0.50 and \$1 rush hour and overnight charges.
MTA_tax	\$0.50 MTA tax that is automatically triggered based on the metered rate in use
Improvement_surcharge	\$0.30 improvement surcharge assessed on hailed trips at the flag drop. The improvement surcharge began being levied in 2015.
Tip_amount	Tip amount – This field is automatically populated for credit card tips. Cash tips are not included.
Tolls_amount	Total amount of all tolls paid in trip.
Total_amount	The total amount charged to passengers. Does not include cash tips.

Trip_type	A code indicating whether the trip was a street-hail or a dispatch that is automatically assigned based on the metered rate in use but can be altered by the driver. <b>1= Street-hail, 2= Dispatch.</b>
Ehail_fee	A significant benefit of the E-hail program is that our customers are not expected to pay any out-of-pocket expenses other than the <b>\$2.75 transit fare</b> for themselves and their guests

## Workflow Diagram



*Flowchart 3: Workflow Diagram*

The entire workflow of this process consists of several parts. At first, the data must be downloaded.

Second, upload the data in Microsoft Azure into several subfolders to access it from a remote location or SaaS platform.

Third, ingest the data into Databricks.

After that, create tables, clean, and analyse, answer the business questions through the existing data in hand. Furthermore, run machine learning algorithms on the dataset with performance matrix (RMSE).

## Part 1: Data Ingestion and Preparation

1. **Download Datasets:** In this part, all the datasets that are required (From 2019 to April 2022) was downloaded from the official website of NYC (<https://www1.nyc.gov/>). Then in the storage account of Azure, a new container was built to push the data.
2. **Read and copy data into DBFS:** There are total 40+40 = 80 items as parquet files, uploaded in Azure. All the 80 files were read and merged to work on both the dataset at once. For these parts, there were several subfolders in the azure. Despite having a smaller size for parquet files, the files were still large to import for the databricks community edition. As a result, several folders were made inside the container to preserve the yellow data. However, for

green data, no such measure was necessary. In addition, there was a disparity between the yellow and green data schema. One column (Airport Fee) was integer in some datafiles and double in others. This issue was hindering the dataset from merging for later purposes. As a solution, the integer types were typecasted to double while ingesting into DBFS.

Authentication method: Access key (Switch to Azure AD User Account)


Location: bde-at2 / yellow

Search blobs by prefix (case-sensitive)


+ Add filter

Name


☐

 [.]


☐

 double1


☐

 double2


☐

 double3


☐

 double4


☐

 double5


☐

 int1


☐

 int2


☐

 int3

☐

 int4

☐

 int5

Authentication method: Access key (Switch to Azure AD User Account)


Location: bde-at2 / green

Search blobs by prefix (case-sensitive)


+ Add filter

Name


☐

 [.]


☐

 green\_tripdata\_2019-01.parquet


☐

 green\_tripdata\_2019-02.parquet


☐

 green\_tripdata\_2019-03.parquet


☐

 green\_tripdata\_2019-04.parquet


☐

 green\_tripdata\_2019-05.parquet


☐

 green\_tripdata\_2019-06.parquet


☐

 green\_tripdata\_2019-07.parquet


☐

 green\_tripdata\_2019-08.parquet

☐

 green\_tripdata\_2019-09.parquet

☐

 green\_tripdata\_2019-10.parquet

☐


 green\_tripdata\_2019-11.parquet

Figure 1: Yellow and Green Files in Microsoft Azure Blob Storage.

3. **Total number of rows:** For both tables, the number of rows has been measured. For the yellow table, the number of rows was 152823008 and 9390483 for the green.

<pre>yellow.count()</pre> <p>► (2) Spark Jobs</p> <p>Out[19]: 152823008</p> <p>Command took 5.33 seconds -- by mostafa.m</p>	<pre>green = spark.read.parquet('/dbfs/green') green.count()</pre> <p>► (3) Spark Jobs</p> <p>► green: pyspark.sql.dataframe.DataFrame = [VendorID</p> <p>Out[12]: 9390483</p> <p>Command took 2.16 seconds -- by mostafa.m.jalal@stude</p>
--	---

Figure 2: Yellow and Green Count.

4. **File Size Comparison** In this part, the data file for April 2022 was converted to a CSV File. Parquet files can store the same data as CSVs and take fewer spaces. In Azure, having the same data, the

parquet file takes 52.66 MB while the CSV takes 398.83 MB. Parquet file takes 7.57X less space than CSV. This is why saving data in a parquet file than CSV is more reasonable.

5. **Removing unrealistic trips:**

- a. In this step, the trips were identified for both the tables where the pickup\_datetime is later than dropoff\_datetime. It should always be the other way. So, with these criteria, the yellow table had 65214 and the green had 398 records, which have been deleted.
- b. In this step, 11440 records from the yellow table and 19527 records from the green tables were identified where there was negative trip distance, hence negative speed. For clean data, these records are discarded from the calculation.
- c. There is a speed limit in NYC. Inside NYC, where the yellow cab is available, the speed should not exceed 25 MPH; outside NYC, where green taxis are available, they are instructed not to cross the speed limit of 55 MPH. Consequently, records were identified in this step where the yellow cab's speed exceeded 25 MPH, and the green cabs exceeded 55 MPH. For yellow cabs, there were 7221637 records with 20862 records from the green. These records have been kept aside from further calculation.
- d. This step identified and deleted too short or too long trips (duration wise) as yellow cabs' coverage is not as much as green cabs. The green cabs' upper bound has been set to 6 hours with a lower bound of 1 minute. Similarly, for yellow cabs, the upper bound is 4 hours, and the lower bound is 1 minute. No person can travel much within 1 minute. As a result, the lower bound is 1 minute. Moreover, with the farthest distance round trip, the green cab should not take more than 6 hours, and yellow cabs should not take 4 hours.
- e. This step finds the unrealistically long or short trip distance-wise. The lower threshold is considered as .5 miles for both yellow and green cabs. However, they differ in upper threshold as the round trip from the two furthest corners for the green cab is nearly 48 miles, and for the yellow, it is 22 miles. Here the upper threshold for green is 25, and for yellow, it is 50 miles. Any datapoint out of this bound has been considered invalid.
- f. There are several logics that have been applied to this project to clean the data. There are 11 steps that have been taken into account for both the green and yellow table. Affected attributes are
  - i. **Records outside the selected time frame:** There are data where the time of starting the trip is before 2019, January 01, and past 2022, April 30. In this project, these records do not interest us. As a result, such records have been removed.
  - ii. **Excessive passenger count:** The maximum amount of people that can travel simultaneously in NYC is 5. Any record exceeding this passenger count of 5 has been excluded.
  - iii. **Invalid payment type:** There are six payment types in both yellow and green taxis. Any record out of this bound is removed from datasets.
  - iv. **Invalid Amounts:** There are several columns in these datasets that contains different amounts of charges along the trip. My understanding is, no amount can be minus. I have searched on the internet and found that none of these amounts can be used as a minus value for a discount or any other reason. So, since there is a number, the driver has put the number deliberately. For pressing Zero, he should only press one key. However, when there is a minus double value, it may be possible that the driver has put the minus without his knowledge or by

mistake. As a result, in this analysis, I have converted all the amounts if that is minus. Affected columns in these operations are as follows:

1. Fair\_amount
  2. Extra
  3. MTA\_Tax
  4. Improvement\_surcharge
  5. Tip\_amount
  6. Tolls\_amount
  7. Total\_amount
  8. Congestion\_surcharge
6. **Merging yellow and green:** Here, both the yellow and green data were appended. Since their schema do not match, at first a new unified schema was created and then their values has been inserted into the schema.

```
%sql
CREATE OR REPLACE TABLE yellow_green_merged1
(
  VendorID int,
  cab_type int,
  pickup_datetime timestamp,
  dropoff_datetime timestamp,
  passenger_count int,
  trip_distance double,
  RatecodeID int,
  store_and_fwd_flag string,
  PULocationID int,
  DOLocationID int,
  payment_type int,
  fare_amount double,
  extra double,
  mta_tax double,
  tip_amount double,
  tolls_amount double,
  improvement_surcharge double,
  total_amount double,
  congestion_surcharge double,
  airport_fee double,
  ehail_fee double, trip_type double)

▶ (3) Spark Jobs

OK

Command took 5.11 seconds -- by mostafa.m.jalal@student.uts.edu
```

Figure 3: Creation of unified schema for yellow and green table



```

%sql

--CREATE OR REPLACE TABLE yellow_green_merged AS
--SELECT * FROM
INSERT INTO yellow_green_merged1
(
SELECT
    VendorID,
    1 as cab_type, -- 1 = yellow
    tpep_pickup_datetime as pickup_datetime,
    tpep_dropoff_datetime as 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,
    airport_fee,
    NULL as ehail_fee,
    NULL as trip_type
FROM yellow_clean_table
UNION ALL
SELECT
    VendorID,
    2 as cab_type, -- 2 = green
    lpep_pickup_datetime as pickup_datetime,
    lpep_dropoff_datetime as 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,
    NULL AS airport_fee,
    ehail_fee,
    trip_type
FROM green_clean_table
)

```

Figure 4: Merging yellow and green table

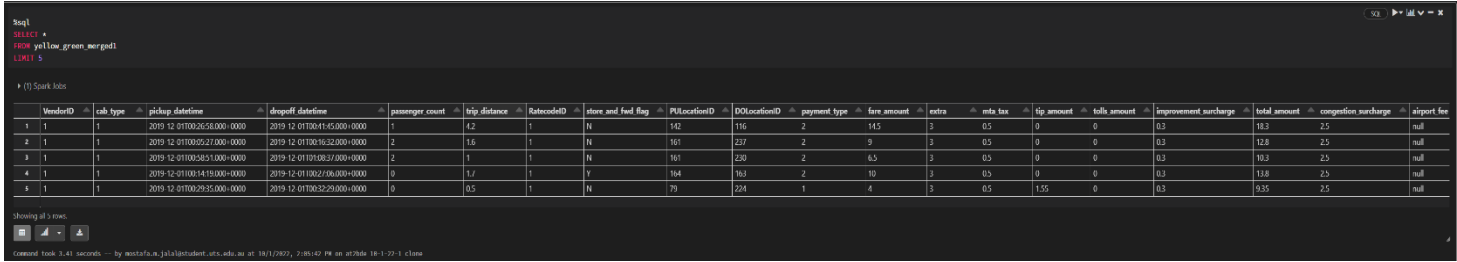


Figure 5: Output of merged table.

7. Later this part, the data was saved into DBFS and later used from DBFS.

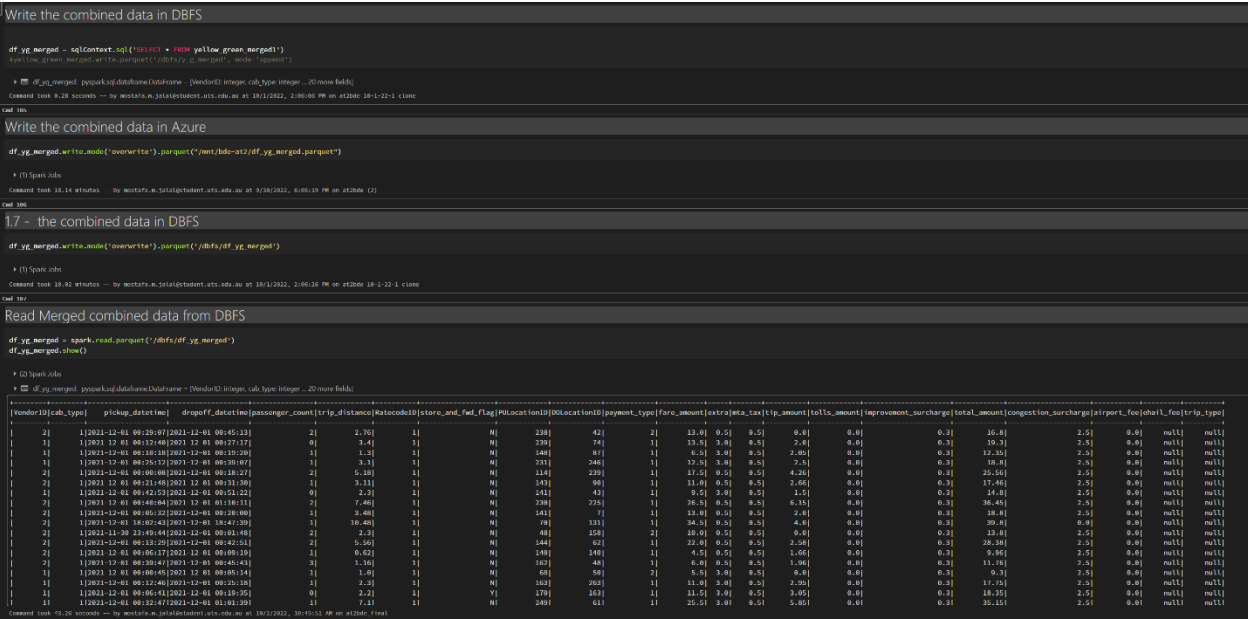


Figure 6: Pushing merged data into DBFS and reading into DBFS.

## Part 2: Business Questions

1. In this part, several insights have been brought out using SparkSQL including, for each month, trip numbers, day of week having most trips, hour of day having most trips, average number of passengers, average amount paid for trips and average amount paid by the passengers. The result is shown below:

	year_month ▲	trip_num ▲	day_of_month ▲	hour_of_month ▲	avg_passener ▲	avg_paid_amount ▲	avg_payment_by_passener ▲
1	Apr 2019	7084397	Tuesday	18	1.44	18.2	18.34
2	Apr 2020	208383	Wednesday	15	1.14	15.05	15.87
3	Apr 2021	1900474	Friday	14	1.3	17.06	17.42
4	Apr 2022	3173656	Friday	18	1.35	19.29	19.34
5	Aug 2019	5695971	Thursday	18	1.44	18.32	18.48
6	Aug 2020	876016	Monday	15	1.27	16.78	17.44
7	Aug 2021	2410636	Tuesday	18	1.35	18.05	18.29
8	Dec 2019	6438552	Tuesday	18	1.42	18.51	18.77
9	Dec 2020	1285708	Tuesday	15	1.29	16.17	16.64
10	Dec 2021	2841295	Thursday	15	1.36	19.03	19.19
11	Feb 2019	6731150	Friday	18	1.43	17.67	17.77
12	Feb 2020	5877874	Saturday	18	1.39	17.44	17.68
13	Feb 2021	1190948	Friday	15	1.29	16.16	16.55
14	Feb 2022	2633430	Saturday	18	1.32	17.97	18.02
15	Jan 2019	7282397	Thursday	18	1.42	14.65	14.73
16	Jan 2020	5952571	Friday	18	1.39	17.19	17.52
17	Jan 2021	1179549	Friday	15	1.28	15.74	16.22
18	Jan 2022	2140459	Friday	17	1.32	16.85	16.9
19	Jul 2019	5929314	Wednesday	18	1.44	18.26	18.38
20	Jul 2020	690534	Thursday	15	1.23	16.55	17.26
21	Jul 2021	2449691	Thursday	18	1.37	17.92	18.15
22	Jun 2019	6603213	Saturday	18	1.44	18.72	18.82
23	Jun 2020	476131	Tuesday	15	1.2	16.59	17.47
24	Jun 2021	2506107	Wednesday	18	1.34	18	18.23
25	Mar 2019	7496582	Friday	18	1.44	18.08	18.18
26	Mar 2020	2799114	Tuesday	18	1.35	17.21	17.5
27	Mar 2021	1681412	Wednesday	15	1.29	16.28	16.67
28	Mar 2022	3195030	Thursday	18	1.32	18.76	18.81
29	May 2019	7193706	Thursday	18	1.43	18.63	18.73
30	May 2020	291968	Friday	15	1.12	16	17.53
31	May 2021	2200866	Saturday	15	1.32	17.27	17.56
32	Nov 2019	6426904	Friday	18	1.42	18.32	18.52
33	Nov 2020	1336363	Monday	15	1.29	16.24	16.72
34	Nov 2021	3092576	Tuesday	18	1.34	19.11	19.32
35	Oct 2019	6776756	Thursday	18	1.41	18.67	18.91
36	Oct 2020	1492050	Thursday	15	1.3	17.06	17.54
37	Oct 2021	3084451	Friday	18	1.35	18.81	19.03
38	Sep 2019	6170578	Thursday	18	1.42	18.79	18.96
39	Sep 2020	1186597	Wednesday	15	1.29	16.44	16.98
40	Sep 2021	2590030	Thursday	18	1.34	19.19	19.44

Showing all 40 rows.

Command took 7.66 minutes -- by mostafa.m.jalal@student.uts.edu.au at 18/2/2022, 8:16:47 PM on at2bde\_final

Figure 7: Result of Business question 1

2. Here, In a single table, for both yellow and green vehicles
  - a. the average, median, minimum and maximum trip duration
  - b. the average, median, minimum and maximum trip distance in km
  - c. the average, median, minimum and maximum speed in km per hour have been identified.

-- 1 means yellow  
-- 2 means green

▶ (2) Spark Jobs

	cab type	avg trip duration	median trip duration	minimum trip duration	maximum trip duration	avg distance km	median distance km	minimum distance km	maximum distance km	avg speed	median speed	minimum speed	maximum speed
1	1	14.02	11.08	1.2	239.98	4.24	2.74	0.81	40.25	17.5	16.2	0.21	40.25
2	2	17.45	13.07	1	359.9	6.47	3.78	0.81	80.5	20.65	18.06	0.14	88.55

Showing all 2 rows.

Command took 5.33 minutes ... by mactata.m.jala@student.uts.edu.au at 18/2/2022, 1:23:48 PM on at3bde\_final

Figure 8: Result of 2.2

- Percentage of trips where drivers received tips is shown here.

▶ (5) Spark Jobs

	trips_with_tips
1	70.13

Showing all 1 rows.

Figure 9: Percentage of drivers receiving tips

- Percentage of drivers receiving tips at least \$10 among the pool of tips receiving drivers.

	trips_with_tips_at_least_10
1	2.12

Showing all 1 rows.

Figure 10: percentage where the driver received tips of at least \$10

- Average speed and average distance per dollar with associated bins.

	bins ▲	AVG_SPEED ▲	avg_distance_per_dollar ▲
1	under 5 minutes	20.56	0.14
2	from 5 minutes to 10 minutes	16.77	0.17
3	from 30 minutes to 60 minutes	22.47	0.34
4	from 20 minutes to 30 minutes	18.28	0.27
5	from 10 minutes to 20 minutes	16.44	0.22
6	at least 60 minutes	22.34	0.43

Showing all 6 rows.

Command took 1.74 minutes -- by mostafa.m.jalal@student.uts.edu.au at 9/30/2022, 11:19:00 AM

Figure 11: avg speed and avg distance per dollar with bins

- According to the previous image (figure 11), if a driver wants to maximize his income, he should focus on targeting trips with minimal duration. For example, if the trip duration is less than 5 minutes, average distance crossed per dollar is 0.14. As a result, the fuel cost is minimal in this scenario. Also, if we look at the speed, the speed in this bin is 20.56, which is also good. This is the most profitable zone. If we extend our boundary a bit much, trips within 5-10 minutes as not bad as well. Avg distance per dollar is 0.17, not increasing much from the previous bin. However, the speed reduces in this bin. Even though the drive will not be able to take much trips like the previous bin in this timeframe. Looking at the other bins, the more the time increases, the more the distance per dollar increases. This is not profitable for the driver. The best bin to maximize the profit is under 5 minutes. Also, 5-10 minute is not bad. Otherwise either the speed decreases or the avg distance per dollar increases.

### **Part 3: Machine Learning**

In this stage, 2 machine learning models have been implemented on the dataset. Here only a few columns were used. For the ease of computation, I have broken down the elements of the pickup\_datetime column. From that column I have generated year, month, date, day, hour. In addition, I also calculated trip\_duration from the pickup\_datetime and dropoff\_datetime.

The list of predictors for the machine learning model are as follows

1. Cab\_type
2. Trip\_distance
3. Year
4. Month

5. Day
6. Hour
7. Trip\_duration

Train data: all data from 2019 – March 2022

Test Data: data from April 2022

Target Feature: Fare\_amount

For Random Forest,

fare_amount	prediction
9.5	9.30042550883837
7.5	8.021208808876331
10.5	10.522647828383402
10.5	10.522647828383402
8.5	8.303206397763384
4.0	6.234951014635998
6.5	6.701071926922529
11.5	11.79434993841566
8.0	8.686301691635203
13.5	13.909199938444775
5.0	6.417601121110581
5.0	6.471641997819766
6.5	7.019329321968698
12.5	12.17488478287189
8.5	8.16646560626565
6.0	6.47148387358948
8.0	8.133591598410526
4.5	6.324150011509952

*Figure 12: Prediction of fare\_amount with Random Forest Regressor*

## RMSE on Trainset for Random Forest

```
evaluator = RegressionEvaluator(  
    labelCol="fare_amount", predictionCol="prediction", metricName="rmse")  
rmse = evaluator.evaluate(rf_train_preds)  
print("Root Mean Squared Error (RMSE) on training data = %g" % rmse)
```

► (1) Spark Jobs

Root Mean Squared Error (RMSE) on training data = 159.629

Command took 15.32 minutes -- by mostafa.m.jalal@student.uts.edu.au at 10/1/2022, 6:46:34 PM on at2bc

*Figure 13: RMSE on the trainset with Random Forest Regressor*

```
evaluator = RegressionEvaluator(  
    labelCol="fare_amount", predictionCol="prediction", metricName="rmse")  
rmse = evaluator.evaluate(rf_test_preds)  
print("Root Mean Squared Error (RMSE) on test data = %g" % rmse)
```

► (1) Spark Jobs

Root Mean Squared Error (RMSE) on test data = 4.63796

Command took 5.24 minutes -- by mostafa.m.jalal@student.uts.edu.au at 10/1/2022, 6:46:47 PM on

*Figure 14: RMSE on the testset with Random Forest Regressor.*

For Generalized Linear Model,

fare_amount	prediction
9.5	10.257891664975546
7.5	8.626458089791129
10.5	10.148164893324122
10.5	10.60884851466443
8.5	9.436740285427561
4.0	5.782950976183201
6.5	7.816015856355534
11.5	13.638448003403596
8.0	8.93979468981189
13.5	14.233126168301908
5.0	6.446860728664319
5.0	6.90806825622758
6.5	7.927961975308051
12.5	11.51012554644899
8.5	7.939956673088659
6.0	7.300632450704541
8.0	9.600414555097984
4.5	6.384699468367842

Figure 14: Prediction of fare\_amount with Generalized Linear Model

```
RMSE on trainset for GLR

evaluator = RegressionEvaluator(
    labelCol="fare_amount", predictionCol="prediction", metricName="rmse")
rmse = evaluator.evaluate(glr_train_preds)
print("Root Mean Squared Error (RMSE) on train data = %g" % rmse)

▶ (1) Spark Jobs

Root Mean Squared Error (RMSE) on train data = 159.637

Command took 9.30 minutes -- by mostafa.m.jalal@student.uts.edu.au at 10/2/2022, 11:22:13 AM on at2bd
```

Figure 15: RMSE on the trainset with Generalized Linear Model



## RMSE on testset for GLR

```
evaluator = RegressionEvaluator(  
    labelCol="fare_amount", predictionCol="prediction", metricName="rmse")  
rmse = evaluator.evaluate(glr_test_preds)  
print("Root Mean Squared Error (RMSE) on test data = %g" % rmse)
```

► (1) Spark Jobs

Root Mean Squared Error (RMSE) on test data = 4.91407

Command took 5.17 minutes -- by mostafa.m.jalal@student.uts.edu.au at 10/2/2022, 11:22:17 AM

*Figure 16: RMSE on the testset with Generalized Linear Model*

For the pictures above, we can see that the Training RMSE for Random Forest Regressor is 159.629 whereas it is 159.637 for Generalized Linear Model. Both the models performed almost similarly on the training dataset. But the performance of Random Forest is a bit better than GLM.

For the testset, RMSE for Random Forest is 4.63 whereas the RMSE for GLM is 4.91. Again, for the testset, the random forest works better.

**Reason to choose Random Forest Regressor:** Random Forest Regressor is useful to solve business problems where it is required to predict continuous values such as prices/costs. It uses many classifying decision trees on various sub samples of the dataset and uses averaging to improve predictive accuracy and control over-fitting. Simply put, this is an useful algorithm to control overfitting and predict continuous values. Because of these reasons, Random Forest Regression has been the model of choice for this project.

## Issues

1. Combining the yellow and green table together because of the disparity in datatype for airport\_fee in both type.  
To solve this issue, I have tried myself to solve it by browsing on the internet. After some try, I consulted with two of my classmates then we talked to our teacher for the help.
2. I was unable to use multiple CTE and use them. Through this project, from business question 2.1, I learnt to use multiple CTEs and join them to get desired output.
3. I faced problem while running the ML model as the one hot encoding was not working properly. As a result, I had to get back to the table merging and change the string type column into integer type while creating the schema. This way, the problem was solved.