

# **New York Taxi Fare Amount Model**

Name: Wenying Wu

Date: 26/04/2022

## OBJECTIVE

The main objective of this project is to analyse the total fare amount in New York City Taxi based on data collected by The New York City Taxi and Limousine Commission (TLC). The project focuses on analysing the data from January 2019 to July 2021 using Spark. This report will present the process of loading, transforming and analysing data. Furthermore, the selection from multiple machine learning models is made for predicting a total amount of a trip.

## DATA DESCRIPTION

The raw data used are published by the TLC, the agency responsible for licensing and regulating New York City's taxi cabs. The data is separated by year, month, and type (yellow/green/...), and is available as a comma-separated value (CSV) file. Each row record one trip information for each trip. Below is a brief summary information of the dataset used in this project:

1. The data used is focused on street hail livery (green) taxis and New York City's medallion (yellow) taxis.
2. The data include trips from January 2019 to July 2021.
3. 62 CSV files included (12.23 GB), 1 file for each year, month, and type.
4. There are 8,348,567 trip records for green taxis and 124,048,218 trip records for yellow taxis.

Although TLC provides detailed data dictionaries with descriptions of the features, the trip data was collected and provided to the TLC by technology service providers. Hence the data accuracy cannot be guaranteed. For example, the taxi trip data were captured by taxi meters, which were subject to multiple sources of errors, for instance, hardware and software malfunction, wireless signal issue, and human interference. In addition, the number of passengers is hand-input by drivers from the data dictionary, which is prone to human errors.

## DATA LOADING

Loading data to Databricks includes five steps and each step will be briefly explained in Appendix 1 and the details are shown in the notebook.

## DATA EXPLORATION

### Data Structure

The green taxi data has 8,348,567 rows and 20 columns, while the yellow taxi data has 124,048,218 rows and 18 columns. Below are some findings on the overall data structure:

- The first point is that columns in green and yellow taxi data are inconsistent.
- The date when the meter was engaged is recorded as "lpep\_pickup\_datetime" in green taxi data and recorded as "tpep\_pickup\_datetime" in yellow taxi data. They are changed to "pickup\_datetime" at the beginning of the analysis for consistency.
- The date and time when the meter disengaged is recorded as "lpep\_dropoff\_datetime" in green taxi data and recorded as "tpep\_dropoff\_datetime" in yellow taxi data are changed to "dropoff\_datetime" at the beginning of the analysis for consistency.

- “ehail\_fee” and “trip\_type” are green taxi data but are not included in yellow taxi data.

## **Data cleaning**

Pyspark is used in this report to perform any required data cleaning to remove unrealistic trips, including illogical, extreme, and null values. There are five steps in transforming raw data into valuable features to get the most out of the data. Please see appendix 3 for details.

## **Combine data**

After the above cleaning processes, 18.87 % of records were dropped in yellow taxi data, and 32.71 % were dropped in green taxi data.

All data are combined into one data frame for the business questions and modelling. Two features, “ehail\_fee”, and “trip\_type” contained in green taxi data, were added to yellow taxi data. “ehail\_fee” is filled with 0, and “trip\_type” is filled with “no” for yellow taxi data. The total number of combined cleaned data is 106,262,678. This project uses 2019 and 2020 to train and validate the model and predict the total amount in 2021. Therefore, there are 94,782,741 records in train data and 12226710 in test data.

## **BUSINESS QUESTION:**

### **1. For each year and month:**

Due to the size of the screenshot result, this section only summarises the answers to the first business questions, detailed SQL outputs are shown in Appendix 4.

- The total number of trips for each year and month ranges from 171,206 trips in April 2020 to 6,939,892 trips in March 2019.
- Fridays in March 2019 have the most trips (1,243,533) in this period when grouping the data by weekdays. Thursday and Friday are the most frequent ‘most trips’ weekdays in this period.
- The busiest time slot for a taxi driver is 18:00, followed by 15:00.
- The average number of passengers per trip has dropped from 2 ppl/trip in 2019 to 1 ppl/trip in 2020 and 2021.
- The average amount per trip ranges from \$11.94 in January 2019 to \$15.57 in October 2019. Generally speaking, the average amount per trip in 2019 is higher than those same months in 2020 and 2021.
- The average amount per passenger ranges from \$9.93 in January 2019 to \$12.94 in October 2019. In addition, the average amount per passenger does not change much in this period.

The result suggests that the COVID-19 pandemic might have a significant impact on the taxi industry in New York by affecting the total number of trips and average passengers per trip. Looking into the considerable drop in full trips per month from February 2020 (5,483,463) to March 2020 (2,579,394) and the continuing reduction in trip numbers until the lowest point in April 2020 (171,206). It is reasonable to argue that the COVID pandemic likely causes the impact.

## 2. For each taxi colour (yellow and green):

### a. What was the average, median, minimum, and maximum trip duration in seconds?

The minimum duration of the yellow taxi is around one minute and for the green taxi is only 12 seconds. The max duration for the yellow taxi is around 62 minutes, and 76 minutes for the green taxi. The average duration for yellow taxis is less than for green taxis, but both are about 12 minutes. And the median duration for both colours is around 10 minutes; The medium duration is smaller than the average duration, which means the duration distribution is still right skew after cleaning.

taxi_color	avg_duration	med_duration	min_duration	max_duration
yellow	711.61	590	56	3749
green	748.85	593	12	4560

### b. What was the average, median, minimum, and maximum trip distance in km?

The result for yellow and green are similar. The minimum trip distance of the yellow and green are around 50 meters and 20 meters. The maximum trip distance for the yellow and green taxis is about 30 km and 33 km, respectively. The medium distance is smaller than the average distance, which means the distance distribution is still right skew after cleaning.

taxi_color	avg_distance	med_distance	min_distance	max_distance
yellow	2.97	2.35	0.05	30.9
green	3.4	2.58	0.02	33.09

### c. What was the average, median, minimum and maximum speed in km per hour?

The result for yellow and green are similar. The maximum speed is around 56 km per hour. The average and median for both colours are minor differences in speed.

taxi_color	avg_speed	med_speed	min_speed	max_speed
yellow	15.89	15.24	0.23	56.93
green	16.81	16.41	0.01	55.33

## 3. What was the percentage of trips where the driver received tips?

According to the data dictionary, cash tips are not included in this dataset, and Tip\_amount is automatically populated for credit card tips. Therefore, trips with Tip\_amount > 0 should only be valid for Payment\_type equals credit card records. Assuming this question asks the percentage of trips where the driver received tips when the customer paid by card, the driver received tips from 95.93% of trips paid by card.

#### 4. For trips where the driver received tips, the percentage where the driver received tips of at least \$10.

Based on the same assumption as the above question, only around 0.21% of trips paid by card will the driver receive tips of at least \$10.

#### 5. Classify each trip into bins of durations:

Under 5 Mins, from 5 mins to 10 mins, from 10 mins to 20 mins, from 20 mins to 30 mins, and at least 30 mins.

##### a. Average speed (km per hour)

The average speed in these speed bins ranges from 14.26 km/hr to 23.2 km/hr. The result suggests that the average\_speed is lower when the duration is longer, but the average speed is higher when the duration is higher than 30 than the duration between 20 to 30 minutes.

duration_bins	average_speed
<5	23.2
5-10	17.31
10-20	15.29
>30	14.96
20-30	14.26

##### b. Average distance per dollar (km per \$)

The average distance per dollar ranges from 0.13 km/\$ to 0.28 km/\$ across these duration bins. The result shows that the higher the duration, the higher the average distance per dollar. Consider the result from the last question, where trips with 20-30 minutes duration have slower speed are likely to be affected by traffic congestion, or trips with a duration > 30 minutes are more likely to be passing toll, so the speed is higher than trips in the 20-30 bin.

duration_bins	average_distance_dollar
<5	0.13
5-10	0.17
10-20	0.21
20-30	0.24
>30	0.28

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

I would advise a taxi driver to target trips in the duration bin <5 minutes because these trips have the lowest average distance per dollar and the highest average speed among all these duration bins. This means the driver can travel less for each dollar income, plus the higher average speed will save some fuel costs. Furthermore, shorter duration means the driver can take more trips in the same amount of time, which is another advantage of targeting these trips.

# MACHINE LEARNING

## Feature engineering

The exploration analysis of all features and the correlation between the numeric variables is performed to understand how these features affect total\_amount. It suggested that “trip\_distance”, “duration”, “tip\_amount”, “tolls\_amount” has a significant correlation with “total\_amount”. “Fare\_amount” will not be used in the model as it’s the main source of the total amount of trip, and it is the highest correlated variable with total\_amount.

Considering the limitation of the community version of Databricks and time constrain, no heavy feature engineering is performed. Created features only includes: year, month, weekday, and hour created from “pickup\_datetime” to show the impact on total\_amount by date time variables.

## Models selected

A few models were compared to select the best fit. As it is a regression question, four types of models included in SparkML are considered to train the model, including Random Forest Regression Model, Linear Regression Model, Generalized Linear Regression Model, and Gradient Boosted Tree Regression Model. Considering the relatively limited native feature selection functions in Pyspark's MLlib.

The reason why tree-based models are considered is that the feature importance score estimated from a tree-based model can be used to select the variables that are with the highest importance. In addition, with the lack of linear relationship between the independent and most of the dependent variables, tree-based models are considered.

## Evaluation metric

The goodness-of-fit of the models was assessed by the Root of Mean Squared Error (RMSE) of label. RMSE closer to 0 indicates better model fitting. When the RMSE is less than the standard deviation of the label, the predictive model is considered useful. The standard deviation of the total amount is 6.34 in this dataset.

summary		total_amount
count		106262678
mean		14.811979136310464
stddev		6.33978166446633
min		3.31
max		73.25

Figure 2. Description table of total\_amount

R2 could be our backup or alternate measure of RMSE in case the RMSE is very large and hard to judge the model quality, as R2 value is in the range 0 to 1 and makes the model selection easier.

## Data preparation for models

The first step is to normalizing the features. Continuous variables are normalized and categorical variables are one hot encoded to make the data more computer-friendly. Categorical variables were transformed by one hot encoder, numerical variables were transformed by standard scaler, features and label were transformed as two vectors and these two vectors were the final inputs of the model. In addition, a pipeline was used to process the data.

2019 data was split into a train set, 2020 was split into validation set and 2021 data was split into a test set, each data set is 69%, 19% and 11% of the total data set respectively.

## Experiments

During the model training process, a sample dataset is created to test the model and help to decide on features used in the final models. Using the reduced-size sample data frame helped to save a lot of time.

Two selections of variables were evaluated, first selection was processed in order to select the main features and test the performance of each model.

### Experiment 1

The first experiment examines four abovementioned models on the sample dataset as an initial attempt to screen some irrelevant features. The result in figure 3 suggests that linear models are probably overfitting because the model performs much better on the train set than on the validation set. The feature importance in figure 4 from the random forest model suggests that the numeric features are more important than categorical features. It is clear that only three variables are with feature importance score over 0.1, which means other features are not with much significance to the model and probably introduced many noises into the model. In order to reduce the noise to models and reduce the complexity of data, only these three features are selected for the second experiment.

```
RandomForestRegressionModel
train RMSE = 1.6021955981952851 R^2 = 0.9414377932964247
val RMSE = 1.3542058566139357 R^2 = 0.9410885213930706
LinearRegressionModel
train RMSE = 0.5212720432872062 R^2 = 0.9938010914150687
val RMSE = 1.1148459032731455 R^2 = 0.9600735927426767
GeneralizedLinearRegressionModel
train RMSE = 0.5592276603543218 R^2 = 0.9928654981794262
val RMSE = 1.101576416901184 R^2 = 0.9610183868491686
GBTRegressionModel
train RMSE = 1.0962316319368055 R^2 = 0.9725848249166876
val RMSE = 1.4569923236504203 R^2 = 0.9318061711855228
```

Figure 3. Performance of model in Experiment 1

	idx	name	score
1	381	duration_scaled_0	0.457345
0	380	trip_distance_scaled_0	0.269699
4	384	tip_amount_scaled_0	0.142983
5	385	tolls_amount_scaled_0	0.065496
56	50	PULocationID_one_hot_138	0.019068
338	332	payment_type_one_hot_1	0.012515
2	382	speed_scaled_0	0.011650
339	333	payment_type_one_hot_2	0.005510
346	340	month_one_hot_01	0.002429
79	73	PULocationID_one_hot_132	0.001920
296	290	DOLocationID_one_hot_257	0.001100
3	383	extra_scaled_0	0.001013
343	337	taxi_color_one_hot_yellow	0.000938
337	331	trip_type_one_hot_1	0.000915
236	230	DOLocationID_one_hot_138	0.000624
199	193	DOLocationID_one_hot_13	0.000545
336	330	trip_type_one_hot_no	0.000519
342	336	ehail_fee_one_hot_0.0	0.000483
357	351	weekday_one_hot_6	0.000424
265	259	DOLocationID_one_hot_26	0.000393

Figure 4. Feature importance of the top 20 features

## Experiment 2

Figure 5 and Figure 6 suggests that all four models are performing better in the second experiment, the models are having lower RMSE scores and the overall feature importance is higher compared to the first experiment. All models have better performance on validation dataset after dropping features with low feature importance score from experiment 1. Moreover, and Generalized Linear Regression Model have the best performance followed by Linear Regression Model base on RMSE score in both experiments. The potential reason why these simple models are out performing more complexed tree-based models might due to the linear relationship between the features and the label. The other possible reason would be I did not tune the tree-based models by modifying hyper-parameters.



```

RandomForestRegressionModel
train RMSE = 1.7607397194897552 R^2 = 0.9292743923652219
val   RMSE = 1.267041360643953 R^2 = 0.9484282187614032
LinearRegressionModel
train RMSE = 1.1931219869081984 R^2 = 0.9675244838216988
val   RMSE = 0.9357000363997029 R^2 = 0.9718742675917899
GeneralizedLinearRegressionModel
train RMSE = 1.2045214429501225 R^2 = 0.9669009570722897
val   RMSE = 0.9194427664817635 R^2 = 0.9728431152858297
GBTRegressionModel
train RMSE = 1.5916135267827922 R^2 = 0.9422088139469557
val   RMSE = 1.2234025290992205 R^2 = 0.9519194645335192

```

Figure 5. Performance of model in Experiment 2

	idx	name	score
0	0	duration_scaled_0	0.362053
2	2	tip_amount_scaled_0	0.338771
1	1	trip_distance_scaled_0	0.299175

Figure 6. Feature importance in Experiment 2

## Final model

Models selected to be trained on the whole dataset are Linear Regression Model and Generalized Linear Regression Model based on their performance from the two experiments. And Generalized Linear Regression Model has better performance with a lower RMSE score as shown in Figure 7. Moreover, the RMSE for predicted test data is about 0.8254 (shown in Figure 8) by Generalized Linear Regression Model, which is even better than the score on the validation set. This might be due to the size of the test dataset being very small compared to train and validation sets.

```

LinearRegressionModel
train RMSE = 1.1978467007422922 R^2 = 0.9672954726819738
val   RMSE = 0.9387373507860265 R^2 = 0.9729049306855952
GeneralizedLinearRegressionModel
train RMSE = 1.2088108103795254 R^2 = 0.9666940316480883
val   RMSE = 0.9279069153676184 R^2 = 0.9735265285988735

```

Figure 7. Performance of model

```

test   RMSE = 0.8254403825781403 R^2 = 0.9775963663007785

```

Figure 8. Performance of the model on test (glr)

## PROBLEM FACED AND REFLECTION

There are many problems encountered in this project, and most of them are related to Databricks. The first problem is cluster stops due to many reasons. This seems to be the case with the community edition. Users need to create a new cluster every time and run it. Even in the same session, if your cluster stops due to inactivity or other reasons, you will need to create a new cluster, which might waste some time re-running codes.

The other problem is the lost connection to the cluster, or the notebook may have been detached. One common cause for this error is that the driver is undergoing a memory bottleneck. The driver crashes with an out-of-memory (OOM) condition and gets restarted or unresponsive due to frequent full garbage collection. It is a good practice not to store too many files and models on databricks to avoid this situation.

Furthermore, the performance or computing limitation of the Databricks community cluster is also a problem. For example, it happened to me many times when doing data cleaning for yellow taxi data by a for loop, which works well on the sample dataset. This may be due to the larger data as `df_yellow` has more than 10 million records. It would be better to choose another method, for example, SQL codes, to do the same task because python for loop might consume too much memory for the community version.

Exporting files from Databricks is also complicated for new users. To try more models and methods and to make wiser use of limited time when doing this project, I created additional Databricks community accounts to run queries. And one critical step is to export data files from one account to another. I have done a lot of research and trial and error to successfully transferred the parquet file to CSV in the original account, then saved it in FileStore and downloaded it to another account by making a URL link.

Another problem all community version users might face occasionally is that Databricks might be down for the whole day, and users cannot access any service from them. There is no way for us users to solve this problem and all we can do is wait. A lesson to me is that always organise some time buffer for projects if these problems occur.

In conclusion, this is an interesting and meaningful project for me. The size of the raw dataset to be analysed is the largest one I have ever met; perhaps this is why it is so challenging. I did not expect learning to use a new platform (Databricks) to be this challenging. And I did not expect a simple model could predict a better result. Sometimes we may need to avoid the fixed mindset of data scientists to chase for sophisticated models because sometimes less is more.

## REFERENCE:

Radečić, D. (2021). *CSV Files for Storage? No Thanks. There's a Better Option*. Medium. Retrieved 26 April 2022, from <https://towardsdatascience.com/csv-files-for-storage-no-thanks-theres-a-better-option-72c78a414d1d>.

TLC Trip Record Data - TLC. (n.d.). Nyc.Goc. <https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page>

## APPENDIX 1:

### DATA LOADING

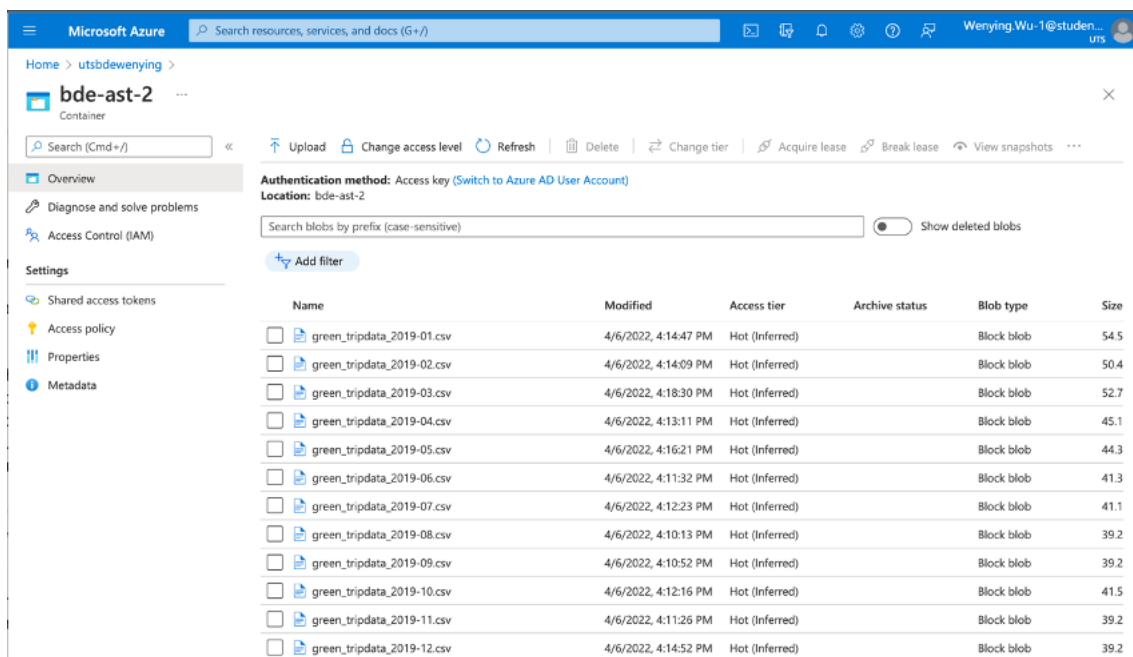
Loading data to Databricks includes five steps, and each step will be briefly explained in this section, and the details are shown in the notebook.

#### 1. Download Data

The first step is to download the dataset (62 CSV files) for yellow and green taxi cabs from Jan 2019 to July 2021. One method is to download manually from the NYC page, the other way is to use the query in Part 1.1 in the notebook (Refer bde-ast-2-14007025.ipynb for detail).

#### 2. Upload data into Azure

The second step is to upload the 62 CSV files to the storage account on the Microsoft Azure container. Figure 1 shows that the CSV files are saved in the Azure container “bde-ast-2”.



Name	Modified	Access tier	Archive status	Blob type	Size
<input type="checkbox"/> green_tripdata_2019-01.csv	4/6/2022, 4:14:47 PM	Hot (Inferred)		Block blob	54.5
<input type="checkbox"/> green_tripdata_2019-02.csv	4/6/2022, 4:14:09 PM	Hot (Inferred)		Block blob	50.4
<input type="checkbox"/> green_tripdata_2019-03.csv	4/6/2022, 4:18:30 PM	Hot (Inferred)		Block blob	52.7
<input type="checkbox"/> green_tripdata_2019-04.csv	4/6/2022, 4:13:11 PM	Hot (Inferred)		Block blob	45.1
<input type="checkbox"/> green_tripdata_2019-05.csv	4/6/2022, 4:16:21 PM	Hot (Inferred)		Block blob	44.3
<input type="checkbox"/> green_tripdata_2019-06.csv	4/6/2022, 4:11:32 PM	Hot (Inferred)		Block blob	41.3
<input type="checkbox"/> green_tripdata_2019-07.csv	4/6/2022, 4:12:23 PM	Hot (Inferred)		Block blob	41.1
<input type="checkbox"/> green_tripdata_2019-08.csv	4/6/2022, 4:10:13 PM	Hot (Inferred)		Block blob	39.2
<input type="checkbox"/> green_tripdata_2019-09.csv	4/6/2022, 4:10:52 PM	Hot (Inferred)		Block blob	39.2
<input type="checkbox"/> green_tripdata_2019-10.csv	4/6/2022, 4:12:16 PM	Hot (Inferred)		Block blob	41.5
<input type="checkbox"/> green_tripdata_2019-11.csv	4/6/2022, 4:11:26 PM	Hot (Inferred)		Block blob	39.2
<input type="checkbox"/> green_tripdata_2019-12.csv	4/6/2022, 4:14:52 PM	Hot (Inferred)		Block blob	39.2

Figure 1: The new Azure container named bde-ast-2 after uploading the CSV files

#### 3. Create cluster and notebook in databricks

To use Databricks, we need to create a new cluster and a notebook from the databricks home page. Preferred programming language (Python, Scala, SQL, or R) can be chosen when creating a notebook, and I select Python in this project. Once the Notebook is set up, the coding part can be started by referencing the library that would be used, instantiating a local Spark Session, and

loading data.

#### **4. Mount the Azure Blob Storage into the Databricks File System.**

Because the dataset is in Azure Blob Storage, the data source should come from Blob Storage, so the first thing to do is to set the connection settings for Blob Storage. The next step is to set the Access Key, which can be found from the Blob Storage page under the settings section in Azure Portal. The last step is to get the location of the data and the data content. Spark APIs and Databricks APIs can be used to get the storage location and set the data source for spark.

#### **5. Define schemas and convert CSV files to Parquet**

It is essential to define a good schema to improve the import performance. A well-defined schema can avoid reading the same data multiple times and provide validation on import.

Another thing to be noted is to convert CSV files to parquet. While both CSV and parquet are a certain format of storing data, the key difference between CSV and parquet is that CSV utilizes row storage and parquet utilizes column storage. And the column storage files parquet is more lightweight because only one data type is presented in one specific column, and hence more efficient compression can be performed column-wise. And a further benefit of column storage files is that both queries and aggregations are faster than those on row storage files. In a nutshell, parquet is simply a more efficient file format for large files that consumes less disk space and allows queries operate faster compared to CSV.

I have tested a few queries on the same dataset saved as parquet files and CSV files, and noticed that there is a significant reduction in execution time by storing files as parquet.

## APPENDIX 2

### Data Dictionary:

#### 1. Administrative data:

Feature Name	Description	Possible Value
VendorID	A code indicating the TPEP provider that provided the record. 1= Creative Mobile Technologies, LLC; 2= VeriFone Inc.	1 and 2
RatecodeID	The final rate code in effect at the end of the trip. 1= Standard rate, 2=JFK, 3=Newark, 4=Nassau or Westchester, 5=Negotiated fare, 6=Group ride	integer between 1 to 6
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. Y= store and forward trip N= not a store and forward trip	Y and N

#### 2. Trip data:

pickup_datetime	The date and time when the meter was engaged.	the same year_month with files_date
dropoff_datetime	The date and time when the meter was disengaged.	the same year_month with files_date
PULocationID	TLC Taxi Zone in which the taximeter was engaged	If yellow taxi: in the range of 1-263 If green taxi, in range of Boro zone
DOLocationID	TLC Taxi Zone in which the taximeter was disengaged	in the range of 1-263
passenger_count	The number of passengers in the vehicle. This is a driver-entered value.	> 1 and <= 6, by law restrict
trip_type	(only include in green taxi data)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. 1= Street-hail 2= Dispatch	1 and 2

trip_distance	The elapsed trip distance in miles reported by the taximeter.	> 0
payment_type	A numeric code signifying how the passenger paid for the trip. 1= Credit card, 2= Cash, 3= No charge, 4= Dispute, 5= Unknown, 6= Voided trip	int between 1 to 6

### 3. Payment data:

fare_amount	The time-and-distance fare calculated by the meter.	>= 2.5
Extra	Miscellaneous extras and surcharges. Currently, this only includes the \$0.50 and \$1 rush hour and overnight charges.	> 0
mta_tax	\$0.50 MTA tax that is automatically triggered based on the metered rate in use.	> 0.5
improvement_surcharge	\$0.30 improvement surcharge assessed trips at the flag drop. The improvement surcharge began being levied in 2015.	0.3
tip_amount	Tip amount – This field is automatically populated for credit card tips. Cash tips are not included.	> 0; tip_amount has value only when payment_type = 2
tolls_amount	Total amount of all tolls paid in trip.	> 0
total_amount	The total amount charged to passengers. Does not include cash tips.	> 0
congestion_surcharge	no description in the original dictionary	If yellow taxi: = 2.5; If green taxi: = 2.75
ehail_fee	(only include in green taxi data) no description in the original dictionary	> 0

## APPENDIX 3

### Data cleaning

#### Step 1: Drop duplicate

The dataset contains records that all features are the same. I have dropped duplicates and only keep one record.

#### Step 2: Clean based on the dictionary

The dataset is cleaned based on the dictionary from TLC. Please check the Appendix 1 for possible values and check notebook 4.3 for the details process. Below are only examples of cleaning.

1. VendorID should be 1 or 2
2. Pickup year month and should be the same of the file's year month.
3. Identify the pickup and drop-off location of each trip. Location ID should be an integer in range 1 to 264
4. The number of passengers should not be higher than 6, by law restrict.
5. Payment features should not be negative.
6. Only Tips with payment by credit card are considered as valid value.
7. congestion\_surcharge for yellow taxi data should be 0 or 2.5, for green taxi data should be 0 or 2.75.

#### Step 3: Create features

Two essential features, "Duration" and "speed" were derived in the data cleaning section in order to select data. Only two features are created due to the limitation of the databricks cluster's performance.

1. Duration (in second): duration of each trip has been calculated by the difference between pickup time and drop-off time.
2. Speed (in mile/hour): The average speed was calculated as the ratio of trip distance and duration.

#### Step 4: Clean outlier based on assumptions

Additional cleaning was taken as there are still illogical values and extreme values.

Implausible data have been dropped based on common sense and the empirical distribution of this dataset.

1. Only pickup time smaller than drop-off time is considered a valid value and kept.
2. As fare amount has \$2.50 initial charge, and Plus 50 cents per 60 seconds in slow traffic or when the vehicle is stopped or per 1/5 mile when traveling above 12mph. Approximated fare amount was been calculated by  $2.5 + 0.5 * \text{"duration"}$  (in minutes). The fare amount inside of the 50% of the calculated fare amount and 150% of the calculated fare amount are considered a valid value and kept.
3. Numeric features are further cleaned. Including duration, speed, and payment features are further cleaned. Some of these features contain extremely high and lower values.



The value inside of 0.01 approximate quantile and 0.99 approximate quantiles are considered a valid value and kept. The reason why 0.01 and 0.99 quantile are chosen is because the empirical 1.5xIQR rule cuts off too much data in this case.

### Step 5: Fill missing value

Some features contain missing data, including “VendorID”, “store\_and\_fwd\_flag”, “RatecodeID”, “payment\_type”, “passenger\_count”, “trip\_type”, “congestion\_surcharge”, “ehail\_fee”, “improvement\_surcharge”. The main findings are:

1. There are five features: “VendorID”, “store\_and\_fwd\_flag”, “RatecodeID”, “payment\_type” and “passenger\_count” shown to be missing data on same rows.
2. There are 14.27 % of green taxi data (1,191,314 records) that all these features are null. But the percentage of null values reached 43.67% in the records in the year 2021.
3. There are 1.52 % of yellow taxi data (1,890,197 records) that all these features are null. And 5.56 % are null in the year 2021.

Below are other finding of the missing value:

1. “congestion\_surcharge” have a lot of null value
1. Null value of “improvement\_surcharge” only exist in green taxi data. Only two missing values were found.
2. Most of Ehail\_fee data are null

```
congestion_surcharge    4855981
VendorID                1890197
passenger_count         1890197
RatecodeID              1890197
store_and_fwd_flag      1890197
payment_type            1890197
dtype: int64
```

```
'VendorID', 'store_and_fwd_flag', 'RatecodeID', 'payment_type', 'passenger_count' all have 1191314 null value
percentage of null in total: 1.52 %
percentage of null in 2021: 5.56 %
```

Figure 2. Information of null value in yellow data

```
ehail_fee               8348213
congestion_surcharge    1737696
trip_type               1191678
VendorID                1191314
store_and_fwd_flag      1191314
RatecodeID              1191314
passenger_count         1191314
payment_type            1191314
improvement_surcharge    2
dtype: int64
```

```
'VendorID', 'store_and_fwd_flag', 'RatecodeID', 'payment_type', 'passenger_count' all have same null value
percentage of null in total: 14.27 %
percentage of null in 2021: 43.67 %
```

Figure 2. Information of null value in green data

The missing values were filled by “missing” rather than delete directly. There are three main reasons:

1. The features contain missing value are considered as categorical feature. It is not reasonable to fill the null value with a value like what we do with numerical features, for example, fill by mean or 0.
2. The percentage of null value in total data is not high, but the percentage of null values of green taxi data in the year 2021 are nearly 50%.
3. By further exploration, the average total amount is higher when the five features are missing. Which is around double of average total amount of those records with no missing value.

## APPENDIX 4

### BUSINESS QUESTION:

#### 1. For each year and month:

##### a. What was the total number of trips?

2019 appears to be the busiest year in terms of taxi trips, with almost all the months in 2019 having more trips than 2020 and 2021. This phenomenon is likely to be one of the effects of the COVID-19 pandemic which limits the population from travelling. The three highest total trip numbers come from March, May and January in 2019, with 6,939,892, 6,662,514 and 6,615,819 trips respectively. Another interesting finding is that March is ranked in the top 5 for each year.

##### b. Which day of week (e.g. monday, tuesday, etc..) had the most trips?

Overall, Thursday and Friday have the most trips. The three highest trip numbers by weekday are from Fridays in March 2019, Thursdays in May 2019 and Thursdays in January 2019 with 1,243,533, 1,185,156 and 1,183,105 trips respectively. Sunday appears to be the day people travel the least by taxi because Sunday does not appear in any month in any year as the day with the most trips.

##### c. Which hour of the day had the most trips?

From the SQL output, the hours with the most trips are 18:00 and 15:00 throughout this period.

year	month	trips
2019	3	6939892
2019	5	6662514
2019	1	6615889
2019	4	6541435
2019	10	6276640
2019	2	6253296
2019	6	6073976
2019	11	5995397
2019	12	5970952
2019	9	5702567
2019	7	5451857
2019	8	5190651
2020	1	5499640
2020	2	5483463
2020	3	2579394
2020	10	1381311
2020	11	1233811
2020	12	1196946
2020	9	1080562
2020	8	781949
2020	7	605052
2020	6	406707
2020	5	232673
2020	4	171206
2021	6	2267143
2021	7	2186331
2021	5	2005637
2021	4	1740399
2021	3	1548578
2021	2	1103338
2021	1	1083472

a.

year	month	week	trips
2019	3	Friday	1243533
2019	5	Thursday	1185156
2019	1	Thursday	1183105
2019	10	Thursday	1113678
2019	4	Tuesday	1111263
2019	11	Friday	1071777
2019	6	Saturday	998572
2019	2	Friday	994754
2019	7	Wednesday	988731
2019	8	Thursday	953830
2019	12	Tuesday	949736
2019	9	Thursday	875106
2020	1	Friday	980222
2020	2	Saturday	946637
2020	3	Tuesday	412072
2020	10	Thursday	259158
2020	12	Tuesday	226824
2020	11	Monday	210191
2020	9	Wednesday	206618
2020	8	Monday	131279
2020	7	Thursday	116184
2020	6	Tuesday	77249
2020	5	Friday	46412
2020	4	Thursday	31761
2021	6	Wednesday	409038
2021	7	Thursday	397020
2021	4	Friday	330908
2021	5	Saturday	326084
2021	3	Wednesday	273903
2021	2	Friday	195206
2021	1	Friday	188896

b.

year	month	hour	trips
2019	3	18	482525
2019	1	18	473082
2019	5	18	467390
2019	4	18	463174
2019	10	18	445519
2019	2	18	445427
2019	11	18	419653
2019	6	18	403916
2019	12	18	402349
2019	9	18	398942
2019	7	18	376120
2019	8	18	358535
2020	2	18	404368
2020	1	18	398393
2020	3	18	189107
2020	10	15	107561
2020	12	15	106066
2020	11	15	102982
2020	9	15	84802
2020	8	15	61539
2020	7	15	47929
2020	6	15	33171
2020	5	15	18842
2020	4	15	13306
2021	6	18	171102
2021	7	18	170725
2021	5	18	154546
2021	4	15	137632
2021	3	15	124896
2021	1	15	94318
2021	2	15	91391

c.

**d. What was the average number of passengers?**

The result is round in the nearest integer which is more reason for this question. In 2019, every month have approximately two passengers on average per trip. However, beginning from 2020 March, there is only about one passenger per trip. This might again be due to covid restrictions affecting the number of people traveling.

**e. What was the average amount paid per trip (using total\_amount)?**

October, September, and May in 2019 held the highest average amount paid per trip. In addition, the average total\_amount per trip in 2019 was higher than in 2020 and 2021.

**f. What was the average amount paid per passenger (using total\_amount)?**

October 2019 and June 2020 have the highest average amount per passenger. Interestingly, there is no significant difference between the average amount paid per customer from 2019 to 2020.

year	month	avg_passengers	year	month	avg_amount	year	month	avg_amount_pass
2019	5	2.0	2019	10	15.57	2019	10	12.94
2019	4	2.0	2019	9	15.53	2019	9	12.89
2019	8	2.0	2019	5	15.45	2019	11	12.78
2019	9	2.0	2019	6	15.44	2019	5	12.76
2019	12	2.0	2019	12	15.41	2019	6	12.75
2019	6	2.0	2019	11	15.41	2019	12	12.72
2019	7	2.0	2019	4	15.11	2019	4	12.49
2019	11	2.0	2019	7	15.06	2019	7	12.44
2019	3	2.0	2019	3	14.98	2019	3	12.38
2019	1	2.0	2019	8	14.87	2019	8	12.26
2019	2	2.0	2019	2	14.68	2019	2	12.16
2019	10	2.0	2019	1	11.94	2019	1	9.93
2020	1	2.0	2020	2	14.8	2020	10	12.44
2020	2	2.0	2020	3	14.61	2020	3	12.43
2020	11	1.0	2020	1	14.49	2020	2	12.42
2020	3	1.0	2020	10	14.48	2020	11	12.33
2020	4	1.0	2020	9	14.32	2020	9	12.31
2020	9	1.0	2020	11	14.3	2020	12	12.28
2020	8	1.0	2020	12	14.24	2020	1	12.14
2020	7	1.0	2020	8	13.92	2020	8	12.02
2020	6	1.0	2020	7	13.75	2020	7	11.98
2020	12	1.0	2020	6	13.52	2020	6	11.89
2020	10	1.0	2020	5	12.56	2020	5	11.2
2020	5	1.0	2020	4	11.93	2020	4	10.78
2021	6	1.0	2021	6	15.22	2021	6	12.92
2021	5	1.0	2021	7	14.95	2021	5	12.67
2021	7	1.0	2021	5	14.83	2021	4	12.61
2021	3	1.0	2021	4	14.64	2021	7	12.58
2021	2	1.0	2021	3	14.34	2021	3	12.39
2021	1	1.0	2021	2	14.3	2021	2	12.37
2021	4	1.0	2021	1	13.74	2021	1	11.9

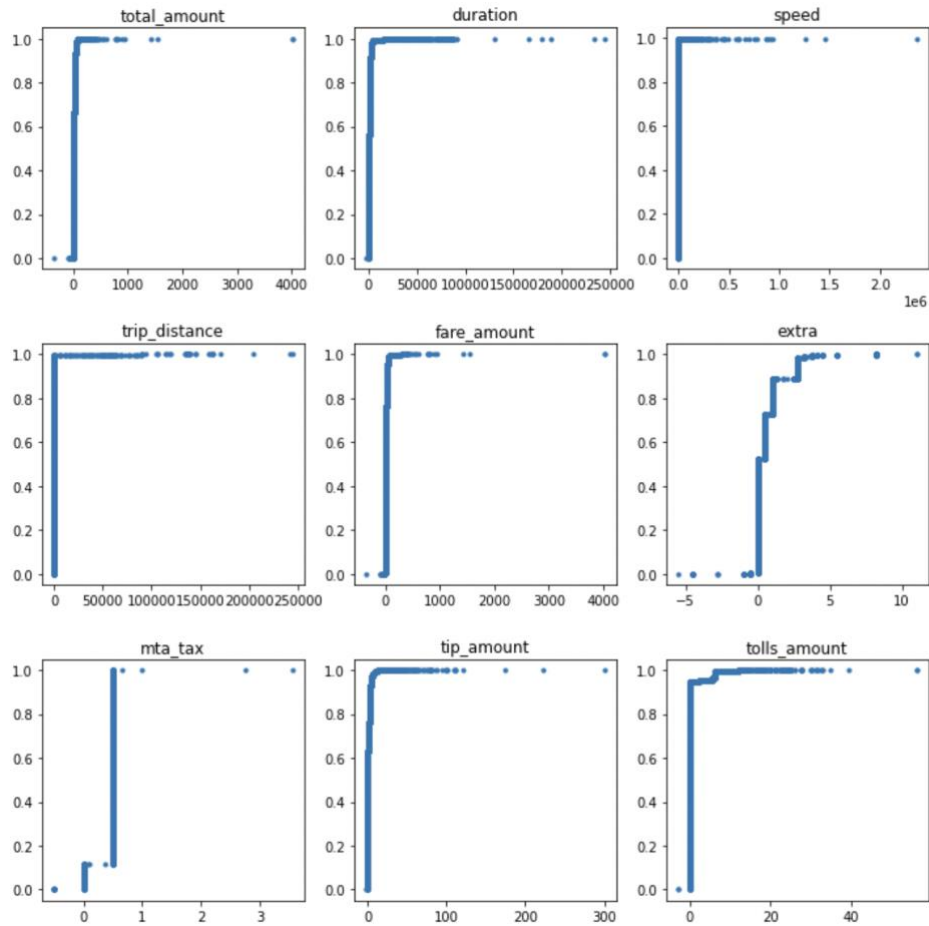
d.

e.

f.

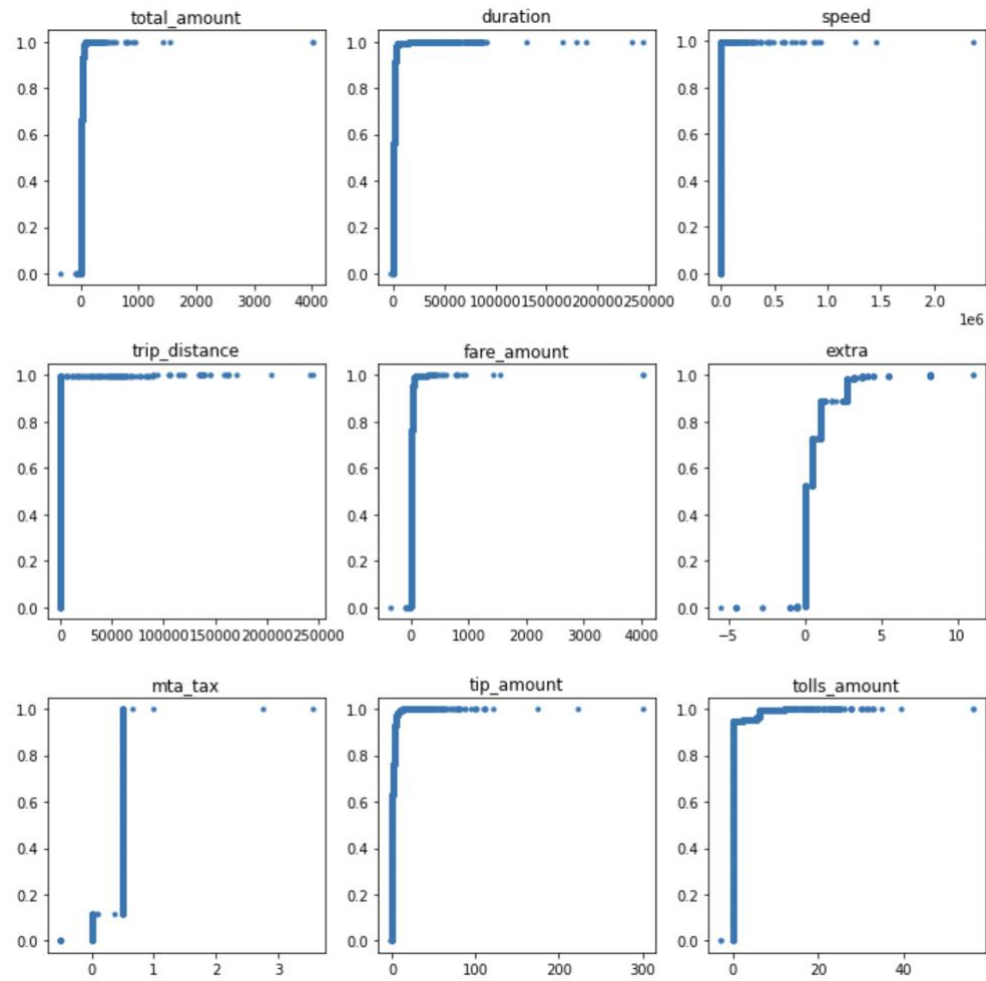
## APPENDIX 5

a. Distribution graph and table of numeric features for yellow taxi data



	Q1	Q3	lower	upper	0.01	0.99	0.005	0.999
<b>total_amount</b>	11.16	20.38	-2.67	34.21	5.30	74.70	3.8	80.52
<b>duration</b>	400.00	1100.00	-650.00	2150.00	56.00	3749.00	12.0	4616.00
<b>speed</b>	7.26	13.46	-2.04	22.76	0.14	35.36	0.0	39.70
<b>trip_distance</b>	0.98	3.01	-2.06	6.05	0.02	19.20	0.0	20.83
<b>fare_amount</b>	6.50	14.50	-5.50	26.50	3.00	52.00	2.5	64.50
<b>extra</b>	0.00	2.50	-3.75	6.25	0.00	3.50	0.0	3.75
<b>mta_tax</b>	0.50	0.50	0.50	0.50	0.50	0.50	0.0	0.50
<b>tip_amount</b>	0.00	2.95	-4.43	7.38	0.00	12.28	0.0	15.00
<b>tolls_amount</b>	0.00	0.00	0.00	0.00	0.00	6.12	0.0	6.12

b. Distribution graph and table of numeric features for green taxi data



	Q1	Q3	lower	upper	0.01	0.99	0.005	0.999
<b>total_amount</b>	8.80	23.16	-12.74	44.70	3.3	69.92	0.0	78.50
<b>duration</b>	420.00	1249.00	-823.50	2492.50	12.0	4560.00	4.0	7224.42
<b>speed</b>	8.47	14.49	-0.55	23.51	0.0	34.41	0.0	39.15
<b>trip_distance</b>	1.09	4.21	-3.59	8.89	0.0	20.56	0.0	23.63
<b>fare_amount</b>	7.00	19.50	-11.75	38.25	2.5	62.15	0.0	71.20
<b>extra</b>	0.00	1.00	-1.50	2.50	0.0	3.25	0.0	4.50
<b>mta_tax</b>	0.50	0.50	0.50	0.50	0.0	0.50	0.0	0.50
<b>tip_amount</b>	0.00	1.86	-2.79	4.65	0.0	8.00	0.0	10.00
<b>tolls_amount</b>	0.00	0.00	0.00	0.00	0.0	6.12	0.0	6.55

## APPENDIX 6

### Correlation for numeric features of all data:

total_amount	trip_distance	fare_amount	extra	tip_amount	tolls_amount	duration	speed
<b>1.00000</b>	0.88850	0.95254	0.07227	0.63116	0.34652	0.92423	- 0.02921
<b>0.88850</b>	1.00000	0.92908	- 0.00770	0.39281	0.31662	0.83320	0.24908
<b>0.95254</b>	0.92908	1.00000	0.00657	0.41670	0.28319	0.97341	- 0.03737
<b>0.07227</b>	-0.00770	0.00657	1.00000	0.05241	0.00135	0.00929	- 0.04204
<b>0.63116</b>	0.39281	0.41670	0.05241	1.00000	0.17585	0.39864	0.00576
<b>0.34652</b>	0.31662	0.28319	0.00135	0.17585	1.00000	0.24326	0.05239
<b>0.92423</b>	0.83320	0.97341	0.00929	0.39864	0.24326	1.00000	- 0.21022
<b>-0.02921</b>	0.24908	-0.03737	- 0.04204	0.00576	0.05239	-0.21022	1.00000