New York City Taxi Trip Duration

I. Definition

Project Overview

Since the invention of the wheel, human civilization has been rising rapidly due to transportation of goods and led to various other innovative technologies such as car, airplane that play an important role in transportation. [1]

Taxi is one of the main transportation for humans. They are fast, comfortable, and have an affordable price for many people. Fortunately, with the rise of computing powers, collecting sensors, and modern algorithms in computer science, it is possible to improve the taxi service by predicting the total ride duration of taxi trips. If we are able to predict the ride duration at any specific time in this large scale system, taxi companies are able to use the prediction time as the input feature for their optimal price model - this is better than naive guess which might lead to overpriced tag. Passengers could also plan and estimate the destination time more accurately; more productive in their life. Lastly, drivers also are able to compare their time and fare to

[1] "The revolutionary invention of the wheel - ḎḤWTY", 2 JUNE, 2014, https://www.ancient-origins.net/ancient-technology/revolutionary-invention-wheel-001713

evaluate their returns for riding in the particular trip. This gives valuable benefits to all drivers, taxi companies, and customers.

Kaggle, well-known platform using data science to solve the problems, has a dataset and challenges you to build a model that predicts the total ride duration of taxi trips in New York City. The dataset is based on the 2016 NYC Yellow Cab trip record data made available in Big Query on Google Cloud Platform. The data was originally published by the NYC Taxi and Limousine Commission (TLC). The data was sampled and cleaned for the purposes of kaggle competition. Based on individual trip attributes, participants should predict the duration of each trip in the test set. [2]

Problem Statement

The objective is to predict the total ride duration of taxi trips and also minimize the difference between the actual predictions and real duration trips as much as possible. Searching for the best algorithms and hyperparameters which minimize the errors of the evaluation metric, Root Mean Squared Logarithmic Error [3] is the solution. As I have stated in the previous paragraph, If we are able to predict the ride duration at any specific time in this large scale system, all drivers, taxi companies, and customers will have a lot of benefits especially more productive life and more reasonable taxi fare than ever.

[2] "Kaggle data source", https://www.kaggle.com/c/nyc-taxi-trip-duration/data
[3] "Root Mean Squared Logarithmic Error ", https://www.kaggle.com/wiki/RootMeanSquaredLogarithmicError

Metrics

The evaluation metric for this problem is Root Mean Squared Logarithmic Error.

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

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, a_i is the actual trip duration for i, and $\log(x)$ is the natural logarithm of x.

The reasons I choose RMSLE for this project are: Official evaluation metric for kaggle competition and it is the standard for most regression problems.

Firstly, using the same evaluation metric that kaggle used in the competition benefits hugely to us because we can now submit our test set predictions to kaggle private leaderboard evaluating how good our model is.

Secondly, RMSLE, in fact, is the alternative version of RMSE which takes the log of both actual and predicted values before calculating. This is very helpful in this project because it does not penalize both huge actual and prediction values the same way as RMSE. For example, if we have 2 scenarios: A) actual trip duration 10 mins, predicted trip duration 15 mins. B) actual trip duration 60 mins, predicted trip duration 65 mins. If the evaluation metric is RMSE, it will yield the exact same error. In contrast, RMSLE would penalize the scenario A more because it

has a smaller value; and that is more suitable for our domain because in scenario A it will surely upset passengers more than scenario B.

II. Analysis

Data Exploration

In this dataset from kaggle, we have 2 files: the training set containing 1458644 trip records, and the testing set containing 625134 trip records. We will focus mostly on the training set since this is our input for machine learning model to learn.

The dataset is based on the 2016 NYC Yellow Cab trip record data made available in Big Query on Google Cloud Platform. The data was originally published by the NYC Taxi and Limousine Commission (TLC). The data was sampled and cleaned for the purposes of kaggle competition. Based on individual trip attributes, participants should predict the duration of each trip in the test set.

Data field	Description [4]
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

[4] "Kaggle NY city taxi trip duration data", https://www.kaggle.com/c/nyc-taxi-trip-duration/data

dropoff_datetime	date and time when the meter was disengaged
passenger_count	the number of passengers in the vehicle (driver entered value)
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

	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

According to basic descriptive statistics, target, trip duration, contains max value at 3526282 seconds or 979 hours which is not feasible. Furthermore, min value is also not probable, 1 second - dealing with this issue is critically important. Vendor id and passenger count are both categorical features and may not contain insights with descriptive statistics. Lastly, longitude and latitude values are positioned in around -74 and 40 which is the regional area of New York - they unfortunately contain some outliers and are not located in New York - removing or keeping should be implemented.

```
In [85]: train.isnull().sum()

Out[85]: id                  0
         vendor_id           0
         pickup_datetime     0
         dropoff_datetime    0
         passenger_count     0
         pickup_longitude    0
         pickup_latitude     0
         dropoff_longitude   0
         dropoff_latitude    0
         store_and_fwd_flag  0
         trip_duration       0
         dtype: int64

In [86]: test.isnull().sum()

Out[86]: id                  0
         vendor_id           0
         pickup_datetime     0
         passenger_count     0
         pickup_longitude    0
         pickup_latitude     0
         dropoff_longitude   0
         dropoff_latitude    0
         store_and_fwd_flag  0
         dtype: int64
```

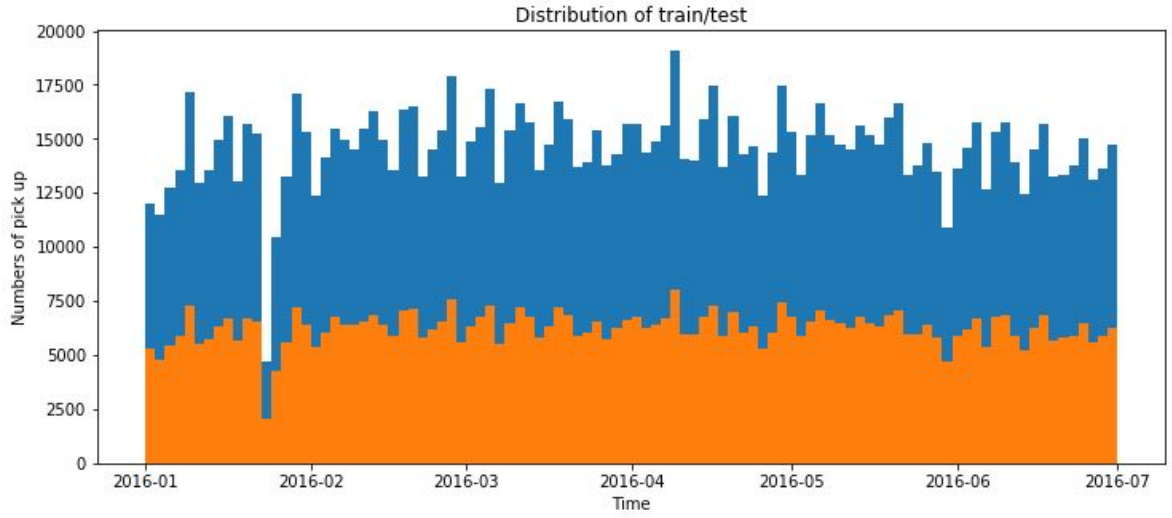
After searching for the null values, no null values exist in this dataset; Kaggle has handled and cleaned all null values before providing this dataset.

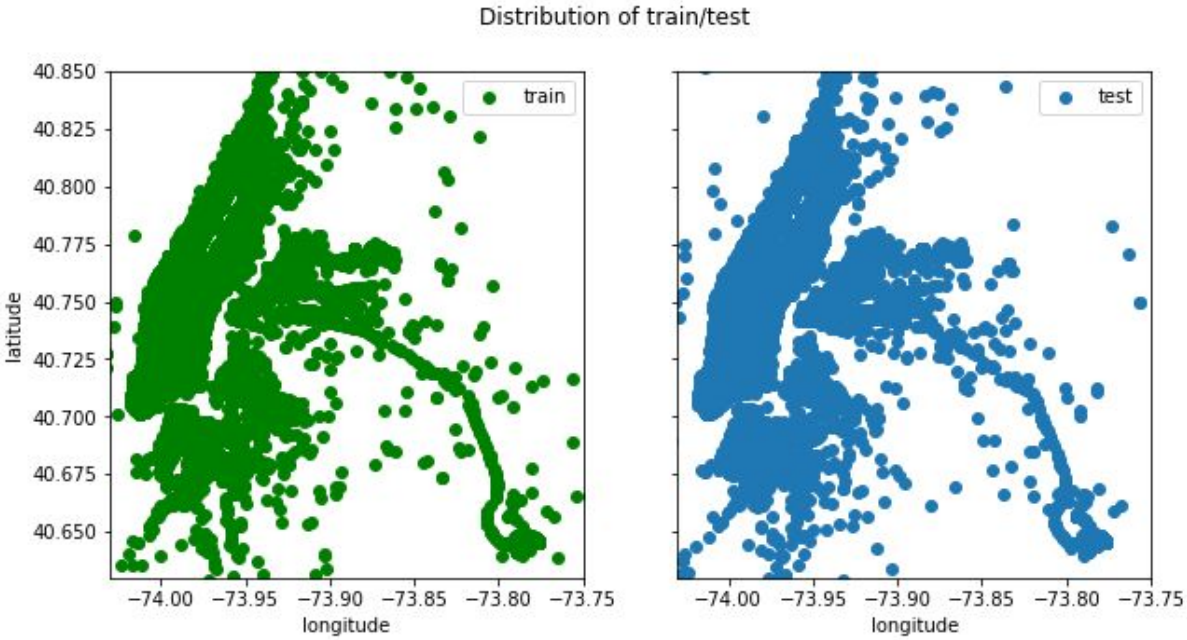
```
In [87]: train.shape == train.drop_duplicates().shape
Out[87]: True

In [88]: test.shape == test.drop_duplicates().shape
Out[88]: True
```

Duplicate values also do not exist in this dataset; Kaggle has dropped and cleaned all duplicate rows before providing this dataset.

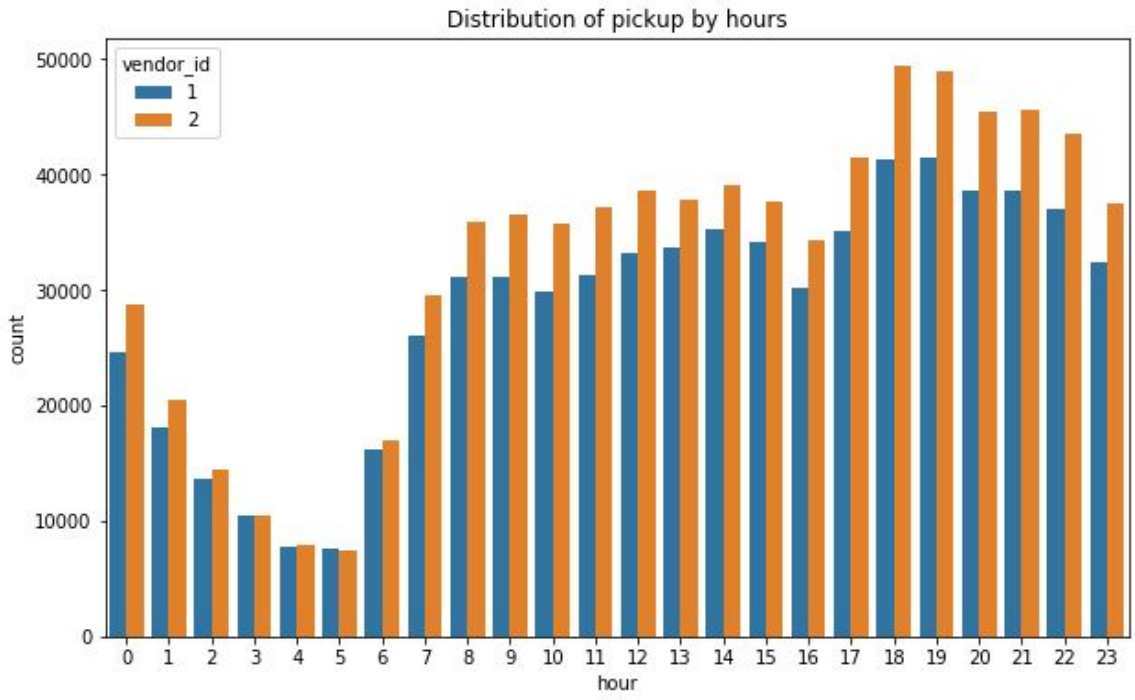
Exploratory Visualization

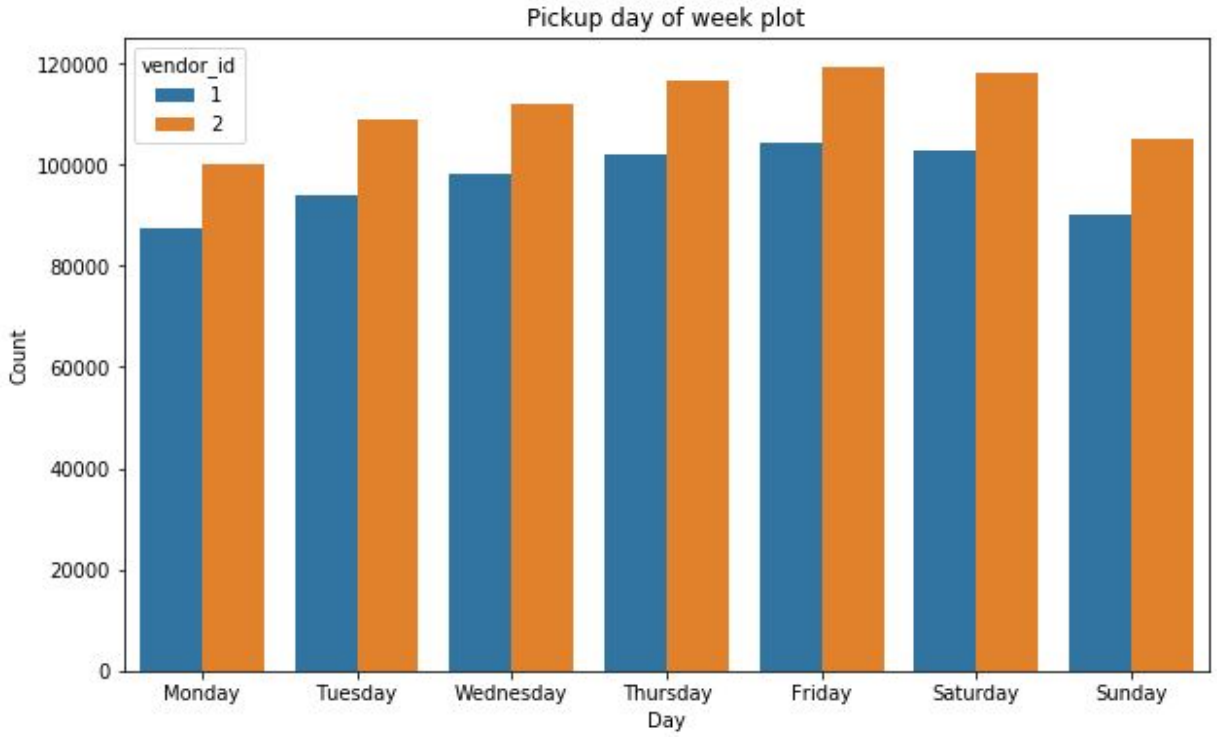
Before proceeding further, Plotting the distribution of training and test set is essential for machine learning process, If the training and test set have hugely different data points/datetime, machine learning model is likely to perform worse in the test set and data scientists are not be able to optimize or evaluate the model efficiently - scientists have to distribute and split training and test dataset again.

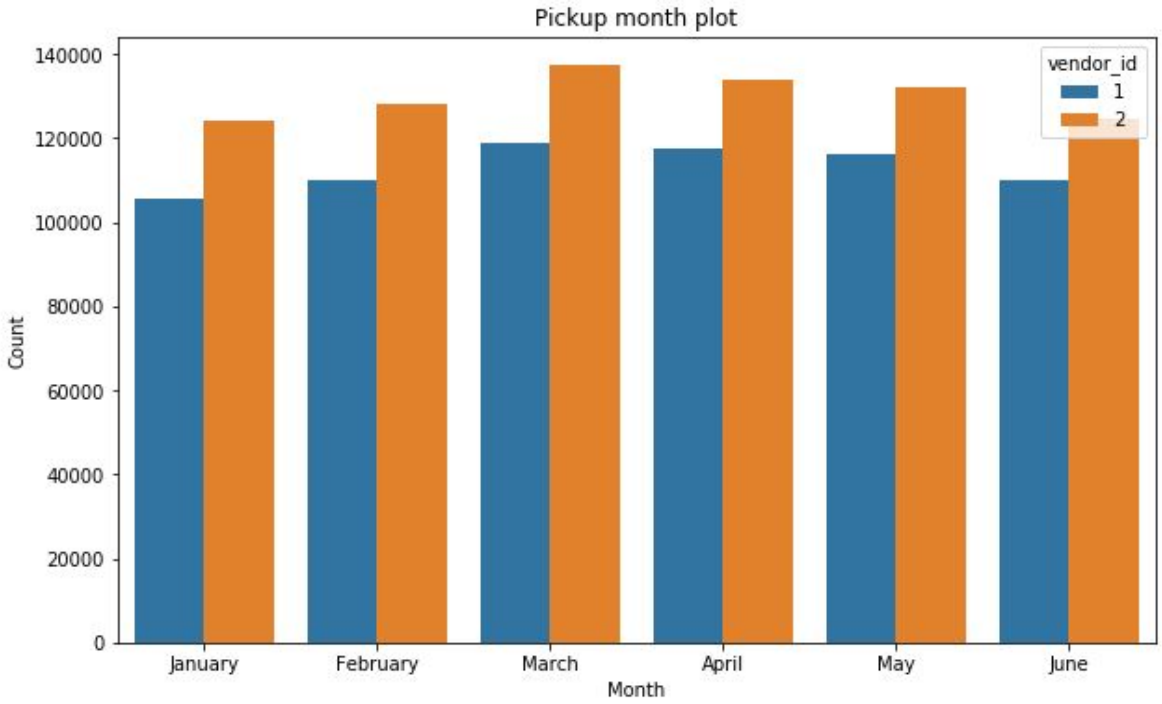




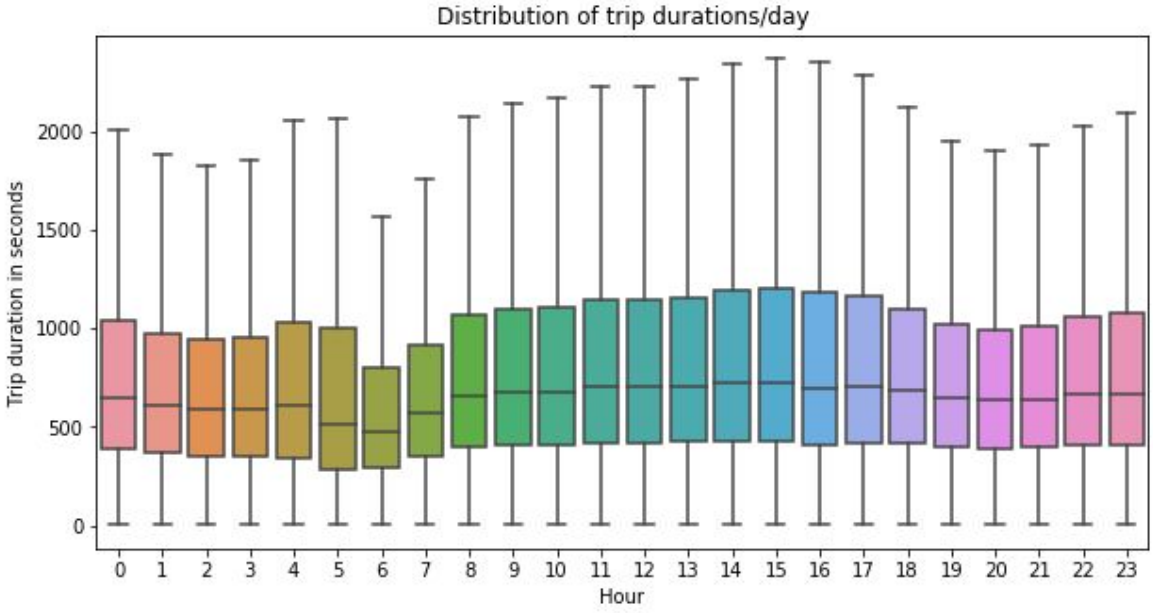
According to distribution of training and test set above, distributing is fairly uniform. Both training and test set have the same distribution which is great for evaluating machine learning model later. Especially, the distribution of datetime is very well splitted since it is from the same time period and we can expect a robust model because of this carefully splitted dataset.


Distribution of pickup by hours
vendor_id
1
2
count
hour
0
10000
20000
30000
40000
50000
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23

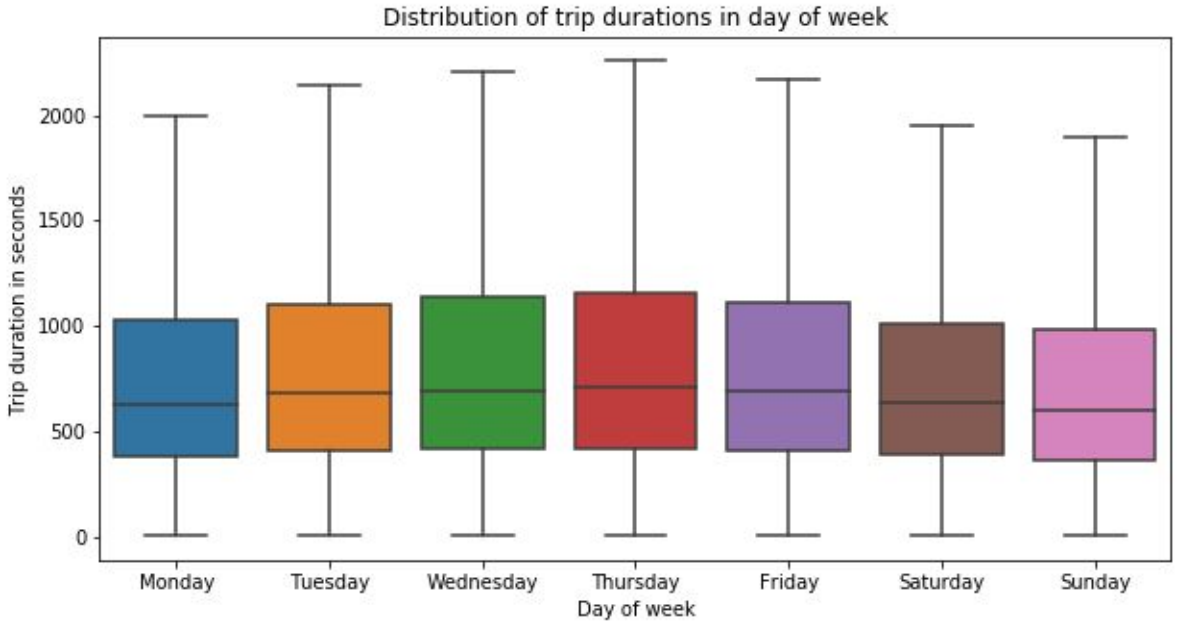

Pickup day of week plot
vendor_id
1
2
Count
Day
0
20000
40000
60000
80000
100000
120000
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday



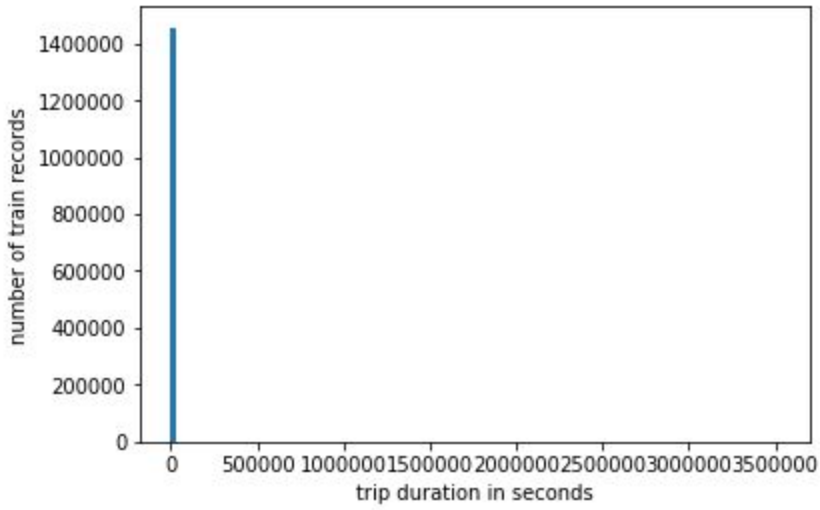
After several plots focusing heavily on vendor id and datetime features, both vendor id 1 and 2 have the same distribution and do not significantly show any evidences containing valuable insights.

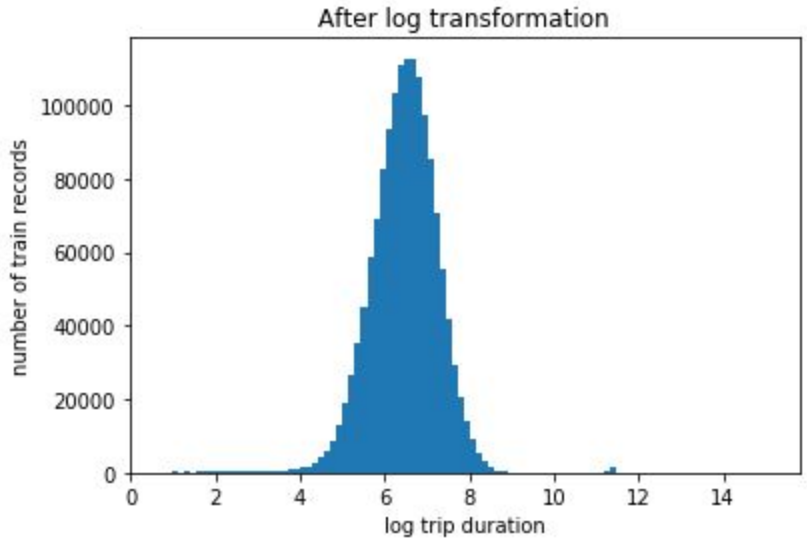


Based on distribution of trip durations/day above, around 6 - 7 am has the lowest average trip duration. This is reasonable because in the morning, the traffic is not heavy and has lower number of cars on the road comparing to other time period throughout the day. In addition, after the prime time (8 am - 6 pm), the trip duration time is slightly decreasing because a number of people already go back home leading to less cars on the road.



According to plot shown above, Saturday and Sunday both have the lowest average trip duration time - many people do not have to work and stay home leading to less cars on the road.





According to the Data Exploration section above, I mentioned that trip duration, the target, contains outliers and may be affected our regression performance. Thus, after performing log transformation, to clean and deal with some outliers, the target value now are much more normally-distributed and ready to take as an input for machine learning models.

Algorithms and Techniques

For my final model, I have used the XGboost library which is one type of the gradient boosted trees. I have decided to use this algorithm with various reasons:

1. It has the best performance on this problem according to " 02 - Picking the Best Algorithm notebooks ".
2. It is able to handle the non-linear relationships between many features because of the nature of the decision tree and still has the great performance.
3. It can prevent the overfit easily, with the concept of boosting and early stopping which at the same time increases the performance and also stop training whenever the model risks on overfitting.

In layman's term, the model works as following:

Gradient boosting tree is the idea to combine multiple weak learners to be the strong learner by using gradient descent technique to optimize and tune hyperparameters for each tree. First of all, we start with 1 decision tree - this first tree doesn't have to be perfect. Then, the decision tree will receive some random features/ samples and try to limit the depth of tree, numbers of leafs, etc. After training and fitting, expectedly, this single tree model's performance is not quite good. So we construct the new tree with prior knowledge from the last one (the first tree in this case) and use the gradient descent technique to slightly optimize weights of hyperparameters for this new tree to be better than the previous tree. This is why Gradient boosting tree is very powerful because they try to learn from the mistakes of the previous trees and use that lessons to apply to the current tree. We repeat this process until it reaches

the given numbers of trees.

Some of the xgboost parameters that we need to be optimized and searched [5]:

- eta [default=0.3] : Step size shrinkage used in update to prevents overfitting. After each boosting step, we can directly get the weights of new features, and eta shrinks the feature weights to make the boosting process more conservative.
- max_depth [default=6] : Maximum depth of a tree. Increasing this value will make the model more complex and more likely to overfit. 0 indicates no limit.
- min_child_weight [default=1] : Minimum sum of instance weight (hessian) needed in a child. If the tree partition step results in a leaf node with the sum of instance weight less than min_child_weight, then the building process will give up further partitioning. In linear regression task, this simply corresponds to minimum number of instances needed to be in each node. The larger min_child_weight is, the more conservative the algorithm will be.
- colsample_bytree [default=1] : Subsample ratio of columns when constructing each tree. Subsampling will occur once in every boosting iteration.
- subsample [default=1] : Subsample ratio of the training instances. Setting it to 0.5 means that XGBoost would randomly sample half of the training data prior to growing trees. and this will prevent overfitting. Subsampling will occur once in every boosting iteration.
- gamma [default=0] : Minimum loss reduction required to make a further partition on a leaf node of the tree. The larger gamma is, the more conservative the algorithm will be.

[5] “Xgboost Parameters”, https://xgboost.readthedocs.io/en/latest/parameter.html

- alpha [default=0] : L1 regularization term on weights. Increasing this value will make model more conservative.

Benchmark

At first glance, I was tempted to use mean prediction as the baseline model because of its simplicity. However, it is not a good baseline in this setting because the duration time has a high variance and the model would predict too naively which make it not suitable to use as a baseline model to beat. Using plain **linear regression** might be more suitable for this domain problem because most of the regression problems have been using this model since early 20th century and its performance is also great for numerical problems - it is the foundation of most modern machine learning algorithms even deep learning. Another reason to use linear regression as baseline model is easily measurable; RMSLE with linear regression in this case is very easy to compare with our solution.

- RMSLE with linear regression on validation set is **0.60291**
- RMSLE with linear regression on public leaderboard (30% of test set) is **0.60910**
- RMSLE with linear regression on private leaderboard (70% of test set) is **0.72923**

III. Methodology

Data Preprocessing

Store and fwd flag : a boolean and converting to binary number is better for machine and models.

```
# Convert store_and_fwd_flag to numerical values
train['store_and_fwd_flag'] = train.store_and_fwd_flag.map({'Y':1, 'N':0})
test['store_and_fwd_flag'] = test.store_and_fwd_flag.map({'Y':1, 'N':0})
```

Passenger_count : It looks like training set has passengers values '7' and '8' which are not in test set. We must remove that in training set because lately we are going to use technique called one hot encoding and both dataset must have the same values.

```
# Remove passenger 7 and 8 and verify the results
train = train[(train.passenger_count != 7) & (train.passenger_count !=8)]
```

Datetime : We are going to extract valuable information from datetime and construct new features such as day of week, day, hour, month, is holiday or not.

```
us_holidays = holidays.UnitedStates()

train['month'] = train.pickup_datetime.dt.month
train['day_of_week'] = train.pickup_datetime.dt.weekday
train['hour'] = train.pickup_datetime.dt.hour
train['day'] = train.pickup_datetime.dt.day
train['is_holiday'] = train.pickup_datetime.apply(lambda x: x in us_holidays)
```

Weather : We will use the external data derived from https://www.kaggle.com/cabaki/knycmetars2016 and merge together with our training and test set in order to adding more information of weather conditions based on that specific time period.

```
weather = pd.read_csv('data/KNYC_Metars.csv', parse_dates=['Time'])
weather['year'] = weather['Time'].dt.year
weather['month'] = weather['Time'].dt.month
weather['day'] = weather['Time'].dt.day
weather['hour'] = weather['Time'].dt.hour
weather = weather[weather.year == 2016][['month', 'day', 'hour', 'Conditions']]
```

Clustering *:* We are going to extract valuable information by grouping them into neighborhoods. This will help our model learn much more easily by using unsupervised learning technique called Kmeans.

```
# Kmeans
kmeans = MiniBatchKMeans(n_clusters=100, batch_size=10000).fit(lat_lng)

train['pickup_cluster'] = kmeans.predict(train[['pickup_latitude', 'pickup_longitude']])
train['dropoff_cluster'] = kmeans.predict(train[['dropoff_latitude', 'dropoff_longitude']])
```

Computing the distance : Another interesting idea we can do with latitude and longitude is to calculate the distance between 2 points. There are several ways to calculate the distance. I have decided to use three methods: Manhattan distance(L1 distance) , Euclidean distance (L2 distance) and haversine distance.

The Manhattan distance is the method to measure the distance of only four cardinal directions, or cardinal points, which are the directions north, east, south, and west.

$$L1\ distance(a,b) = |a_{lat} - b_{lat}| + |a_{lon} - b_{lon}|$$

The Euclidean distance is the method to measure with straight-line distance between two points.

$$L2\ distance(a,b) = \sqrt{(a_{lat} - b_{lat})^2 + (a_{lon} - b_{lon})^2}$$

Haversine formula is the method to measure the distance based on the curvature of the earth's surface. [6]

$$\Delta\hat{\sigma} = 2\arcsin\left(\sqrt{\sin^2\left(\frac{\Delta\phi}{2}\right) + \cos\phi_s \cos\phi_f \sin^2\left(\frac{\Delta\lambda}{2}\right)}\right)$$

$\Delta\sigma$	Interior Spherical Angle
$\Delta\phi$	Latitude1 - Latitude2
ϕ_s	Latitude1
ϕ_f	Latitude2
$\Delta\lambda$	Longitude1 - Longitude2

Trip duration : the target, contains outliers and may be affected our regression performance. Thus, after performing log transformation, to clean and deal with some outliers, the target value now are much more normally-distributed and ready to take as an input for machine learning models.

```
train['log_trip_duration'] = np.log1p(train.trip_duration.values)
```

[6] "Measure the distance on google map using euclidean and haversine", https://bsierad.com/measure-the-distance-on-google-map-using-euclidean-and-haversine/

Convert categorical features using one hot encoding and drop irrelevant features : many models cannot deal directly with categorical features for example they might see passenger_count 6 is closer and more relevant to 5 or 7 than other numbers.

```
dummies = ['vendor_id',
           'passenger_count',
           'store_and_fwd_flag',
           'month',
           'day_of_week',
           'hour',
           'is_holiday',
           'Conditions',
           'pickup_cluster',
           'dropoff_cluster']


train = pd.get_dummies(train,columns=dummies)
train.drop(['id',
            'pickup_datetime',
            'dropoff_datetime',
            'trip_duration',
            'day'], inplace=True, axis=1)
```

Implementation

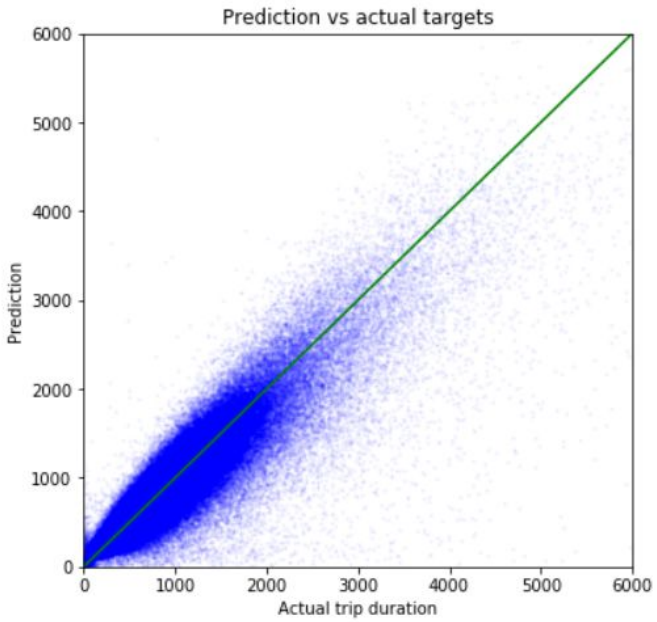
IV. Results

Model Evaluation and Validation

As mentioned in the last section, the final xgboost model with new hyperparameters performs better than the old parameters, improved from **0.4067** to **0.3891** in local validation set. Furthermore, the final xgboost model still performs quite impressive on the unseen data, test set which is never touched in the entire process with the score **0.39262** on public leaderboard (30% of the test set) and **0.38941** on private leaderboard (70% of the test set).

One interesting thing from the final model is that most parameters used for controlling or preventing overfit is generally high values. This is a great signal that this xgboost model could handle unseen data well because of strictly regulated trees.

However, visualization is still critically important. I decide to make a simple plot to show how well model prediction is.



Model generally could predict correctly or close to the true value especially the lower and around mean value of trip duration. One thing to note is that xgboost model never overestimates trip duration values but only underpredicts the targets - there are still some rooms to improve. However, based on the plot above, most of the time, final xgboost model is robust and handle most of the input data very well.

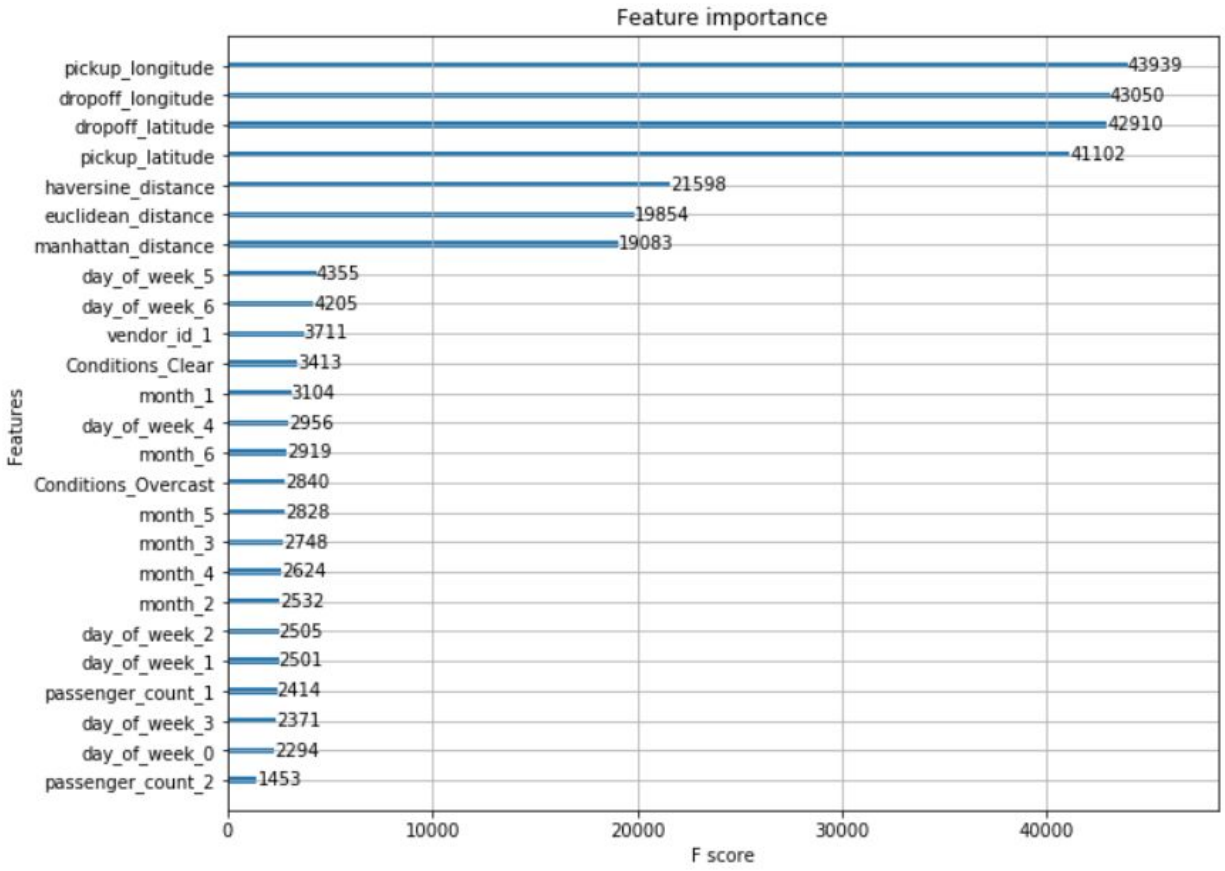
Justification

Comparing to benchmark, the final model has improved our solution a lot. It improves our rmsle score from **0.6029** from validation set to **0.3891** and the 70 % unseen data (test set) from **0.7292** to **0.3894** which is impressive and robust.

In conclusion, the final model is stronger than benchmark model and it is good enough to solve this problem as the plot shown in the last section - final model could generally predict closely to the true value and also does not face with the overfit issue on the unseen data by carefully picking the regularized hyperparameters along with early stopping technique.

V. Conclusion

Free-Form Visualization



One of the great thing tree based model can help us is to display the feature importance based on the f score, how each feature is important to our model.

According to the feature importance plot above, it seems like longitude and latitude are the most important feature; even our three distance features still rank very high comparing to other features. This shows us that we have to construct new features mostly based on longitude and latitude.

Another interesting point are day_of_week_5 and 6 which are Saturday and Sunday. This has confirmed my hypothesis during data visualization that Saturday and Sunday both have the lowest average trip duration time - many people do not have to work and stay home leading to less cars on the road. Our model can capture this pattern and use this a lot in final model.

The weather conditions also play a major role in the model. “ Conditions Clear ” is ranked very high - this is reasonable because without bad weathers, the shorter trip duration.

Reflection

The entire end-to-end problem solution can be summarized as following:

1. The input data is preprocessed such as converting features to proper types
2. Constructing new features from current features using some domain knowledge such as computing distances.
3. Trip duration is transformed to be more normal distribution by applying log transformation.
4. Categorical features are transformed by one hot encoding method.
5. Xgboost model with well-chosen hyperparameters are trained with early stopping.
6. Final model is ready to predict the unseen data.

The most interesting aspect of this project is to construct the new features based on the existing features. I think with current feature engineering, there are still plenty of rooms to improve or drop out of the input data. Also, the most difficult part in this project is to deal with feature engineering. Without any domain knowledge and experience in this industry, I cannot figure a lot of possibilities to create new features. The another problem is that I do not have enough of time to do some experiments with building the new features.

In conclusion, the final xgboost model generally predicts trip duration well based on our scores and the plot in the Model Evaluation and Validation section. We started with the linear

regression model that has fairly average score for rmsle and we have been tremendously improved from that point, through Ridge regression, Random Forest and Xgboost.

Improvement

There are plenty of rooms to improve after analysing the feature importances and model prediction versus actual targets.

Firstly, constructing new features based on latitude and longitude may improve the score dramatically. For instance, implementing OSRM [7](Open Source Routing Machine) to find the fastest route may improve the score or calculating the average speed based on that specific time and use them as the new features.

Secondly, Ensemble learning will improve the score surely by combining various low-correlated predicted model together and use the technique called model stacking [8] to learn. However, The downside of this solution is that it is not practical in the real world since the training time and prediction time are higher and mostly not worth that big tradeoff between time and accuracy.

Thirdly, Dropping irrelevant features may improve the score such as our clustering feature seem not contribute to the model at all and may be seen by the model as the noise. Also, reducing the numbers of features could solve the curse of dimensionality [9].

---

[7] "Open Source Routing Machine", https://wiki.openstreetmap.org/wiki/Open_Source_Routing_Machine
[8] "A Kaggler's Guide to Model Stacking in Practice - Ben Gorman", 27 December 2016, http://blog.kaggle.com/2016/12/27/a-kagglers-guide-to-model-stacking-in-practice/
[9] "Curse of dimensionality", https://en.wikipedia.org/wiki/Curse_of_dimensionality