Cairo University

Faculty of Engineering

Computer Engineering Department

NYC Taxi Trip Duration

May 31, 2022

Team 5

Ahmed Ashraf	1	2
Ahmed Sherif	1	3
Ahmed Magdy	1	5
Abdulrahman Ahmed Fadl	1	32

Contribution

Ahmed Ashraf	Feature Extraction and Training
Ahmed Sherif	Data Cleaning and Visualization
Ahmed Magdy	Feature Extraction and Training
Abdulrahman Ahmed Fadl	Data Cleaning and Visualization

Problem Definition

The goal is to build a model that predicts the total ride duration of taxi trips in New York City. The primary dataset is one released by the NYC Taxi and Limousine Commission, which includes pickup time, geo-coordinates, number of passengers, and several other variables. You can find the dataset here.

Motivation

To improve the efficiency of electronic taxi dispatching systems it is important to be able to predict how long a driver will have his taxi occupied. If a dispatcher knew approximately when a taxi driver would be ending their current ride, they would be better able to identify which driver to assign to each pickup request.

Evaluation

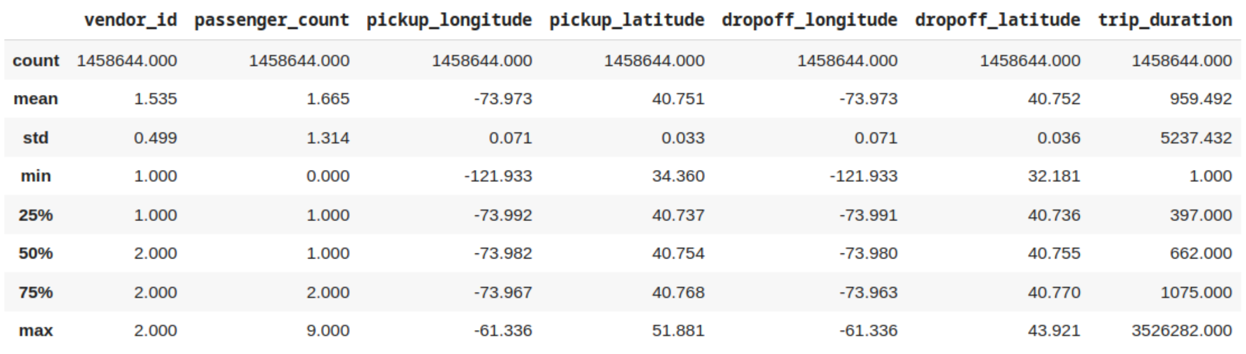
The evaluation metric for this problem is Root Mean Squared Logarithmic Error. The RMSLE is calculated as:

$$\epsilon = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (log(p_i + 1) - log(a_i + 1))^2}$$

Where:

- ϵ is the RMSLE value (score)
- n is the total number of observations in the (public/private) data set,
- p_i is your prediction of trip duration, and
- a_i is the actual trip duration for i.
- $log(x)$ is the natural logarithm of x

Analysis



	vendor_id	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	trip_duration
count	1458644.000	1458644.000	1458644.000	1458644.000	1458644.000	1458644.000	1458644.000
mean	1.535	1.665	-73.973	40.751	-73.973	40.752	959.492
std	0.499	1.314	0.071	0.033	0.071	0.036	5237.432
min	1.000	0.000	-121.933	34.360	-121.933	32.181	1.000
25%	1.000	1.000	-73.992	40.737	-73.991	40.736	397.000
50%	2.000	1.000	-73.982	40.754	-73.980	40.755	662.000
75%	2.000	2.000	-73.967	40.768	-73.963	40.770	1075.000
max	2.000	9.000	-61.336	51.881	-61.336	43.921	3526282.000

The data consists of

- **id** - a unique identifier for each trip
- **vendor_id** - a code indicating the provider associated with the trip record
- **pickup_datetime** - date and time when the meter was engaged
- **dropoff_datetime** - date and time when the meter was disengaged
- **passenger_count** - the number of passengers in the vehicle
- **pickup_longitude** - the longitude where the meter was engaged
- **pickup_latitude** - the latitude where the meter was engaged
- **dropoff_longitude** - the longitude where the meter was disengaged
- **dropoff_latitude** - the latitude where the meter was disengaged
- **store_and_fwd_flag** - This flag indicates whether the trip record was held in-vehicle memory before sending to the vendor because the vehicle did not have a connection to the server - Y=store and forward; N=not a store and forward trip
- **trip_duration** - duration of the trip in seconds

From the statistical summary, we can see that most of the features are normal except the **trip_duration**. It has a minimum of 1 second and a maximum of 3526282 seconds

(980 hours = 40.8 days). No way someone can take a trip that long using a taxi. Also, a 1 second will not get you anywhere. So, we need to remove the outliers for the **trip_duration**.

The **pickup_datetime** feature seems very interesting because we can extract the hour of the day, the day of the week, and the day of the month from it. For example, the trip time differs most likely to be affected if the trip is at the peak hour or if it's in a winter month.

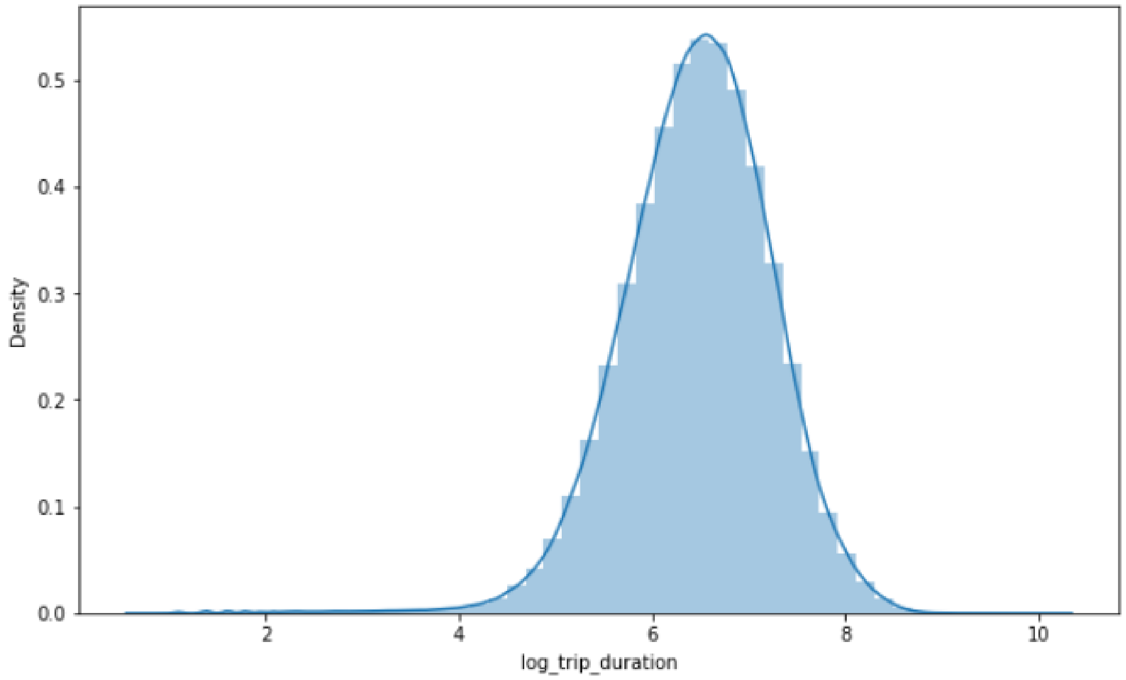
The **passenger_count** feature is an interesting feature because an increased number of passengers may lead to an increased number of stops which may extend the trip time between the starting and ending points.

The **vendor_id** is a code indicating the provider associated with the trip record. Different vendors may provide different shortest-path routes. Some vendors may provide corrupted information on the shortest path but I think this is highly unlikely to happen.

The **store_and_fwd_flag** flag indicates whether the trip record was held in-vehicle memory before sending it to the vendor because the vehicle did not have a connection to the server. Maybe there is a relationship between slow trips and server disconnects so we may consider this feature in the training.

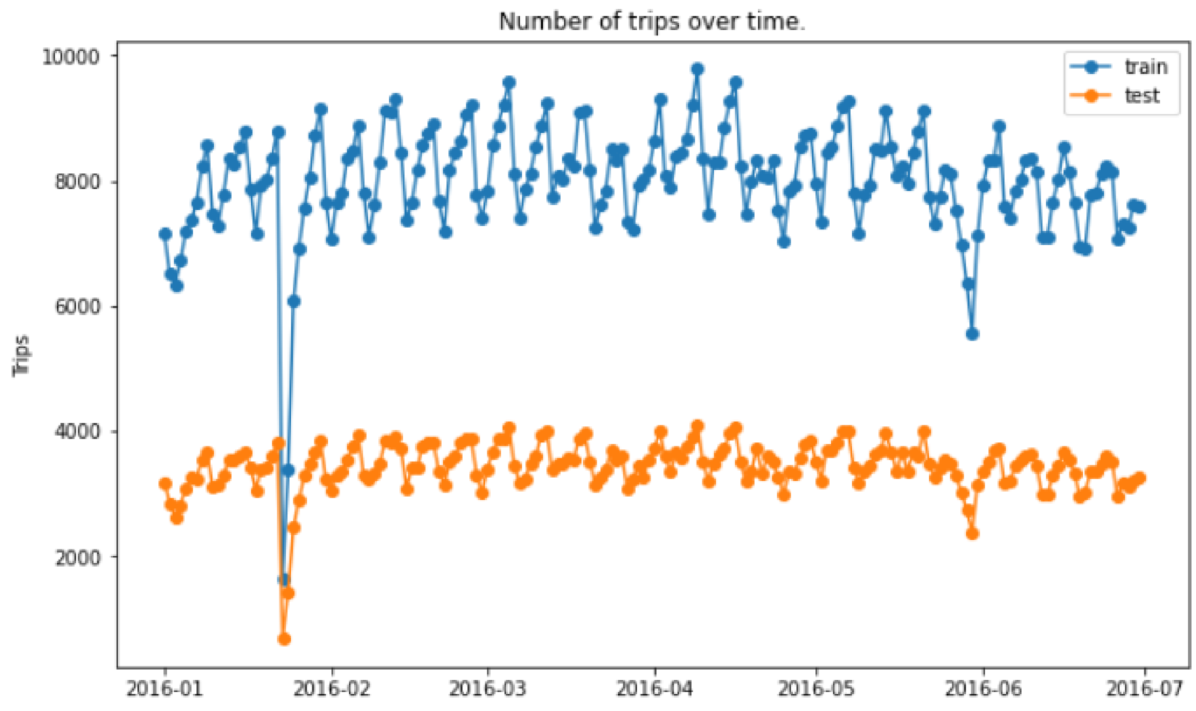
The **pickup_longitude**, **pickup_latitude**, **dropoff_longitude**, and **dropoff_latitude** are the most important features that give value to the model training. We can use them to calculate different types of distances to get heuristics about the real trip distance which effectively affects the trip duration.

Visualizing the histogram of the log transformation of the trip duration after removing the outliers using its mean and standard deviation is useful because the error metric is the mean squared log error.



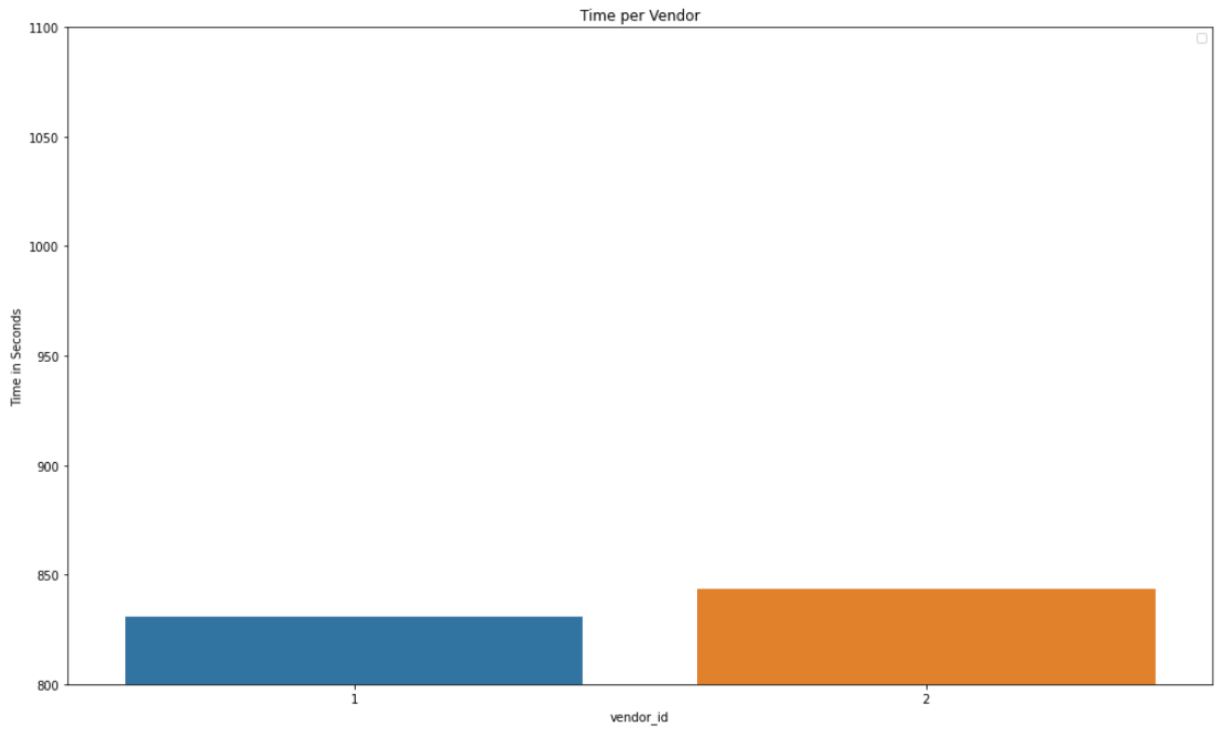
We can see, that the trip duration has normal distribution within a realistic limit.

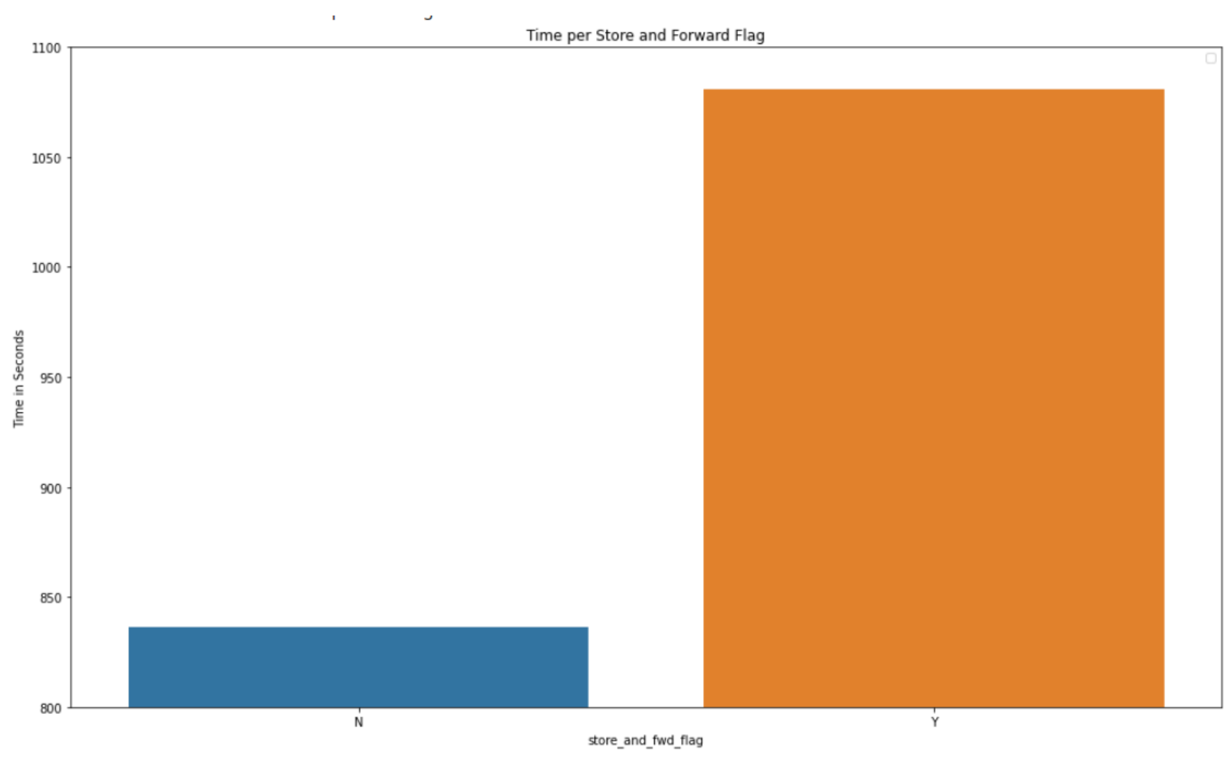
It's interesting to visualize the number of trips over time. This could reveal apparent seasonality in the data, and certain trends, point out any significant outliers and indicate missing values. Also, we can compare the test and train data to see if both of them have the same pattern to ensure uniformity in the test data sample.



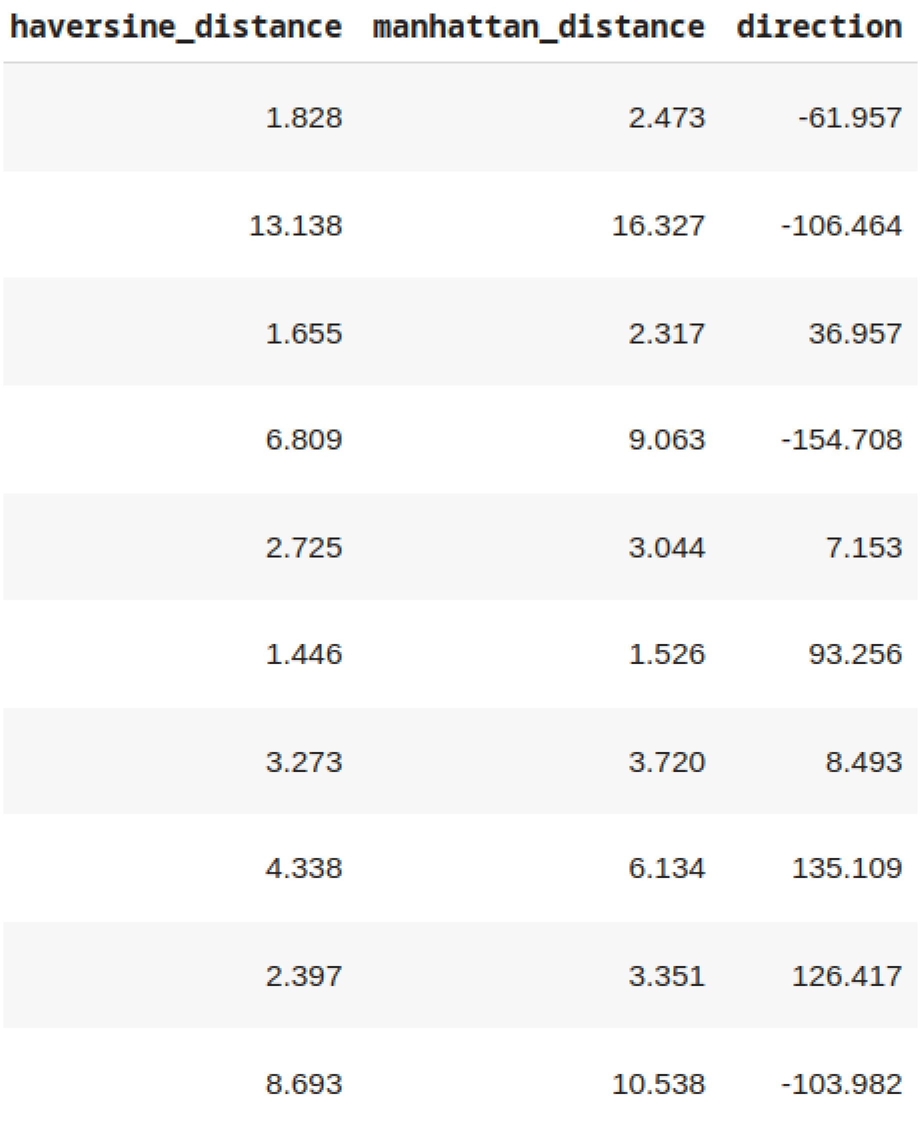
Both the train and test data almost have the same pattern. We can see a significant drop in the number of trips around the last of the Jan. and beginning of Feb. and another drop after 4 months.

It doesn't look like there is a significant difference between the two vendors. They are almost the same. But there is some difference between the stored trips and the disconnected ones.


Time per Vendor
Time in Seconds
1100
1050
1000
950
900
850
800
1
2
vendor_id

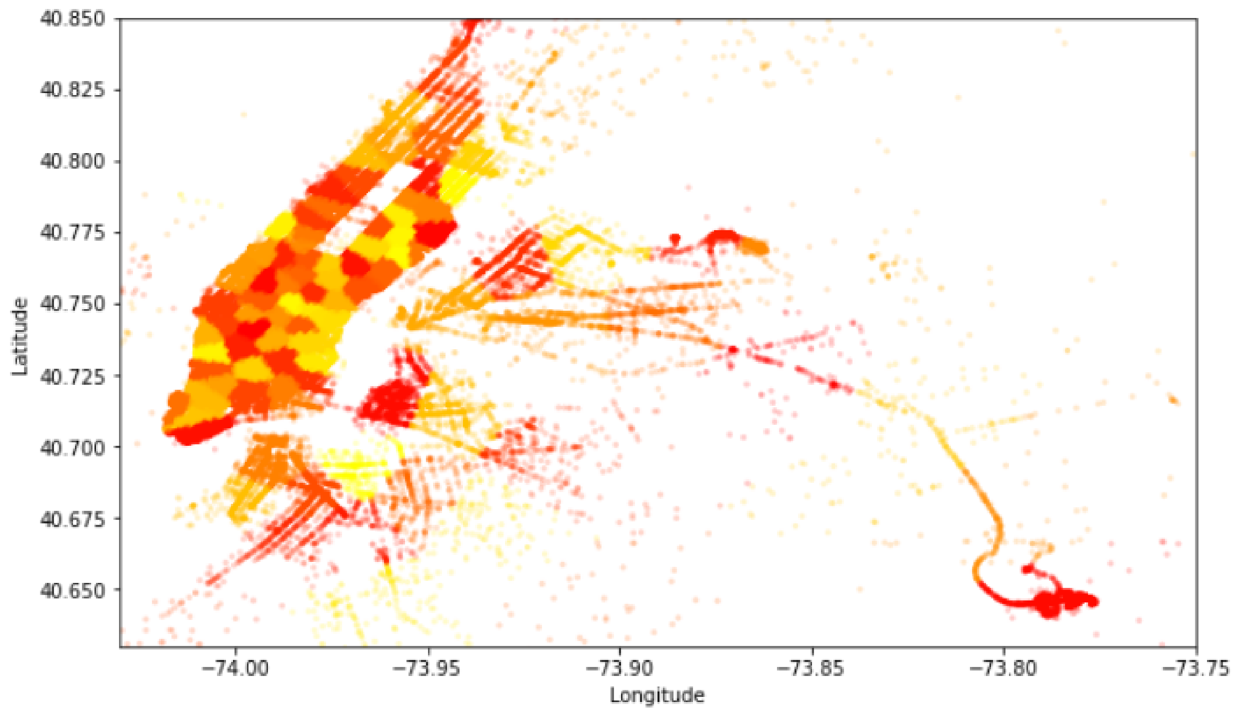


The distance is a very important feature for our experiments so we can calculate the **haversine distance** which is the great-circle distance between two points on a sphere given their longitudes and latitudes. Also, we can then calculate the summed distance traveled in **Manhattan**. Finally, we can calculate the **direction (or bearing)** of the distance traveled. These calculations are stored as variables in separate data sets.

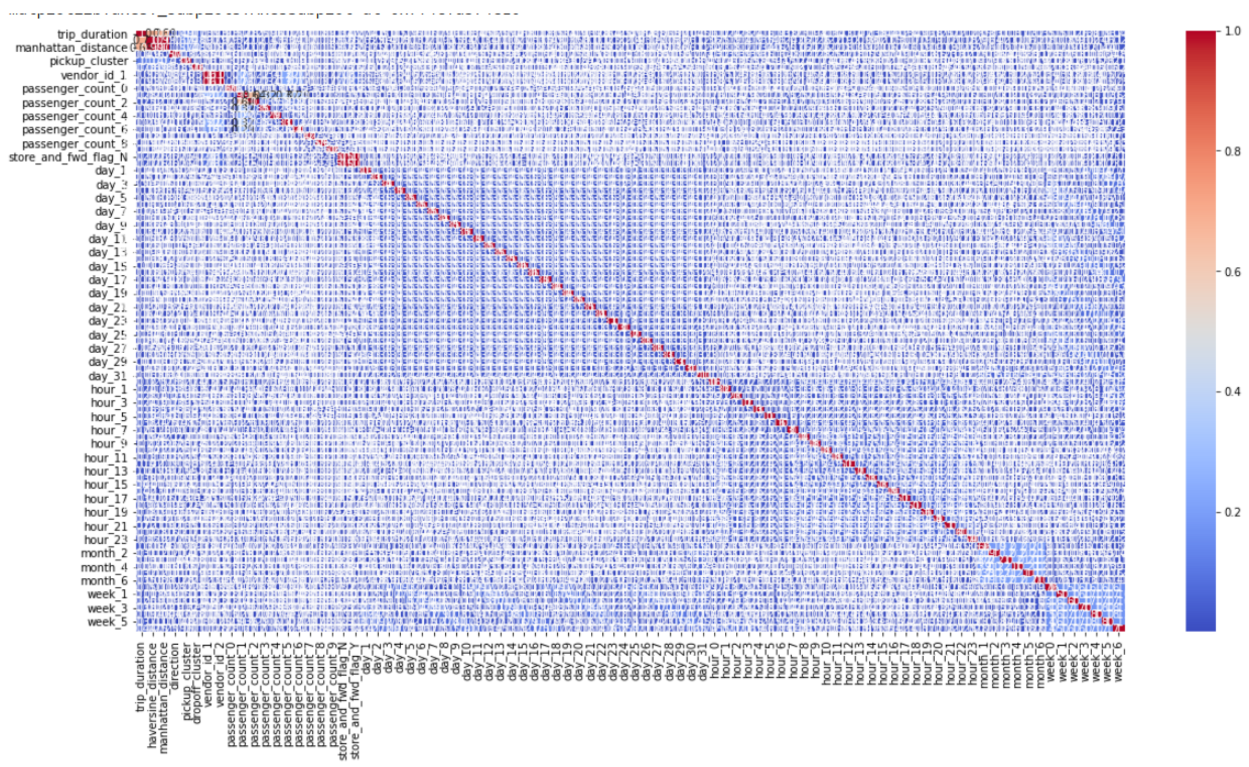


haversine_distance	manhattan_distance	direction
1.828	2.473	-61.957
13.138	16.327	-106.464
1.655	2.317	36.957
6.809	9.063	-154.708
2.725	3.044	7.153
1.446	1.526	93.256
3.273	3.720	8.493
4.338	6.134	135.109
2.397	3.351	126.417
8.693	10.538	-103.982

Also, we can create neighborhoods of distance. For example, splitting the pickup points into clusters or locations. We used KMeans to visualize the coordinates clusters.



After that, we converted the categorical data using the One Hot Encoder to numerical and normalized the numerical data so all of them are of the same scale and contribute to the model equally. The heatmap of the final features is:



Results

We trained Linear Regression, Random Forest, and XGBoost models and compared them to choose the best one.

- Linear Regression is the simplest model we used to tackle the problem. Its training time is very small and doesn't have hyperparameters to tune but its evaluation is bad compared to other submissions in the competition. The most useful usage for Linear Regression is that we used it as a baseline to compare other models.

- Random Forest is a decision tree-based model and actually, the decision tree models perform very well for this problem. The default parameters of the model actually work very well and achieve a score much better than the Linear Regression model but we tried to tune its parameters such as the n_estimators, max_depth, and max_leaf_nodes. By limiting the max_depth, our model's predictions worsen and by increasing the no. of estimators, our model performs a little better, but the computing time increases. Also, we noticed that the Random Forest performs better if we used half of the data to prevent the overfitting problem.
- XGBoost Regressor uses decision trees but with a different technique other than the Random Forest. In the beginning, the XGB model had higher RMSEs than our Random Forest Model so we assumed that the model is under-fitted. So, our first guess would be to increase the no. of estimators in the model. The computing time was massive but, the model is slightly less under-fitted and performs better on the validation set so we reduced the no. of estimators to 1000 estimators which led to an optimum training time. Tuning the learning rate didn't help much. The best combination for the XGBoost was to limit the no. of estimators to 200, max depth to 10, learning rate to 0.2, and min samples to 40. Also, we trained the model on the GPU to boost the training time.

Also, we trained other models that didn't perform much better so we considered them as a failed trails:

- SVM
- Ridge

Conclusion

Model	Data Size	Score (RMLSE)
Linear Regression	1456603	0.61009
Linear Regression	500000	0.58249
Random Forest	100000	0.48705
Random Forest	500000	0.47347
Random Forest	1000000	0.47644
XGBoost ⇒ No Tuning	1456603	0.87776
XGBoost ⇒ Increased Estimators	1456603	0.54441
XGBoost ⇒ Tuning the max depth	1456603	0.50855
XGBoost ⇒ Limiting the estimators, max depth, number of samples, and learning rate	1456603	**0.43993**

The results shows that the best model to tackle the problem using machine learning algorithms is the XGBoost Regressor. This problem is a good one to tickle it using naive neural network algorithms but this is out of scope for this project.