B9DA103 - DATA MINING

PREDICTION OF TAXI TRIPS DURATION IN NEW YORK CITY USING CRISP-DM METHODOLOGY

Lecturer

TERRI HOARE

By

Mayank Ketan Gandhi

Student number: 10394669

Table of Contents

1. INTRODUCTION

The primary objective of this project is to implement CRISP-DM methodology using Rapid Miner to predict the duration of taxi rides in New York City based on features like trip coordinates or pickup date and time. We will build a predictive framework that is able to infer the trip time of taxi rides. The result of such framework will be travel time of a taxi trip.

2. BUSINESS UNDERSTANDING

The main issue with New York City Taxi is the identification of which driver to assign to each pickup request and this issue can be solved by using Data mining tools like rapid miner to apply some machine learning algorithms that will allow us to predict how long a driver will have his taxi occupied by simply predicting the journey duration and this prediction will directly improve the efficiency of electronic taxi dispatching systems.

2.1 Stakeholders

The most important stakeholder as far as taxi industry is concerned is the **customer base**. The customer base is responsible for the revenue that these businesses generate. Second stakeholder will be the **government**, since it acts as a steward for the land on which these aggregators will conduct their business. Finally, the **employee** of these companies are also important stakeholders in this situation.

2.2 Potential Analytics Solution:

Prediction of Taxi Trip Duration of taxi rides in New York that will help to improve the efficiency of Electronic Taxi Dispatching Systems which will decrease time loss and increase the profits.

2.3 Refined Problem Statement:

To predict the duration of taxi rides in New York City based on features like trip coordinates or pickup date and time.

2.4 Benefits:

Once we predict the duration of taxi trips, we can use the results to dispatch vacancies of the taxis. This will help us to reduce waiting time of a customer for a taxi ride and which in turn will increase profits for taxi company.

3. DATA UNDERSTANDING

The dataset is taken from Kaggle's "NEW YORK CITY TAXI TRIP DURATION" Competition. The dataset comes in the shape of 1.6 million observations and 11 variables. Each row contains information of one taxi trip.

Dataset: https://www.kaggle.com/c/nyc-taxi-trip-duration/data

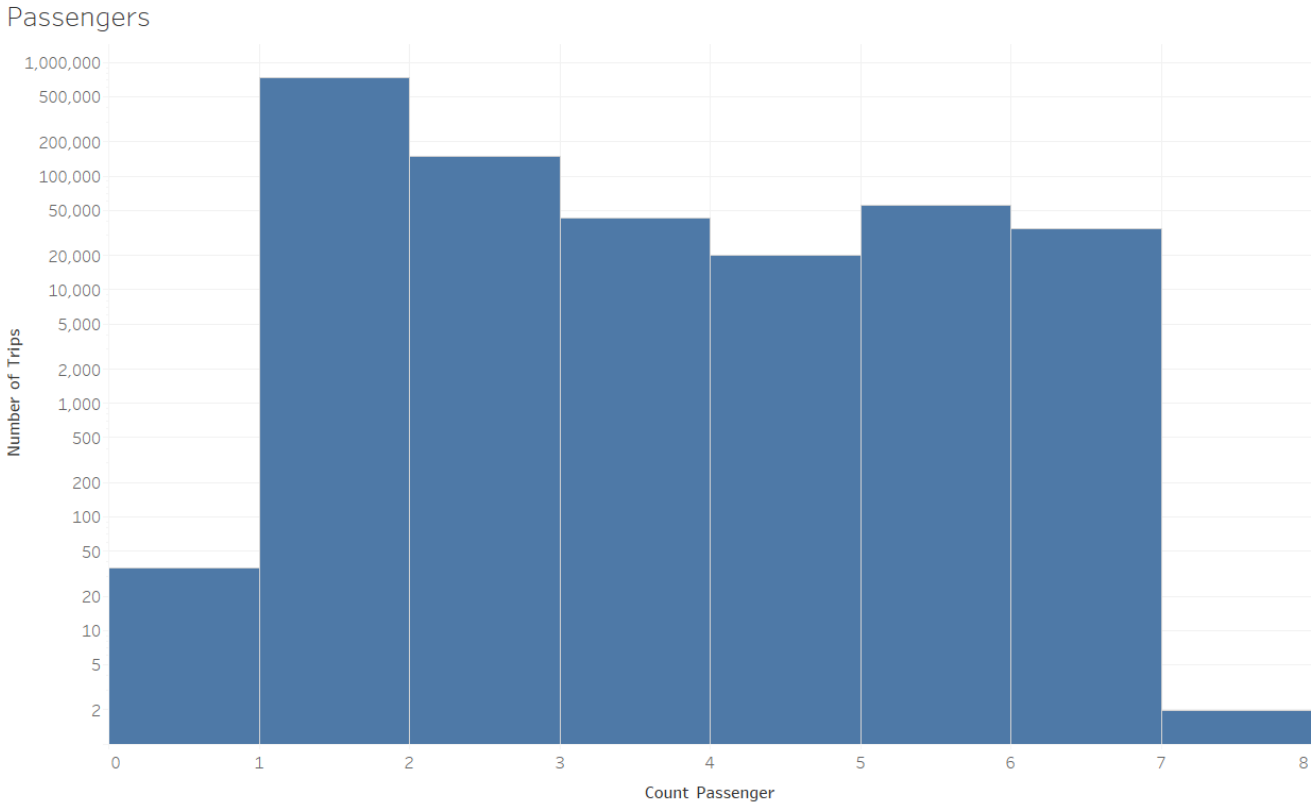
3.1 Data fields

The dataset has 11 variables, in which **"trip_duration"** is the independent/target variable.

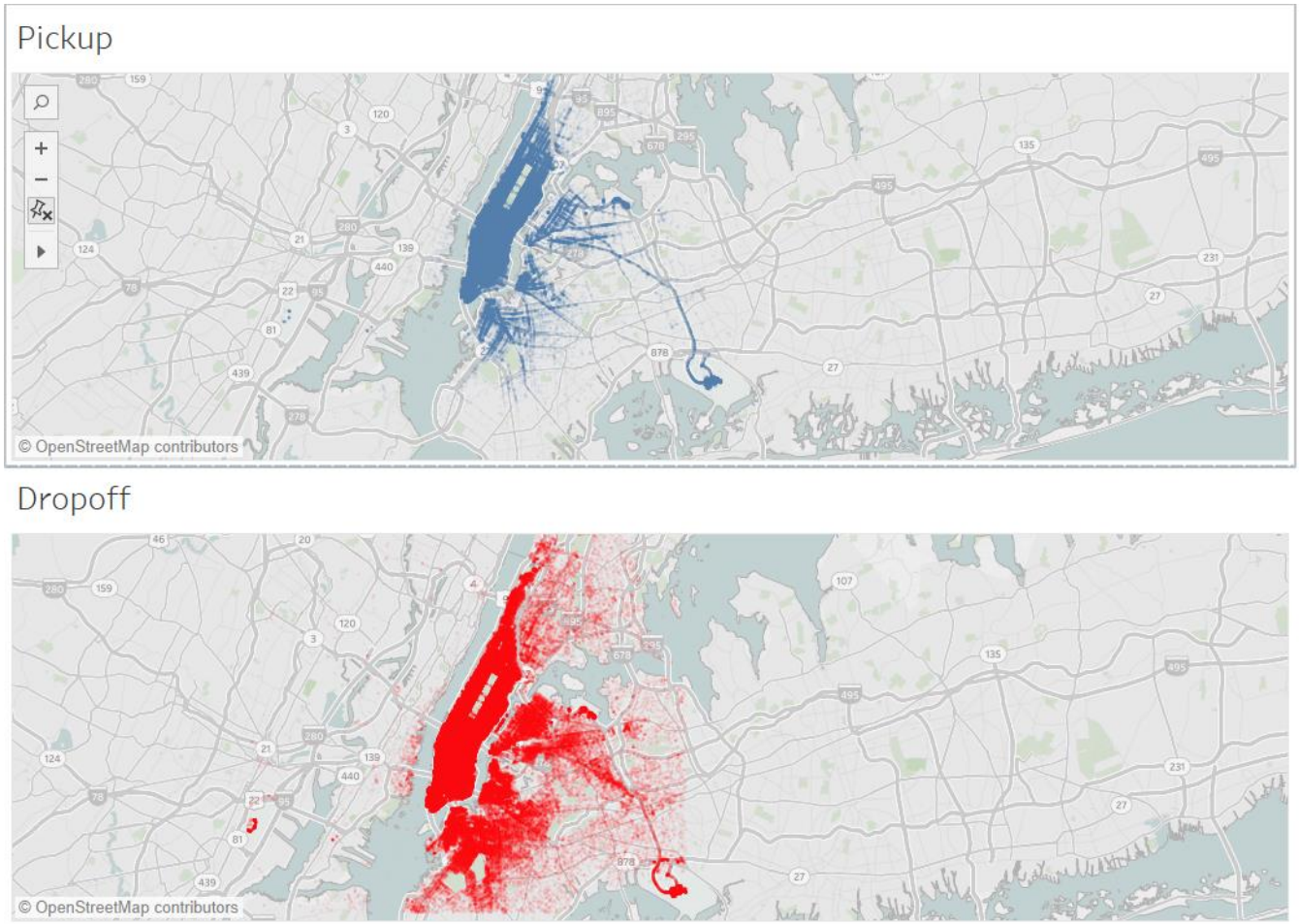
- **id** - a unique identifier for each trip
- **vendor_id** - a code indicating the provider associated with the trip record

- **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 (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 – **(Target variable)**

3.2 Explanatory Data Analysis



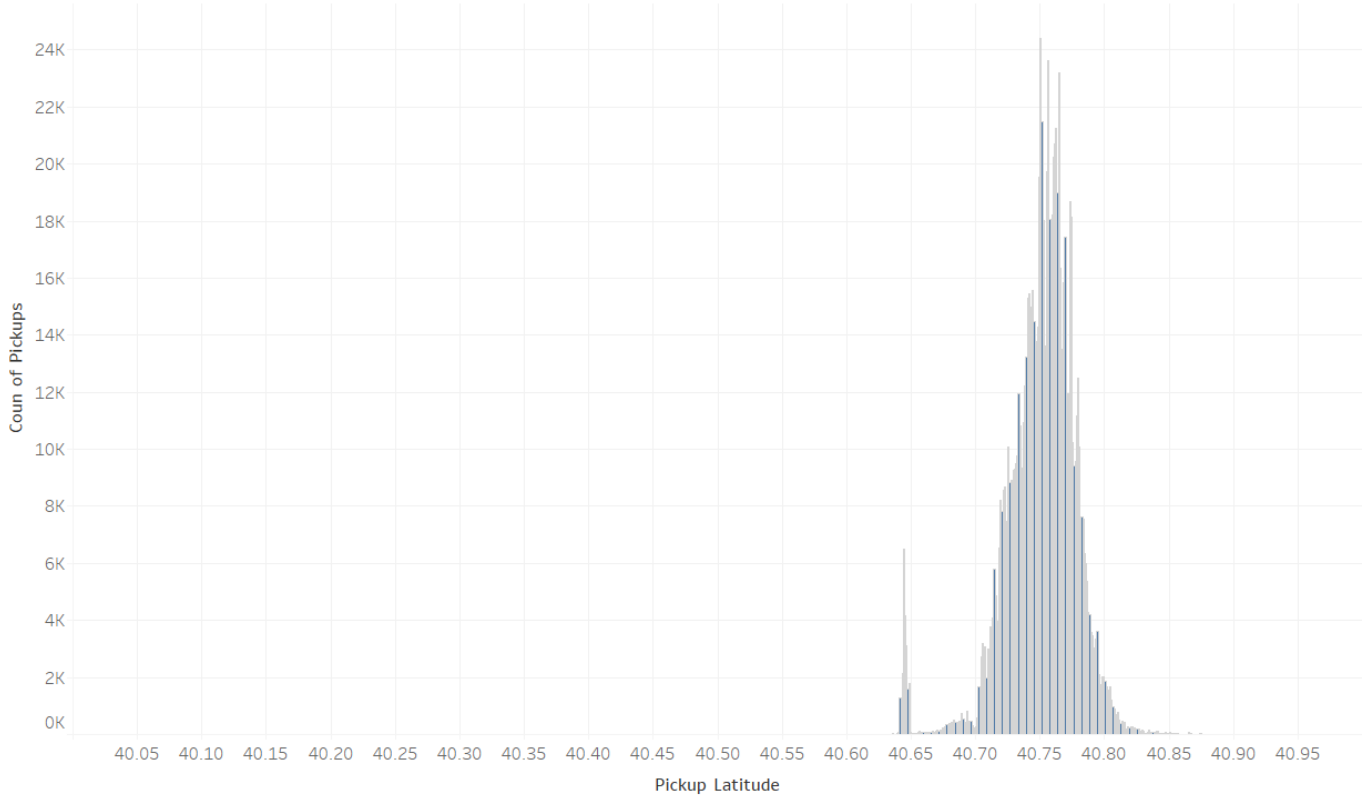
The above bar graph shows us the count of passengers on average for taxi trips. Most of the taxi trips has 1-2 passengers. However, we can see that there are very less trips with more than 6 passengers. Passengers count is important for us as it affects the price of taxi trip and taxi trip duration.



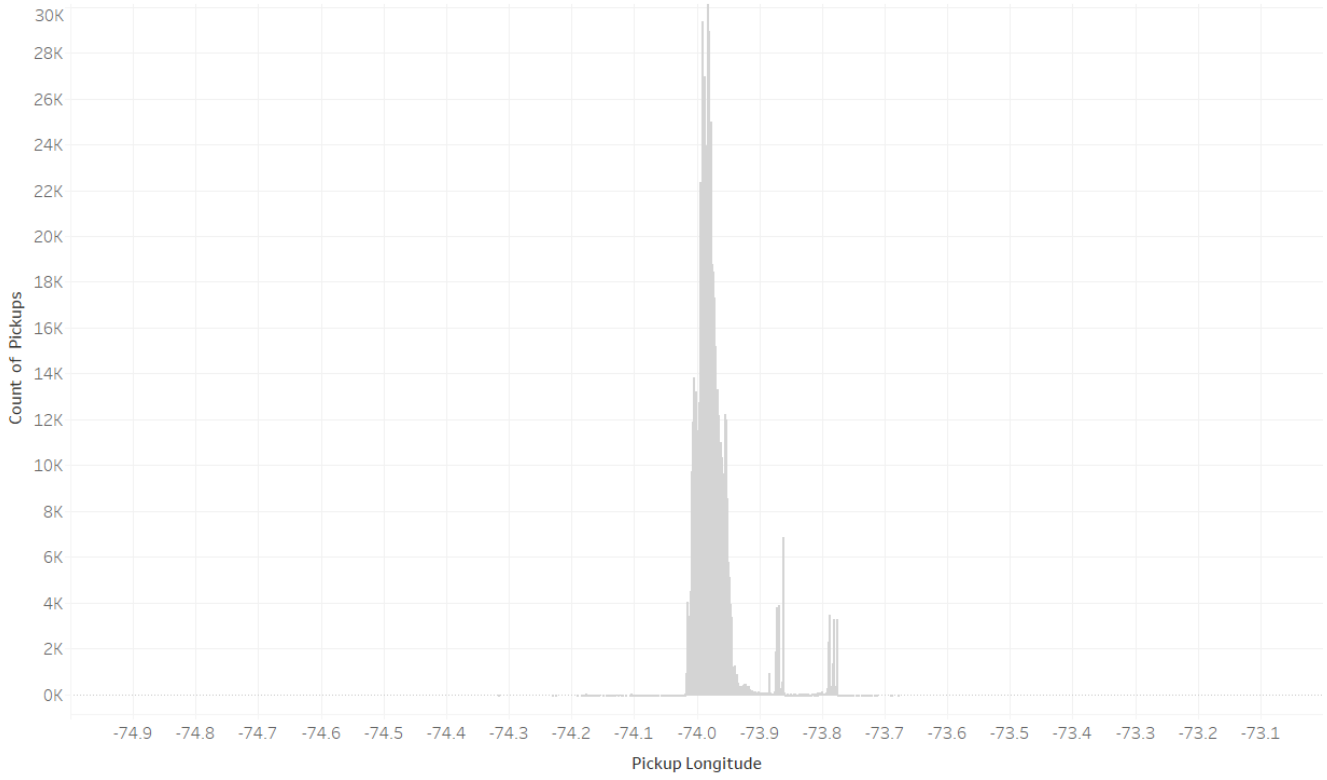
The map illustrates the pick-up and drop-off locations. We could see how the map varies with the time of the pick up and drop. The map shows us that there are many drop-offs outside of New York.

The below 2 graphs represent latitudes and longitudes of New York. We can eliminate the outlying latitudes and longitude as we are considering only taxi trips with New York City.

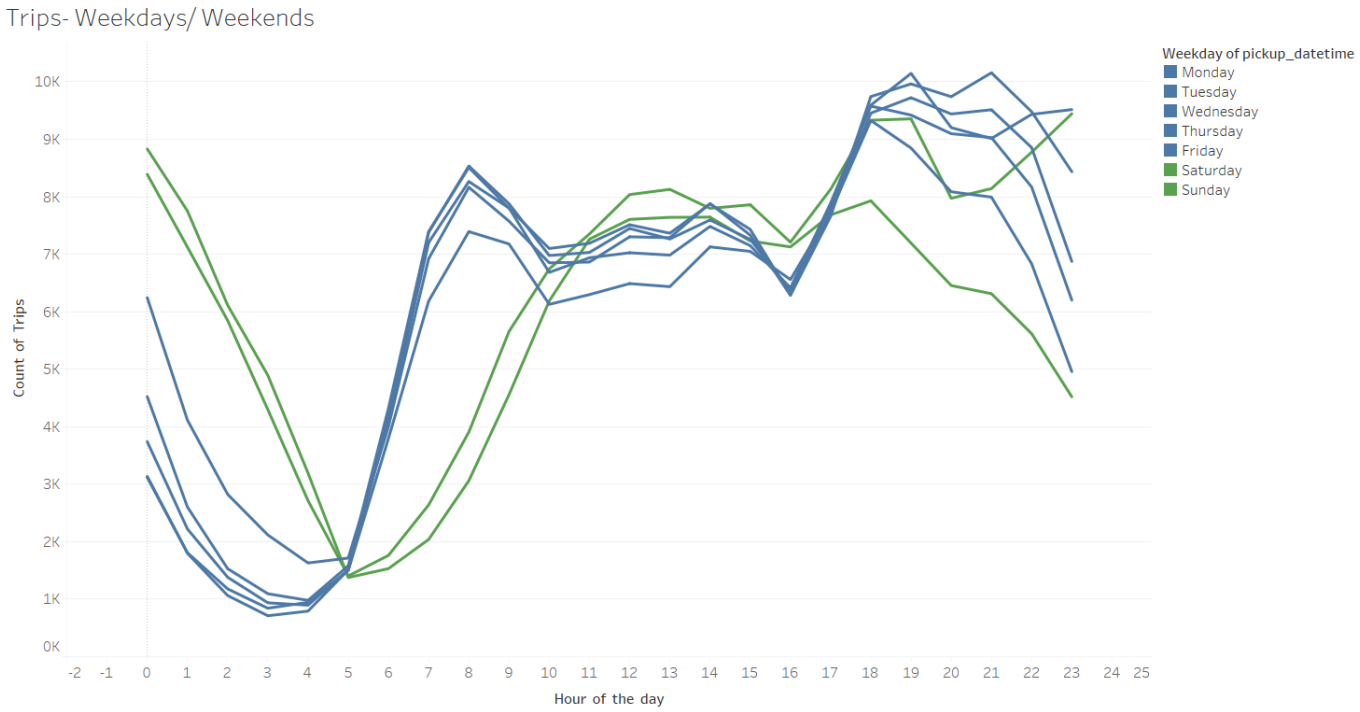
PickupLat



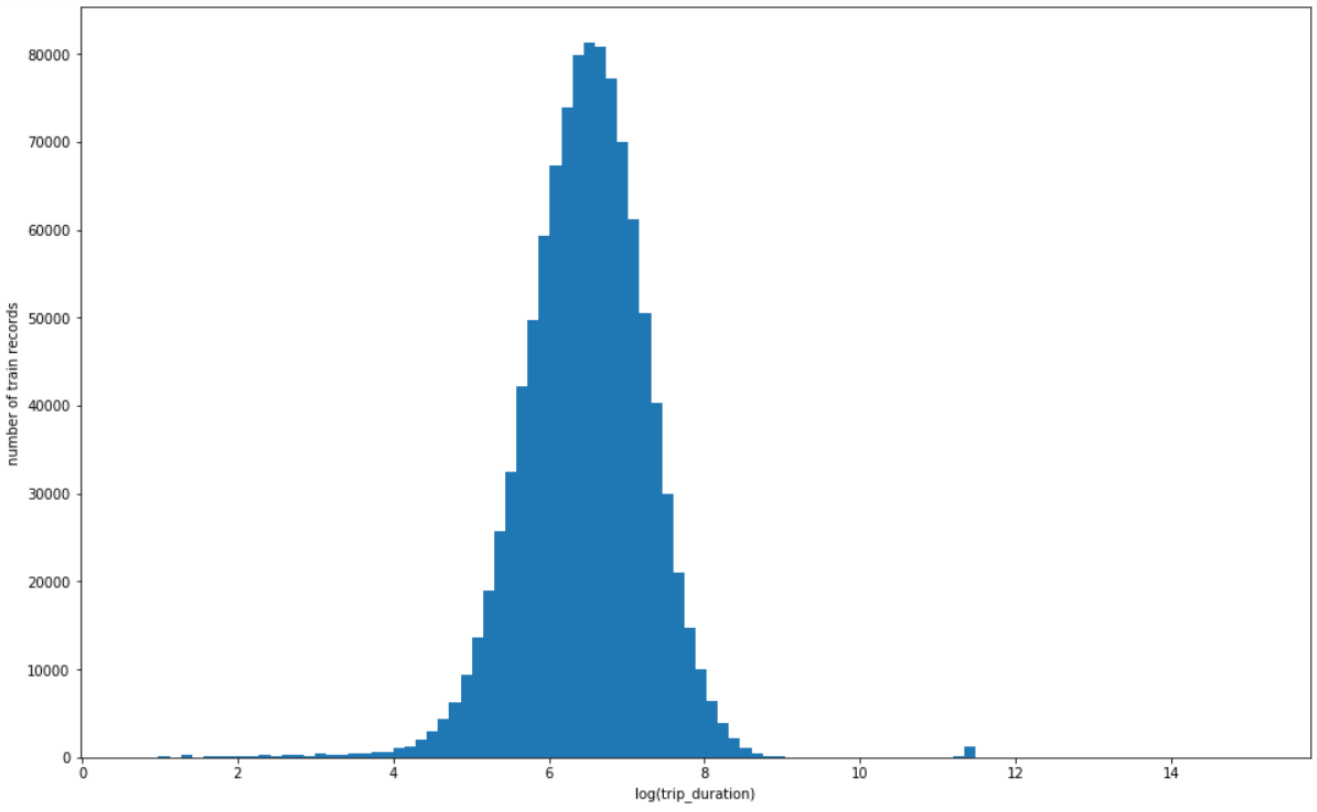
PickupLong

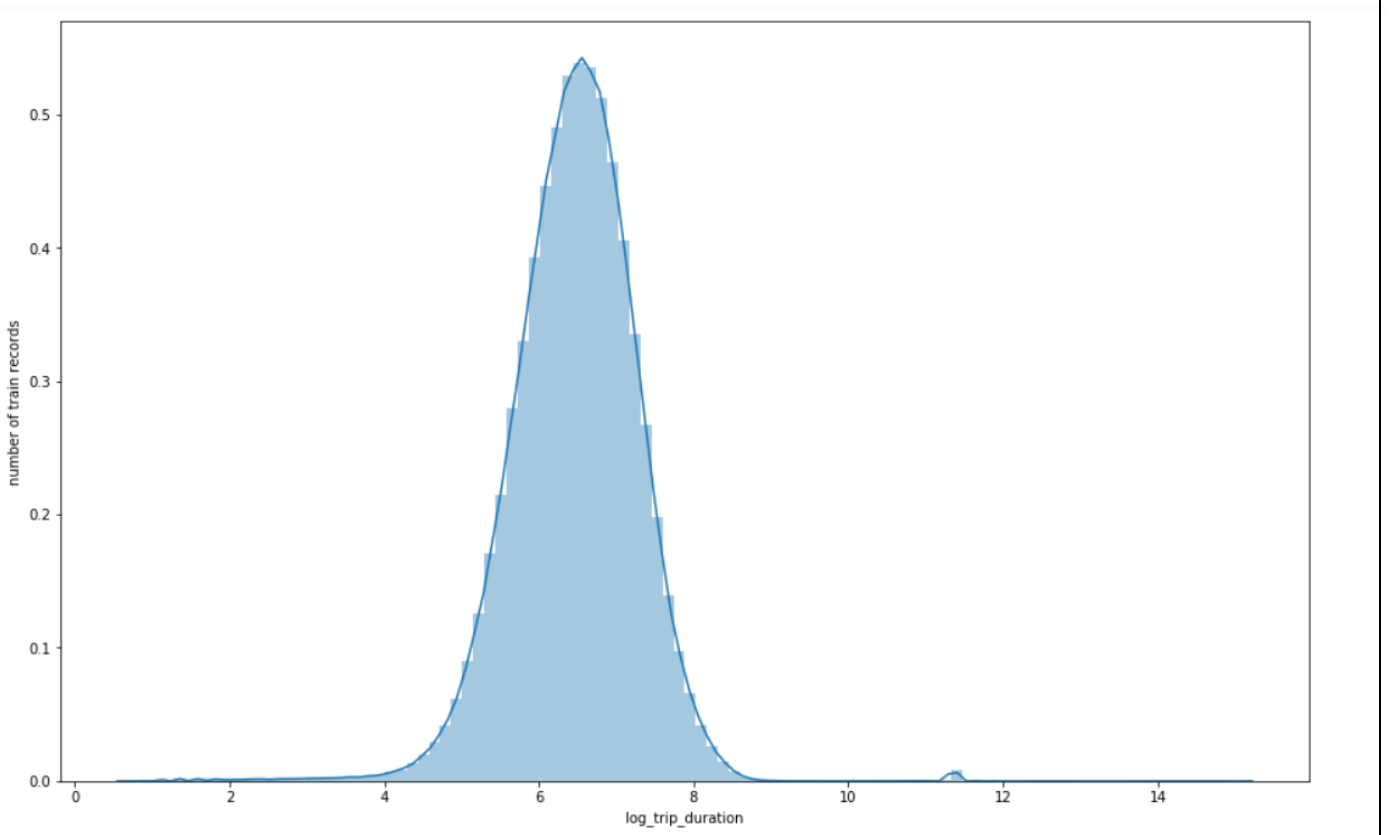


The line graph shows us that the weekend (Sat and Sun, plus Friday to an extend) have higher trip numbers during the early morning hours but lower ones in the morning between 5 and 10, which can most likely be attributed to the contrast between NYC business days and weekend night life. In addition, trip numbers drop on a Sunday evening/night.

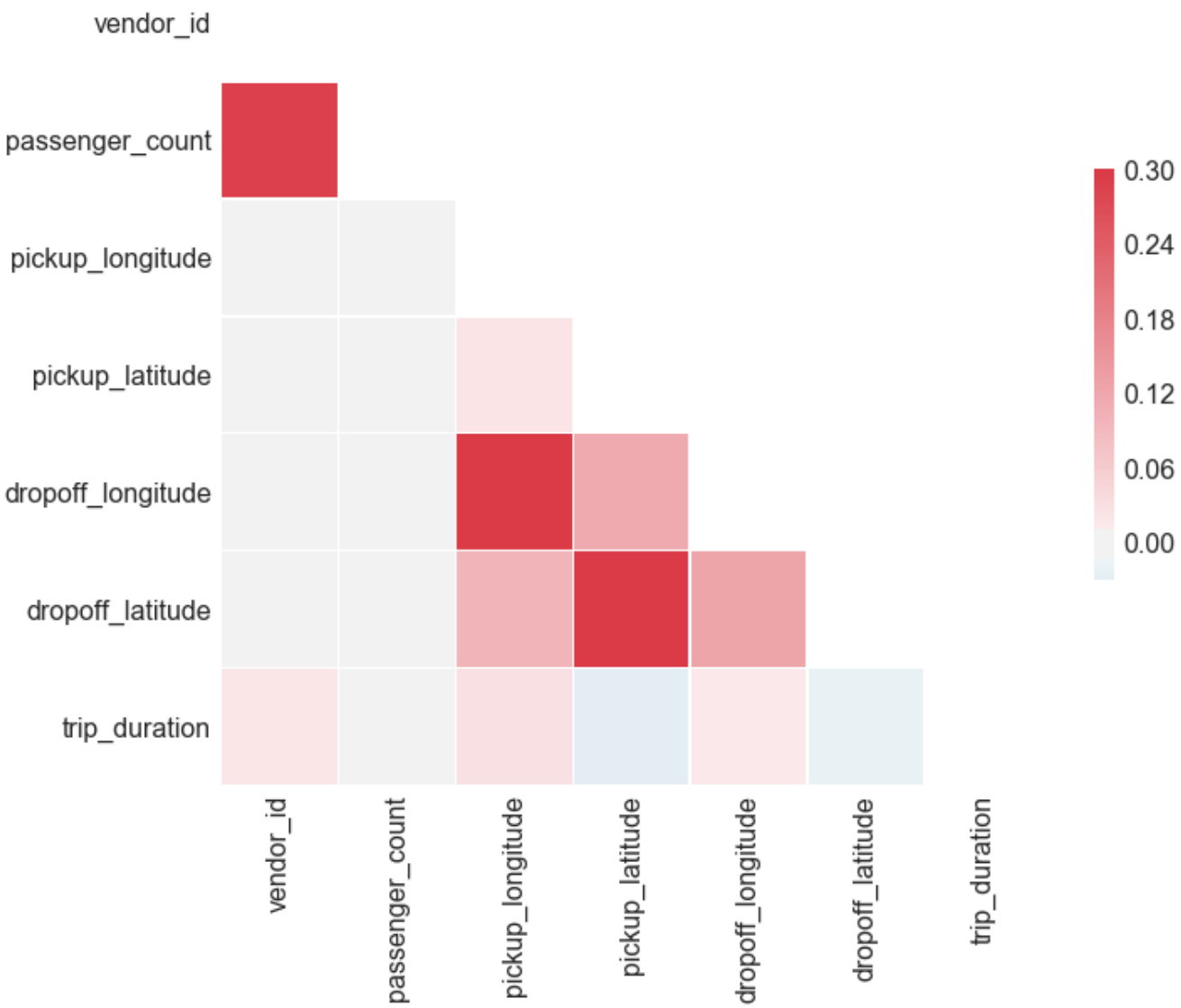


The average trip duration of taxi trips can help us predicting the taxi trip duration. So, we use NumPy's log feature to regularize our target variable (trip_duration), so that it is easier to visualize. The below graphs illustrate log of trip duration.




number of train records
log_trip_duration
0.0
0.1
0.2
0.3
0.4
0.5
0
2
4
6
8
10
12
14

The correlation graph shows us the relationship between the variables. As we can see, we don't have highly correlated variables in our dataset. So we do not have to remove any variables from our dataset.



4. DATA PREPARATION

A brief look on the dataset, with the help on python code we get fields and their type from the dataset.

```
In [28]: train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1048575 entries, 0 to 1048574
Data columns (total 14 columns):
id                  1048575 non-null object
vendor_id           1048575 non-null int64
pickup_datetime     1048575 non-null object
dropoff_datetime    1048575 non-null object
passenger_count     1048575 non-null int64
pickup_longitude    1048575 non-null float64
pickup_latitude     1048575 non-null float64
dropoff_longitude   1048575 non-null float64
dropoff_latitude    1048575 non-null float64
store_and_fwd_flag  1048575 non-null object
trip_duration       1048575 non-null int64
distance            1048575 non-null float64
Time in hrs         1048575 non-null float64
Speed               1048575 non-null float64
dtypes: float64(7), int64(3), object(4)
memory usage: 112.0+ MB
```

```
In [57]: train.describe()
```

Out[57]:

	vendor_id	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	trip_duration
count	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06
mean	1.534503e+00	1.664382e+00	-7.397342e+01	4.075094e+01	-7.397336e+01	4.075183e+01	9.621448e+02
std	4.988084e-01	1.314261e+00	4.280165e-02	3.381389e-02	4.274282e-02	3.645002e-02	5.853002e+03
min	1.000000e+00	0.000000e+00	-7.854740e+01	3.435970e+01	-7.981798e+01	3.218114e+01	1.000000e+00
25%	1.000000e+00	1.000000e+00	-7.399186e+01	4.073738e+01	-7.399131e+01	4.073594e+01	3.970000e+02
50%	2.000000e+00	1.000000e+00	-7.398174e+01	4.075415e+01	-7.397973e+01	4.075455e+01	6.620000e+02
75%	2.000000e+00	2.000000e+00	-7.396731e+01	4.076836e+01	-7.396301e+01	4.076984e+01	1.075000e+03
max	2.000000e+00	9.000000e+00	-6.133553e+01	5.188108e+01	-6.133553e+01	4.391176e+01	3.526282e+06

4.1 Data Cleaning

- Missing Values

Missing data is the nemesis of every data scientist, especially if they are new in the field. It is important to check whether we have data missing in the train and the test set. There are zero missing data in the dataset.

```
In [27]: train.isna().sum()

Out[27]: 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
         distance            0
         Time in hrs         0
         Speed               0
         dtype: int64
```

- Fixing Date Variables

Pickup and Dropoff date are two of the few variables that are provided. This information could play an important role in predicting the total duration time that is why it needs some attention. I will extract different time features such as hour, week, month and so on.

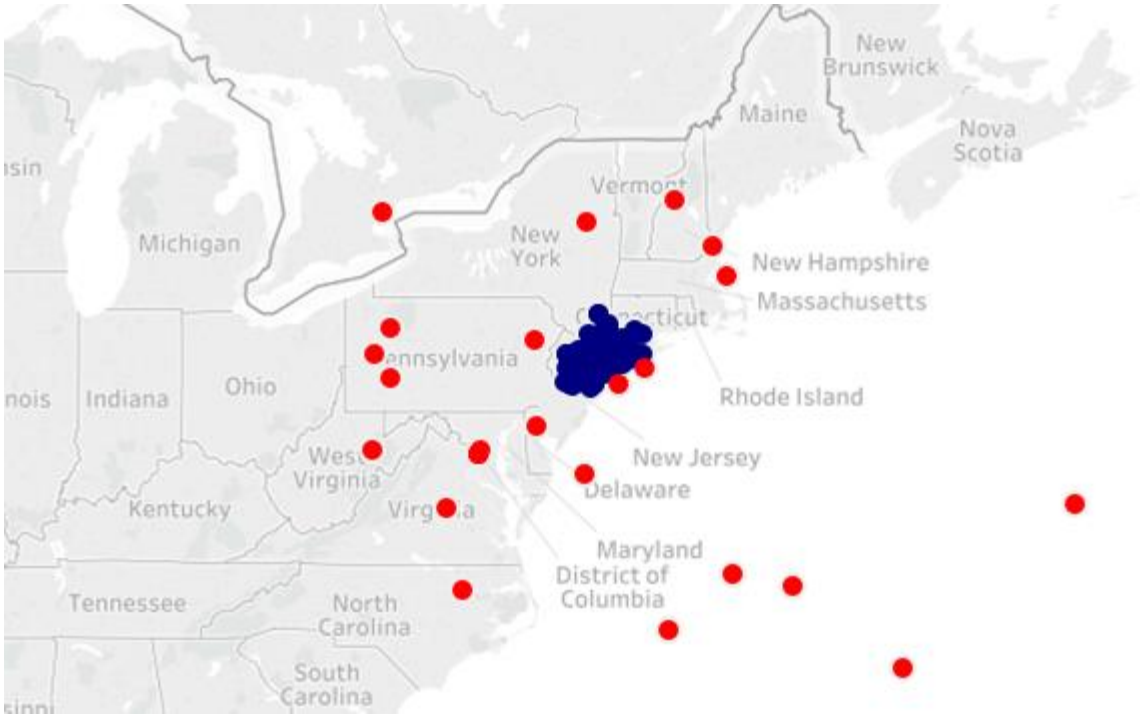
```
In [55]: train['pickup_datetime'] = pd.to_datetime(train.pickup_datetime)
         test['pickup_datetime'] = pd.to_datetime(test.pickup_datetime)
         train.loc[:, 'pickup_date'] = train['pickup_datetime'].dt.date
         test.loc[:, 'pickup_date'] = test['pickup_datetime'].dt.date
         train['dropoff_datetime'] = pd.to_datetime(train.dropoff_datetime)

In [56]: train['pickup_datetime'].head()

Out[56]: 0   2016-03-14 17:24:00
         1   2016-12-06 00:43:00
         2   2016-01-19 11:35:00
         3   2016-06-04 19:32:00
         4   2016-03-26 13:30:00
         Name: pickup_datetime, dtype: datetime64[ns]
```

- Latitude and Longitude Clean-up

Looking into it, the borders of NY City, in coordinates comes out to be:
city_long_border = (-74.03, -73.75)
city_lat_border = (40.63, 40.85)



From the above map and on comparing this to our `train.describe()` output we see that there are some coordinate points (pick ups/drop offs) that fall outside these borders. So, let's limit our area of investigation to within the NY City borders.

4.2 Principal Component Analysis

Since our dataset is big (1.6 million rows), we perform PCA. Principal Component Analysis (PCA) is a technique to transform data points into a new space. The new space typically has less dimensions than the previous one. Each dimension is a linear combination of the existing ones.

We use rapid miner's PCA feature to transform our data. Now we have only 3 reduced components of the transformed attributes. This helps us in building the model efficiently.

train (2)

Add new data sets on the left. Details for the selected data are shown below. You can change the data with the following actions.

TRANSFORM | CLEANSE | GENERATE | PIVOT | MERGE | MODEL | CHARTS | CREATE PROCESS | HISTORY

trip_duration Number	pc_1 Number	pc_2 Number	pc_3 Number
455	-2.892	-1.440	0.054
663	-4.853	-0.756	0.052
2124	-3.000	3.575	-0.069
429	-2.296	-1.567	0.052
435	-4.935	-1.370	0.067
443	-5.843	-1.261	-0.000
341	-0.816	-1.994	0.008
1551	-0.723	2.453	0.011
255	3.602	-2.863	0.010
1225	0.921	1.534	0.037
1274	-3.545	1.042	-0.004
1128	-2.287	0.785	-0.027
1114	-8.572	0.017	-0.037

443

1,048,575 rows - 4 columns (4 numerical)

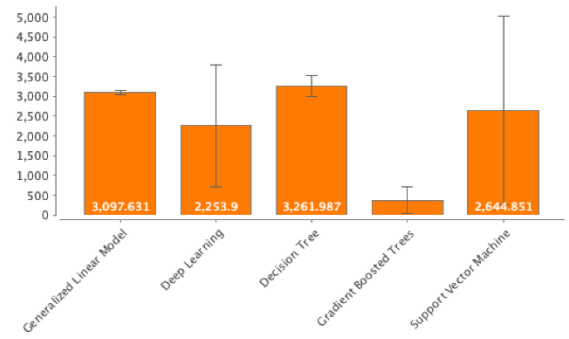
5. MODELLING

After the data transformation is done, it is passed into auto-model to see which model gets us the highest accuracy.

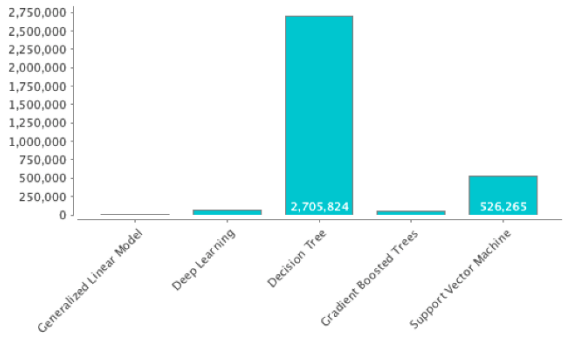
The output of auto-model is shown below,

Overview

Root Mean Squared Error



Runtime (ms)



Root Mean Squared Error

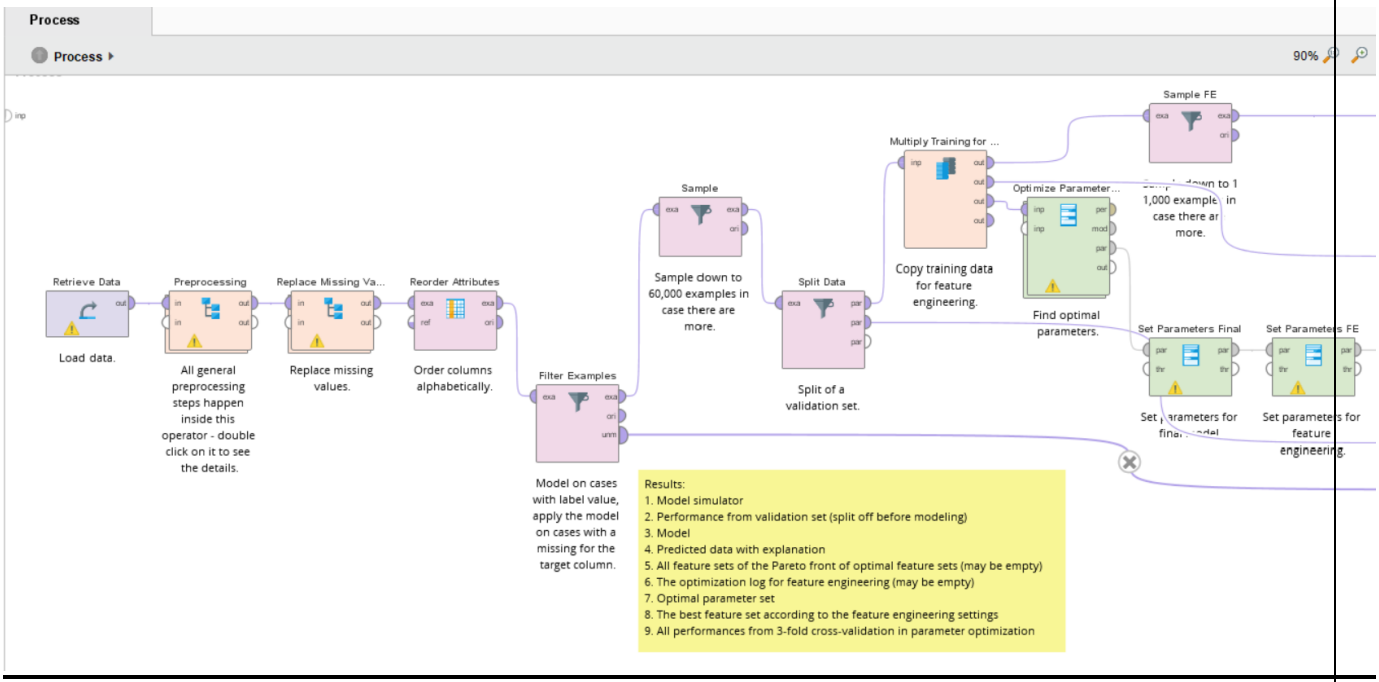
Model	Root Mean Squared Error	Standard Deviation	Runtime
Generalized Linear Model	3097.631	± 64.864	7 s
Deep Learning	2253.900	± 1543.319	1 min 8 s
Decision Tree	3261.987	± 263.466	45 min 5 s
Gradient Boosted Trees	374.889	± 341.692	47 s
Support Vector Machine	2644.851	± 2386.242	8 min 46 s

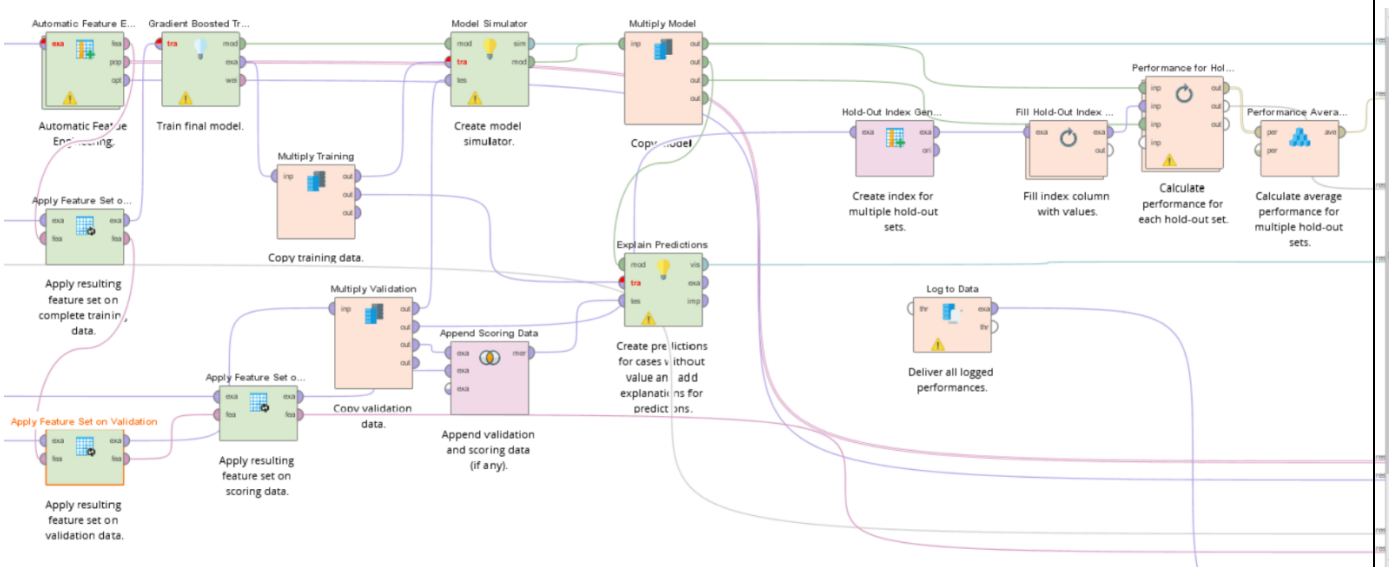
Clearly, we can see that Gradient Boosted Trees gives us best result when compared to other models. The Root Mean Squared Error (RMSE) of Gradient Boosted Trees is 374.8(seconds)

5.1 Gradient Boosted Tree Algorithm

Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. The objective of any supervised learning algorithm is to minimize the error, therefore lower the RMSE better the model.

5.2 Model Design Process





5.3 Model Results

Row No.	trip_duration	prediction(trip_duration)	pc_1	pc_2
1	211	213.400	-3.692	-2.150
2	2331	2268.536	2.113	6.222
3	625	617.833	-3.726	-0.874
4	704	700.771	-5.726	-0.649
5	750	752.298	1.388	-0.409
6	40	140.676	-14.476	-0.780
7	1050	1046.284	-6.906	0.069
8	1988	1968.202	1.000	4.506
9	501	484.154	-3.550	-1.259
10	669	662.417	16.942	-0.813
11	1238	1261.357	-6.467	0.522
12	248	264.140	-8.266	-1.492
13	1290	1285.687	5.785	2.560
14	1179	1207.462	-1.004	1.099
15	1128	1116.287	4.522	1.563
16	544	541.380	1.302	-1.306
17	1499	1543.253	-6.389	1.082
18	810	805.880	-9.885	-0.535

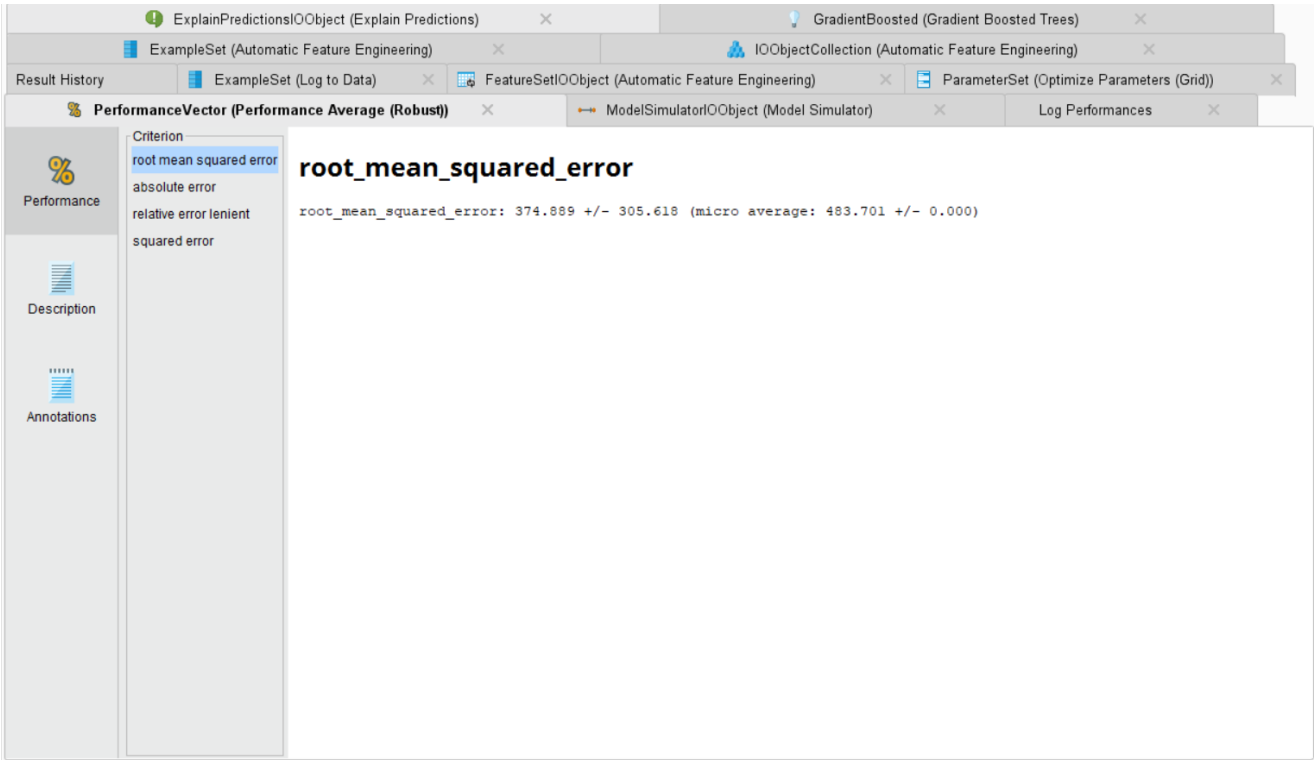
We can see sample of our model results. Our model gives out good predictions of taxi trip duration. We can see that our predicted values are almost close to the actual values.

6. EVALUATION

The evaluation metric which we are going to use for our predictive model is Root Mean Square Error (RMSE) value. RMSE is a quadratic scoring rule that also measures the average magnitude of the error. It's the square root of the average of squared differences between prediction and actual observation.

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

The image below shows us the Performance of our model based on RMSE score. The RMSE score of our Gradient Boosted Model is 374.889.



7. DEPLOYMENT

Our model can be converted to PMML standard for business purpose. PMML stands for "Predictive Model Markup Language". A PMML file may contain a myriad of data transformations (pre- and post-processing) as well as one or more predictive models. PMML is a standard for data mining models and is understood by many databases. This way it can be applied on a regular basis on huge amount of data.

7.1 Applications of the Model

The predictive model built can be used by ride hailing companies such as Uber, myTaxi, Ola, DIDI, etc., for predicting the taxi trip duration and improving the efficiency of Electronic Taxi Dispatching Systems which will decrease time loss and increase the profits.

8. CONCLUSION

- Using our Gradient Boosted Model, the efficiency of Electronic Taxi Dispatching Systems of the Ride Hailing businesses can improve.
- Taxi companies will be able to serve more customers with less taxies and drivers by minimizing taxies free time which will lead to efficiency and profits increment.

9. APPENDIX

Null Values

```
train.isna().sum()
```

Date Formatting:

```
train['pickup_datetime'] = pd.to_datetime(train.pickup_datetime)
test['pickup_datetime'] = pd.to_datetime(test.pickup_datetime)
train.loc[:, 'pickup_date'] = train['pickup_datetime'].dt.date
test.loc[:, 'pickup_date'] = test['pickup_datetime'].dt.date
train['dropoff_datetime'] = pd.to_datetime(train.dropoff_datetime)
```

Latitude and Longitude Clean-up:

```
train = train[train['pickup_longitude'] <= -73.75]
train = train[train['pickup_longitude'] >= -74.03]
train = train[train['pickup_latitude'] <= 40.85]
train = train[train['pickup_latitude'] >= 40.63]
train = train[train['dropoff_longitude'] <= -73.75]
train = train[train['dropoff_longitude'] >= -74.03]
train = train[train['dropoff_latitude'] <= 40.85]
train = train[train['dropoff_latitude'] >= 40.63]
```

Visualization:

Trip Duration

```
train['log_trip_duration'] = np.log(train['trip_duration'].values + 1)
plt.hist(train['log_trip_duration'].values, bins=100)
plt.xlabel('log(trip_duration)')
plt.ylabel('number of train records')
plt.show()
```

Normal Curve

```
plt.xlabel('log(trip_duration)')
plt.ylabel('number of train records')
```

```
    sns.distplot(train["log_trip_duration"], bins =100)
```

Correlation

```
corr = train.select_dtypes(include=[np.number]).corr()
mask = np.zeros_like(corr)
mask[np.triu_indices_from(mask)] = True
plt.figure(figsize=(10, 10))
with sns.axes_style('white'):
    sns.heatmap(corr, mask=mask, linewidths=0.01, square=True, linecolor='white')
plt.xticks(rotation=90)
plt.title('Correlation between features')
```