CAB FARE PREDICTION

PROJECT REPORT

RAVI SHARMA
DATA SCIENTIST
4 OCTOBER, 2020

INDEX

CAB FARE PREDICTION

Project Report

Background:

You are a cab rental start-up company. You have successfully run the pilot project and now want to launch your cab service across the country. You have collected the historical data from your pilot project and now have a requirement to apply analytics for fare prediction. You need to design a system that predicts the fare amount for a cab ride in the city.

Problem Statement:

In this challenge, we need to identify the fare amount for a particular cab ride in a city.

Problem Classification:

This problem belongs to the Regression Supervised Learning category.

Metrics for Evaluation:

This is a regression problem & to evaluate the model predictions the metrics to be used are Mean Squared Error (MSE) & Root Mean Squared Error (RMSE).

- **Mean Squared Error (MSE):**

  The mean squared error tells you how close a regression line is to a set of points. It does this by taking the distances from the points to the regression line (these distances are the “errors”) and squaring them. The squaring is necessary to remove any negative signs. The smaller the means squared error, the closer you are to finding the line of best fit.

  Calculated by:

$$MSE = \frac{\sum(Yo - Yp)^2}{n}$$

  Where,
  Yo is the observed values,
  Yp is the predicted values and
  n is the no of observations.
  The best value is 0.

 ROOT Mean Squared Error (RMSE):

Root Mean Square Error (RMSE) is the standard deviation of the residuals (prediction errors). Residuals are a measure of how far from the regression line data points are; RMSE is a measure of how spread out these residuals are. In other words, it tells you how concentrated the data is around the line of best fit.

Calculated by:

$$RMSE = \sqrt{\frac{\sum(Yo - Yp)^2}{n}}$$

Where,
Yo is the observed values,
Yp is the predicted values and
n is the no of observations.
The best value is 0.

 Mean Absolute Percentage Error (MAPE):

The mean absolute percentage error (MAPE) is a statistical measure of how accurate a prediction system is. It measures this accuracy as a percentage, and can be calculated as the average absolute percent error for each time period minus actual values divided by actual values. Where At is the actual value and Ft is the predicted value, this is given by:

Calculated by:

$$\mathrm{M} = \frac{1}{n}\sum_{t=1}^{n}\left|\frac{A_t - F_t}{A_t}\right|$$

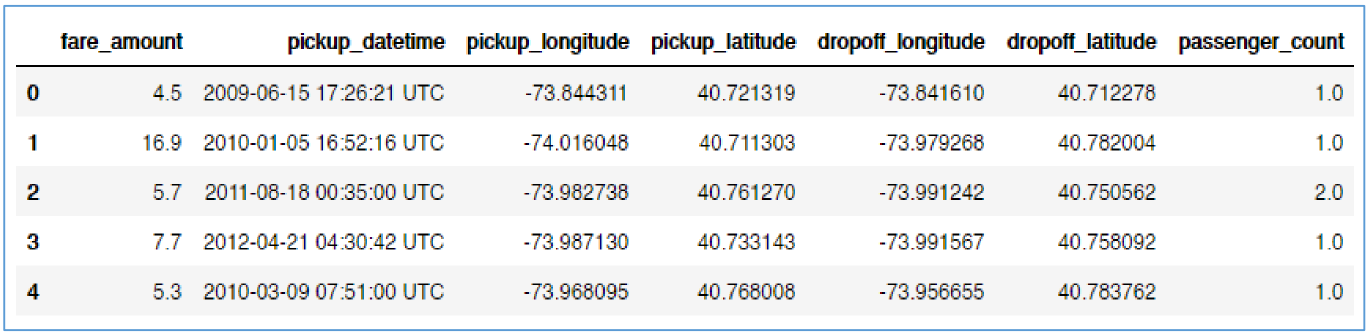
The mean absolute percentage error (MAPE) is the most common measure used to forecast error, and works best if there are no extremes to the data (and no zeros).

Introduction to Data:

- We have been provided with a train & test dataset (.csv format).

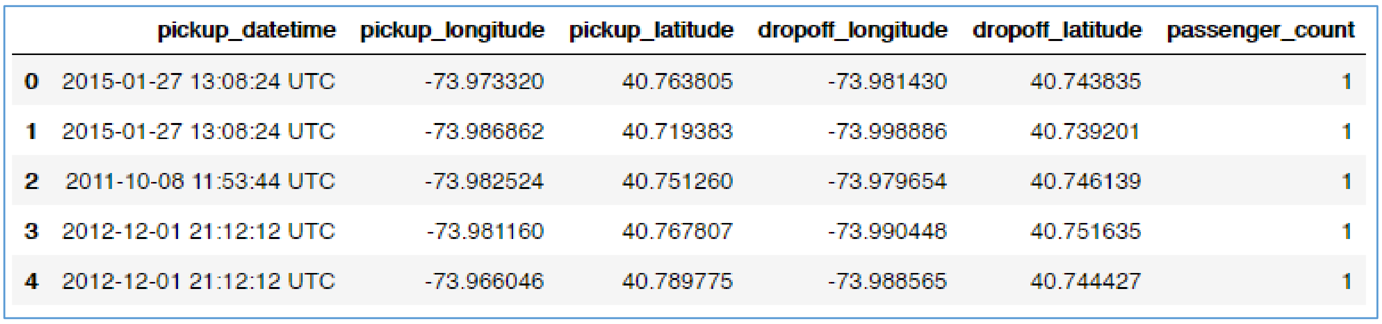
```
Shape of Train dataset is: (16067, 7)
Shape of Test dataset is: (9914, 6)
```

- Train dataset contains 16067 records & 7 variables (1 timestamp variable pickup_datetime, 1 numerical variable passanger_counts, 4 location based numerical variables pickup_latitude, pickup_longitude, dropoff_latitude, dropoff_longitude & 1 numerical target variable fare_amount).



	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	4.5	2009-06-15 17:26:21 UTC	-73.844311	40.721319	-73.841610	40.712278	1.0
1	16.9	2010-01-05 16:52:16 UTC	-74.016048	40.711303	-73.979268	40.782004	1.0
2	5.7	2011-08-18 00:35:00 UTC	-73.982738	40.761270	-73.991242	40.750562	2.0
3	7.7	2012-04-21 04:30:42 UTC	-73.987130	40.733143	-73.991567	40.758092	1.0
4	5.3	2010-03-09 07:51:00 UTC	-73.968095	40.768008	-73.956655	40.783762	1.0

- Test dataset contains 9914 records & 6 variables(1 timestamp variable pickup_datetime, 1 numerical variable passanger_counts & 4 location based numerical variables pickup_latitude, pickup_longitude, dropoff_latitude, dropoff_longitude).



	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	2015-01-27 13:08:24 UTC	-73.973320	40.763805	-73.981430	40.743835	1
1	2015-01-27 13:08:24 UTC	-73.986862	40.719383	-73.998886	40.739201	1
2	2011-10-08 11:53:44 UTC	-73.982524	40.751260	-73.979654	40.746139	1
3	2012-12-01 21:12:12 UTC	-73.981160	40.767807	-73.990448	40.751635	1
4	2012-12-01 21:12:12 UTC	-73.966046	40.789775	-73.988565	40.744427	1

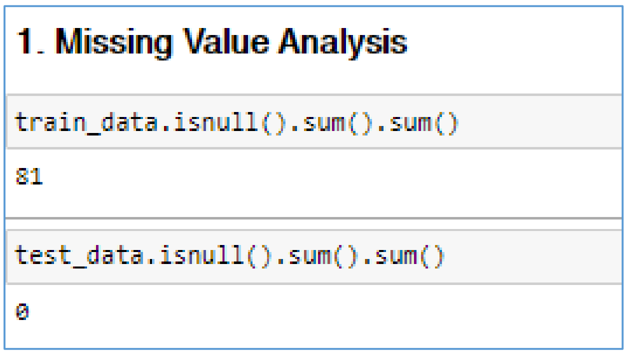
Data Pre Processing:

1. Missing Value Analysis:

Any blank observation in data is known as missing value. These missing values create issues in performing model training as model needs numerical value. To deal with this issue, we have to perform either deletion of observed row containing missing value or imputation at the place of missing value.

Imputation of missing value can done through several ways i.e. by Mean, Median, Mode, Min, Max, K-NN, or other domain specific knowledge based value.

Checking for missing values in the data.



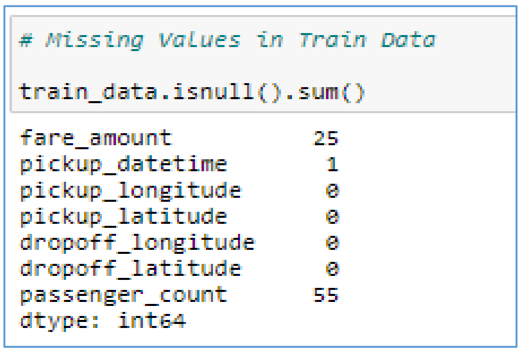
1. Missing Value Analysis

```
train_data.isnull().sum().sum()
```

```
81
```

```
test_data.isnull().sum().sum()
```

```
0
```

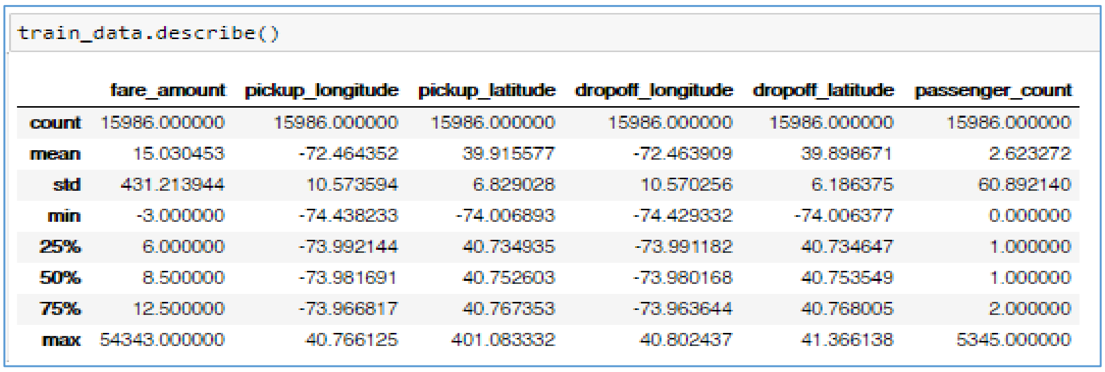


```
# Missing Values in Train Data

train_data.isnull().sum()
```

```
fare_amount          25
pickup_datetime       1
pickup_longitude      0
pickup_latitude       0
dropoff_longitude     0
dropoff_latitude      0
passenger_count      55
dtype: int64
```

Observations from above:

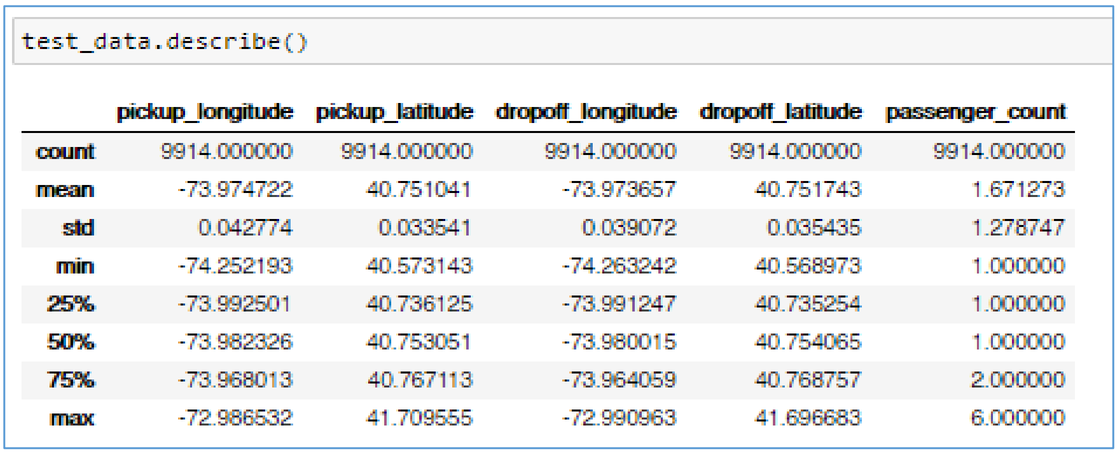
```
Train dataset contains total 81 missing values. 25 in fare_amount column, 1
in pickup_datetime column & 55 in passanger_count column, which is 0.5% of
total observations, thus dropping missing values.

Shape of train dataset after dropping the values is: (15986, 7)
```



```
train_data.describe()
```

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
count	15986.000000	15986.000000	15986.000000	15986.000000	15986.000000	15986.000000
mean	15.030453	-72.464352	39.915577	-72.463909	39.898671	2.623272
std	431.213944	10.573594	6.829028	10.570256	6.186375	60.892140
min	-3.000000	-74.438233	-74.006893	-74.429332	-74.006377	0.000000
25%	6.000000	-73.992144	40.734935	-73.991182	40.734647	1.000000
50%	8.500000	-73.981691	40.752603	-73.980168	40.753549	1.000000
75%	12.500000	-73.966817	40.767353	-73.963644	40.768005	2.000000
max	54343.000000	40.766125	401.083332	40.802437	41.366138	5345.000000

2. Checking training data for impurities:



```
test_data.describe()
```

	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
count	9914.000000	9914.000000	9914.000000	9914.000000	9914.000000
mean	-73.974722	40.751041	-73.973657	40.751743	1.671273
std	0.042774	0.033541	0.039072	0.035435	1.278747
min	-74.252193	40.573143	-74.263242	40.568973	1.000000
25%	-73.992501	40.736125	-73.991247	40.735254	1.000000
50%	-73.982326	40.753051	-73.980015	40.754065	1.000000
75%	-73.968013	40.767113	-73.964059	40.768757	2.000000
max	-72.986532	41.709555	-72.990963	41.696683	6.000000

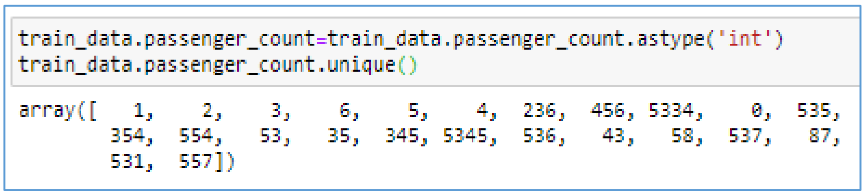
Observations from above:

1. This data is for United States as lat 40.xxx & long -73.xxx locate to New York United States. So, latitude range = 40.xxx to 42.xxx longitude range = -72.xxx to -74.xxx.
2. Impure data in pickup & dropoff longitude as it contains values outside their range i.e. 40.xxx.
3. Impure data in pickup & dropoff latitude as it contains values outside their range i.e. -74.xxx.
4. Fare Amount contains some very high values as Standard deviation of $ 430 is quite high.

3. Treating training data for impurities:

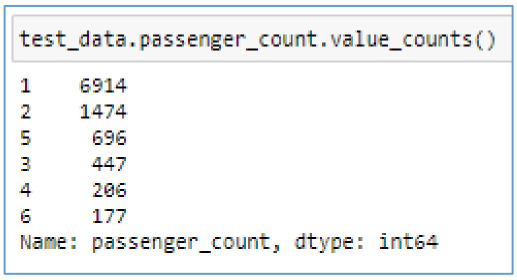
A. Treating Passenger Count

Unique values in passenger_count column in train & test dataset.



```
train_data.passenger_count=train_data.passenger_count.astype('int')
train_data.passenger_count.unique()
```

```
array([   1,    2,    3,    6,    5,    4,  236,  456, 5334,    0,  535,
        354,  554,   53,   35,  345, 5345,  536,   43,   58,  537,   87,
        531,  557])
```



```
test_data.passenger_count.value_counts()
```

```
1    6914
2    1474
5     696
3     447
4     206
6     177
Name: passenger_count, dtype: int64
```

Observations from above:

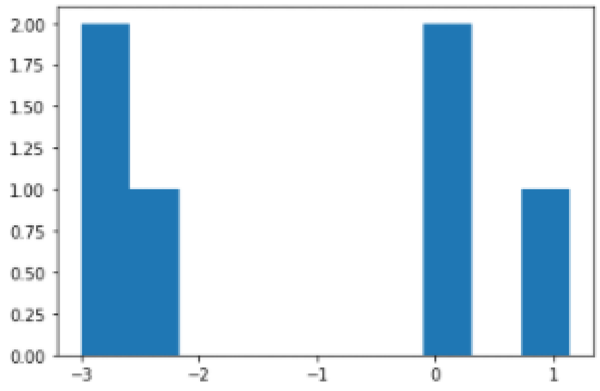
1. As Test data contains the values in range [1,2,3,4,5,6]. Removing all the other values from dataset.
2. Shape of train dataset after dropping the values is: (15909, 7).

B. Treating Fare Amount

As this is the cab fare amount, there should be a minimum fare amount which should be charge for a ride. Let’s take an average minimum amount to be charged as $2.5.
Values in fare_amount column less than $2.5.

```
# Taking he minimum value of fare_amount as $2.5

plt.hist(train_data.fare_amount[train_data.fare_amount<2.5])
```

```
(array([2., 1., 0., 0., 0., 0., 0., 2., 0., 1.]),
 array([-3.   , -2.586, -2.172, -1.758, -1.344, -0.93 , -0.516, -0.102,
         0.312,  0.726,  1.14 ]),
 <a list of 10 Patch objects>)
```

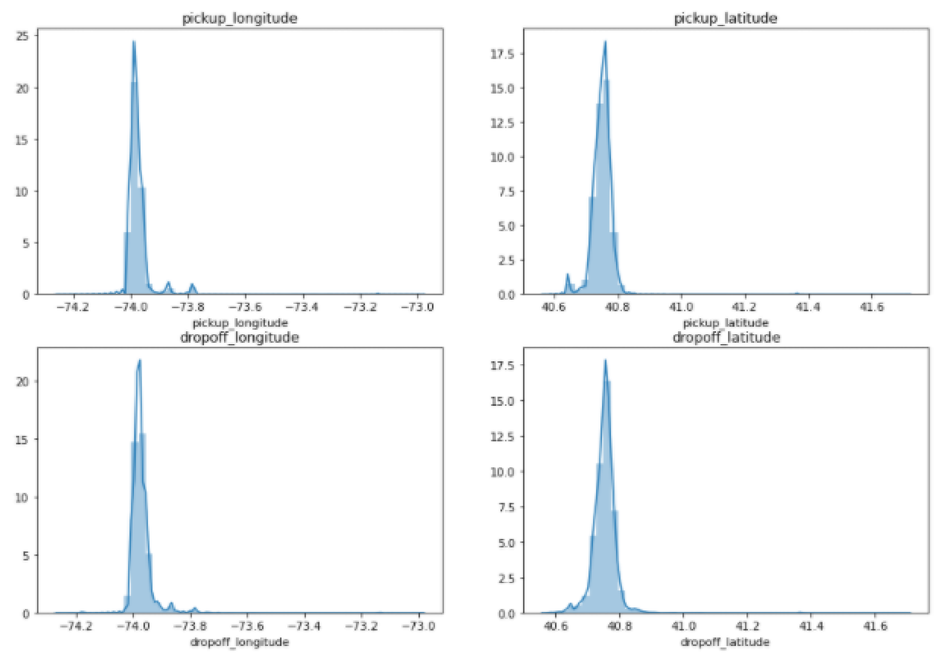


```
Observations from above:
  1. As the above plot shows, there are few values which lies beyond the
     minimum fare value. So, removing all these values from dataset.
  2. Shape of train dataset after dropping the values is: (15903, 7)
```

C. Treating Pickup & dropoff (Lat Long) Values

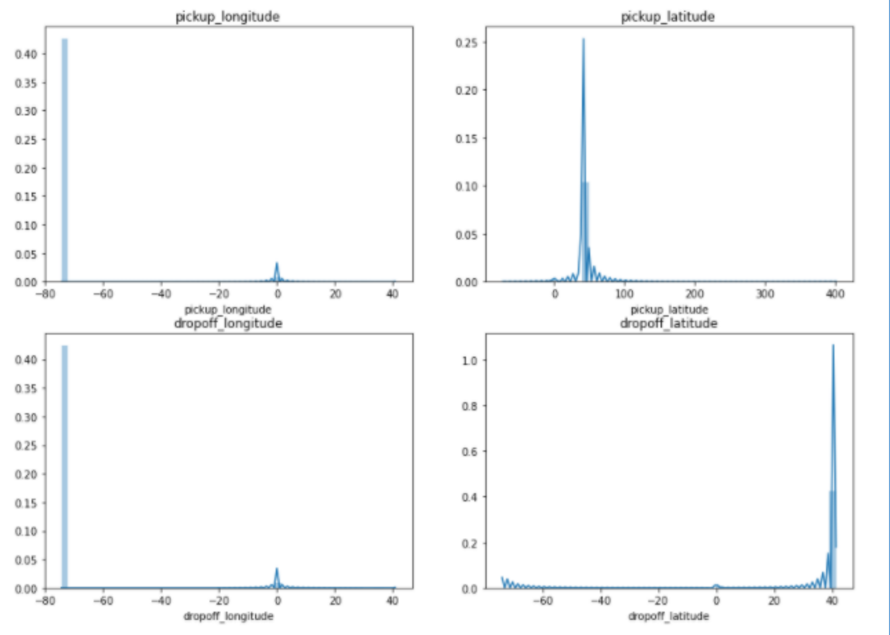
As pickup & dropoff longitude & latitude contains values outside their range. Let’s check test data for location values.

```
plt.figure(figsize=(14,10))
for i,col in enumerate(['pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitude']):
    plt.subplot(2,2,i+1)
    sns.distplot(test_data[col])
    plt.title(col)
plt.show()
```



Let’s check train data for location values.

```
plt.figure(figsize=(14,10))
for i,col in enumerate(['pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitude']):
    plt.subplot(2,2,i+1)
    sns.distplot(train_data[col])
    plt.title(col)
plt.show()
```



Observations from above:

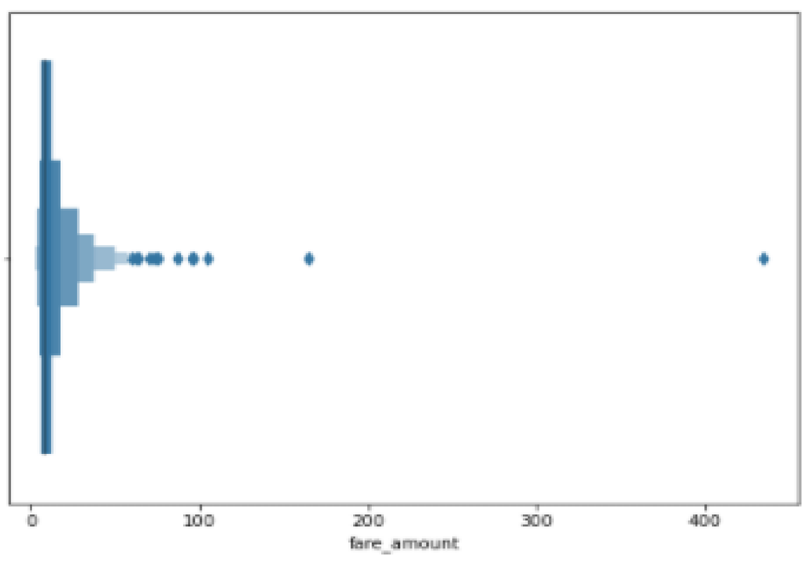
1. As seen from above plots, the test contains latitude values in range 40 to 42 & longitude values in range -74 to -72. While in train data latitude ranges in -74 to 40 & longitude ranges in -74 to 40.
2. Thus removing all the values from the train data which lies outside the range i.e. for, Latitude -> 40 to 42 & Longitude -> -74 to -72.
3. Shape of train dataset after dropping the values is: (12034, 7).

4. Outlier Analysis on Fare amount:

As from data description we know that fare_amount column contains some very high values as it’s standard deviation is quite high i.e. 431. Let’s have a look at the boxplot of fare_amount,

```
plt.figure(figsize=(8,6))
sns.boxenplot(train_data.fare_amount)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x21766c7fa88>
```



```
Observations from above:

  1. As seen from above plot, the fare_amount contains some very high values.
  2. Thus performing outlier analysis on the fare_amount column. The min outboun
     d & max outbound limits are,

     Max out bound:  21.45 Min out bound:  -3.75.

  3. Removing all the values from the train data which lies outside the range.
  4. Shape of train dataset after dropping the values is: (10904, 7).
```

Exploratory Data Analysis & Visualizations:

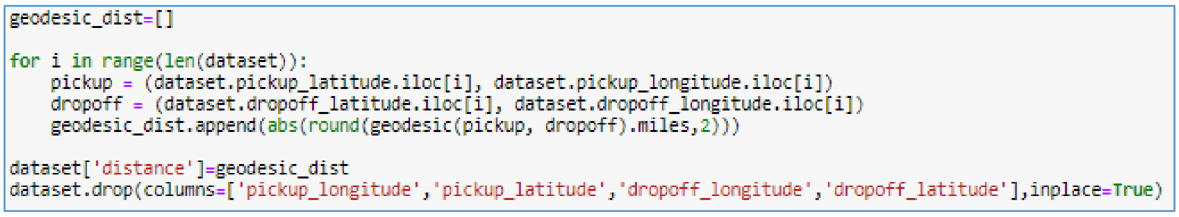
1. Feature Engineering:

Feature engineering is the process of transforming raw data into features that better represent the underlying problem to the predictive models, resulting in improved model accuracy on unseen data.

Here we are creating some new features from the raw data.

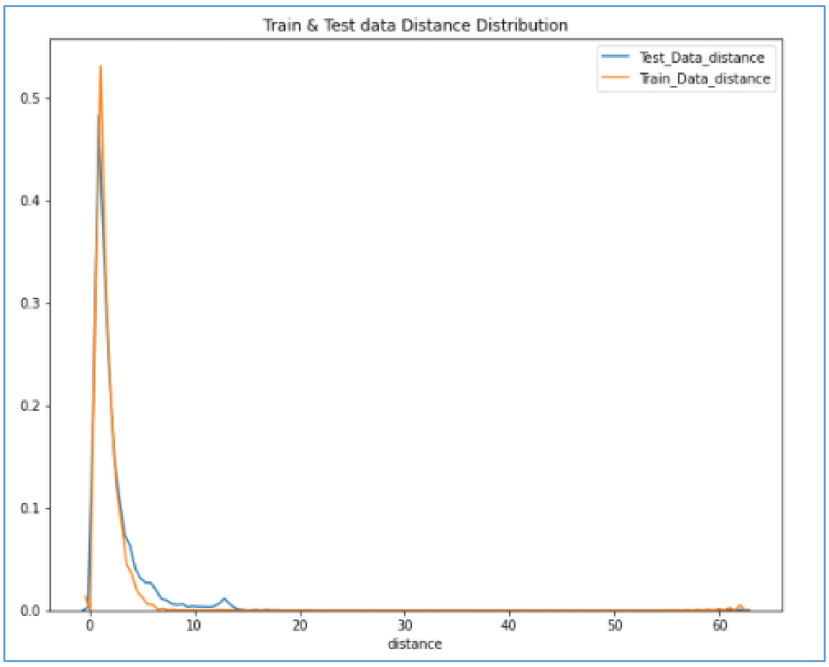
A. Creating distance feature:

Here we are creating a distance feature based on pickup & dropoff location's latitude & longitude coordinates with units in miles, using geosedic function of geopy.distance module. After creating distance variable, we are dropping location coordinates variables.



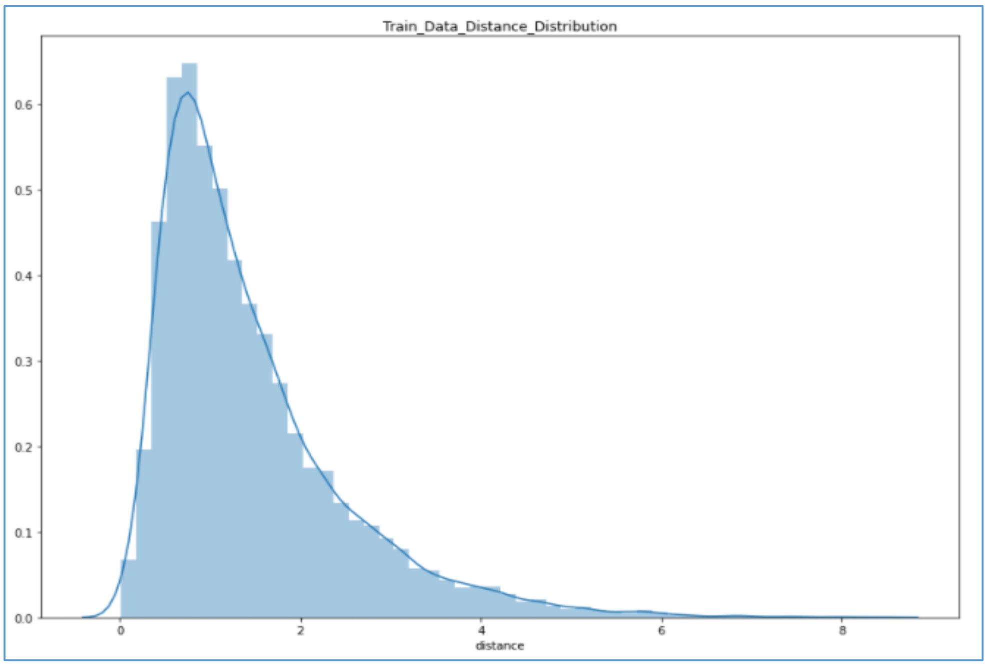
```
geodesic_dist=[]

for i in range(len(dataset)):
    pickup = (dataset.pickup_latitude.iloc[i], dataset.pickup_longitude.iloc[i])
    dropoff = (dataset.dropoff_latitude.iloc[i], dataset.dropoff_longitude.iloc[i])
    geodesic_dist.append(abs(round(geodesic(pickup, dropoff).miles,2)))

dataset['distance']=geodesic_dist
dataset.drop(columns=['pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitude'],inplace=True)
```

The distribution of distance feature in train & test dataset is:



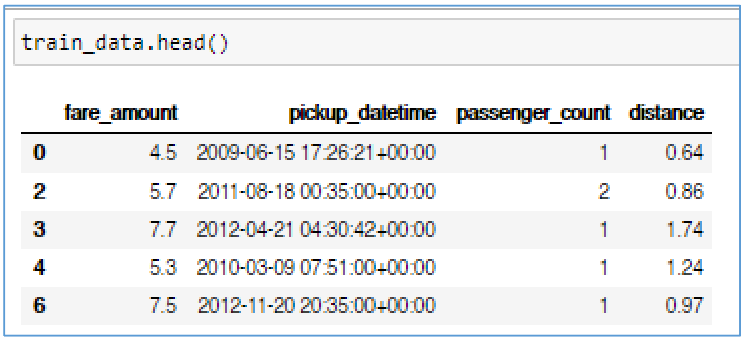
From above plot it's clearly shown that most rides lie between 0 & 10 miles. Thus, removing values above 10 miles & values with exact zero miles as distance travelled cannot be zero.

Train data distance distribution after removing the values:



```
Shape of train dataset after dropping the values is: (10738, 7).
```

```
Train Data head after creating the distance feature:
```

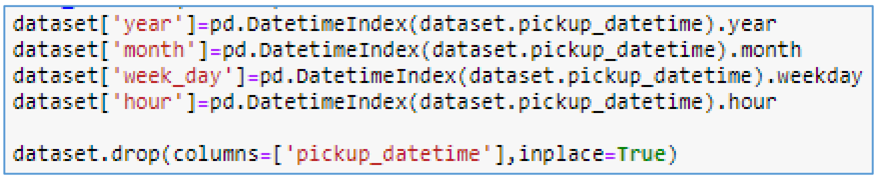


```
train_data.head()
```

	fare_amount	pickup_datetime	passenger_count	distance
0	4.5	2009-06-15 17:26:21+00:00	1	0.64
2	5.7	2011-08-18 00:35:00+00:00	2	0.86
3	7.7	2012-04-21 04:30:42+00:00	1	1.74
4	5.3	2010-03-09 07:51:00+00:00	1	1.24
6	7.5	2012-11-20 20:35:00+00:00	1	0.97

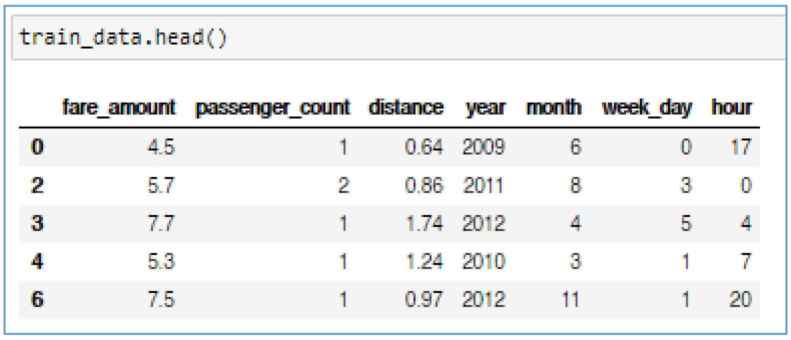
B. Creating Timestamp based feature

Here we are creating year, month, week_day, hour features based on pickup_datetime variable. After creating the features, we are dropping the datetime variable.



```
dataset['year']=pd.DatetimeIndex(dataset.pickup_datetime).year
dataset['month']=pd.DatetimeIndex(dataset.pickup_datetime).month
dataset['week_day']=pd.DatetimeIndex(dataset.pickup_datetime).weekday
dataset['hour']=pd.DatetimeIndex(dataset.pickup_datetime).hour

dataset.drop(columns=['pickup_datetime'],inplace=True)
```

```
Train Data head after creating the timestamp features:
```

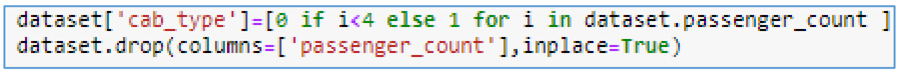


```
train_data.head()
```

	fare_amount	passenger_count	distance	year	month	week_day	hour
0	4.5	1	0.64	2009	6	0	17
2	5.7	2	0.86	2011	8	3	0
3	7.7	1	1.74	2012	4	5	4
4	5.3	1	1.24	2010	3	1	7
6	7.5	1	0.97	2012	11	1	20

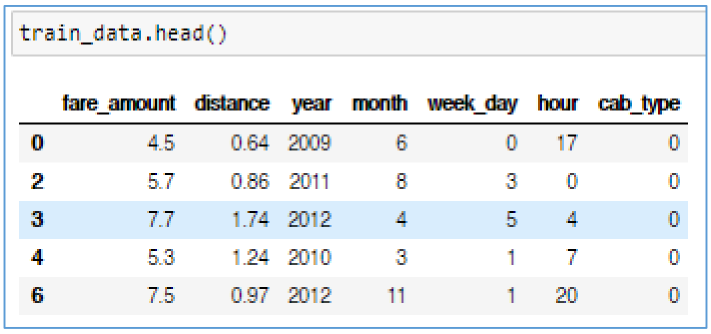
C. Creating Passenger Count based feature

Here we are creating **cab_type** feature based on passenger_count variable i.e. if the passenger count is upto 3 we are assuming it as a small cab, while if passenger count is greater than 3 we are assuming it to be a bigger size cab. Small cab =0 & Bigger cab =1. . After creating the feature, we are dropping the passenger_count variable.



```
dataset['cab_type']=[0 if i<4 else 1 for i in dataset.passenger_count ]
dataset.drop(columns=['passenger_count'],inplace=True)
```

Train Data head after creating the cab_type feature:

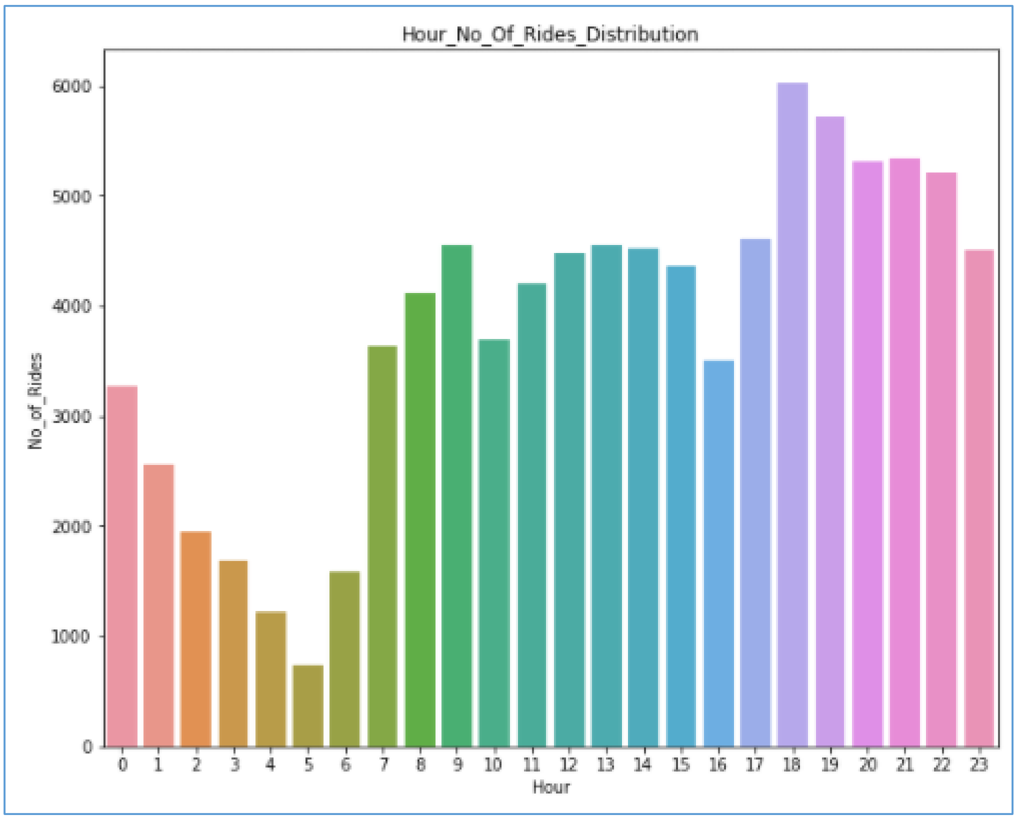


```
train_data.head()
```

	fare_amount	distance	year	month	week_day	hour	cab_type
0	4.5	0.64	2009	6	0	17	0
2	5.7	0.86	2011	8	3	0	0
3	7.7	1.74	2012	4	5	4	0
4	5.3	1.24	2010	3	1	7	0
6	7.5	0.97	2012	11	1	20	0

2. Visualizing the effects of feature's on target variable:

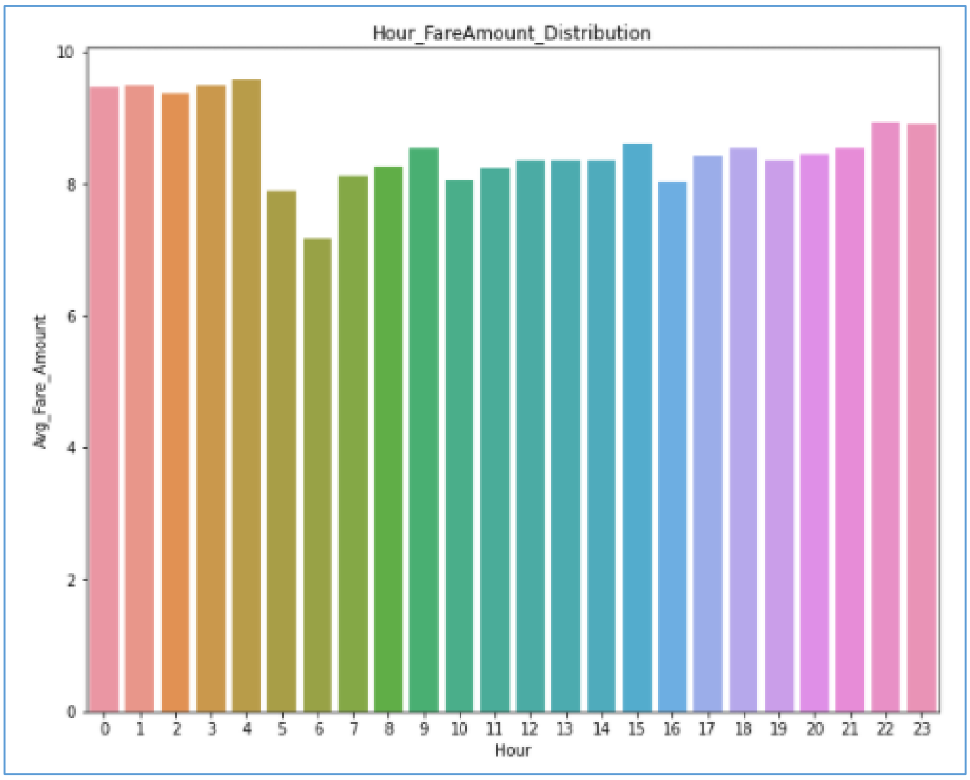
A. Hour effect on No of Rides



Observations from above:

1. 6.00 PM – 11.00 PM hours shows higher number of cab rides.
2. 1.00 AM – 6.00 AM hours shows lower number of cab rides.

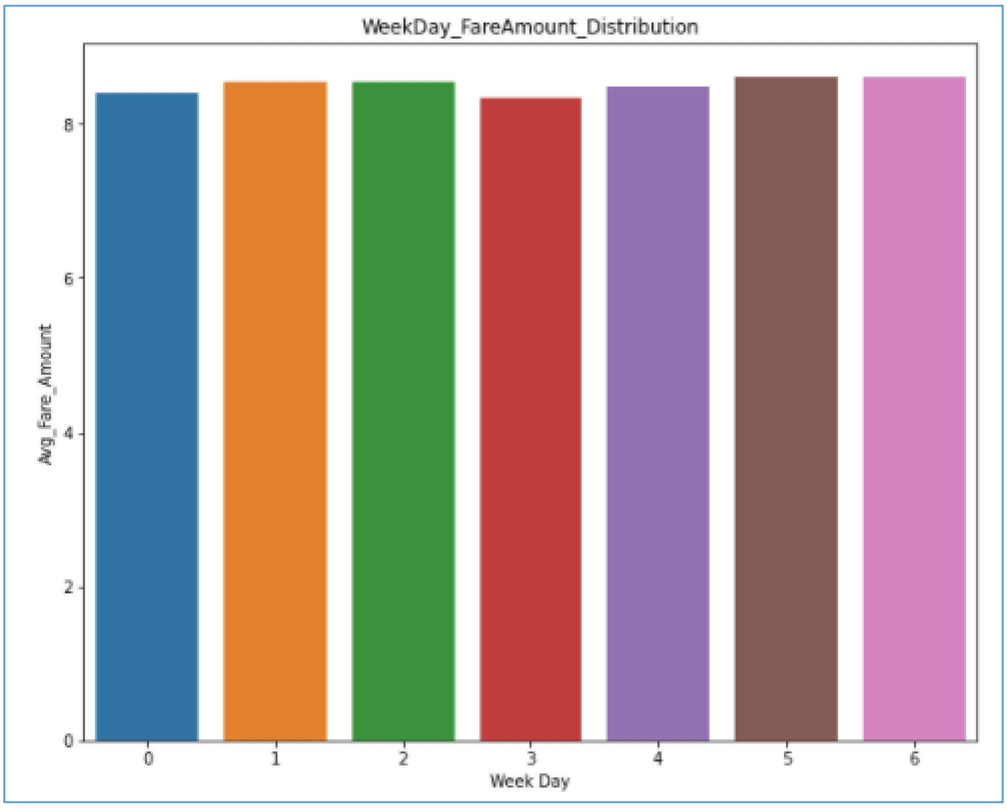
B. Hour effect on Fare Amount



Observations from above:

Late night hours 10.00 PM – 4.00 AM shows higher fare rates.

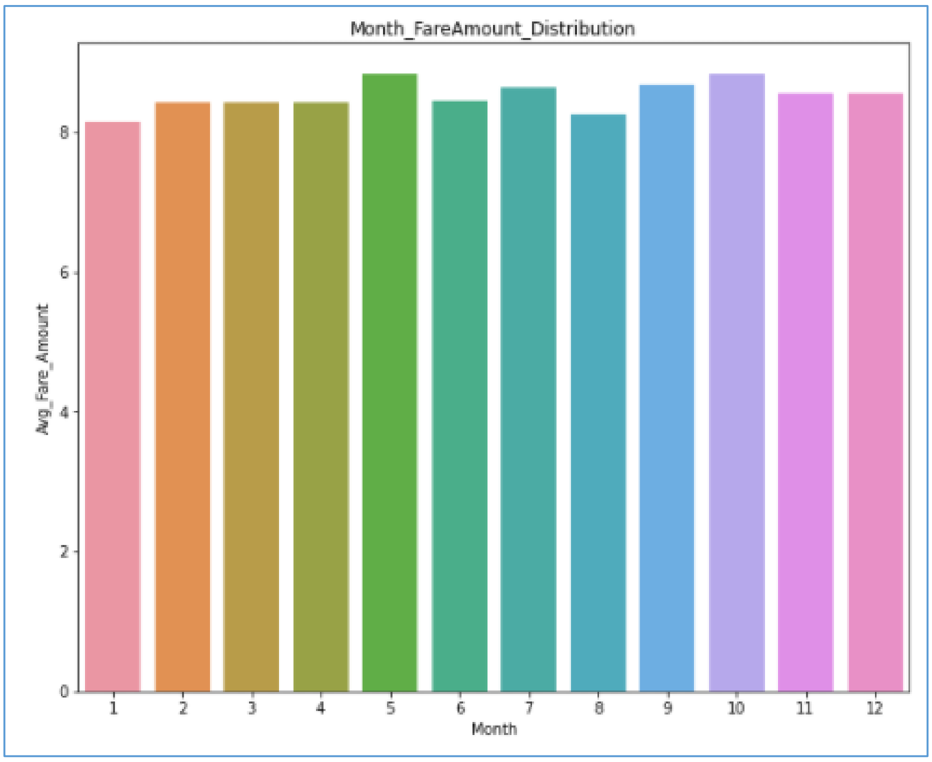
C. Week Day effect on Fare amount



Observations from above:

Weekday shows no major impact on fare_amount.

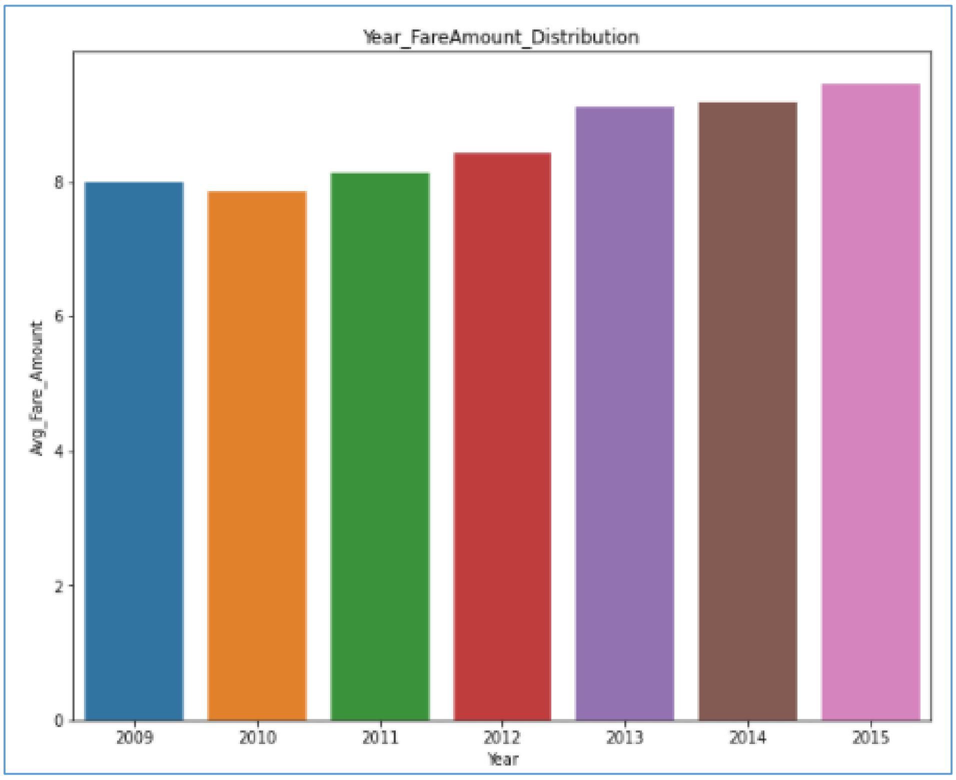
D. Month effect on Fare amount



Observations from above:

Month shows no major impact on fare_amount.

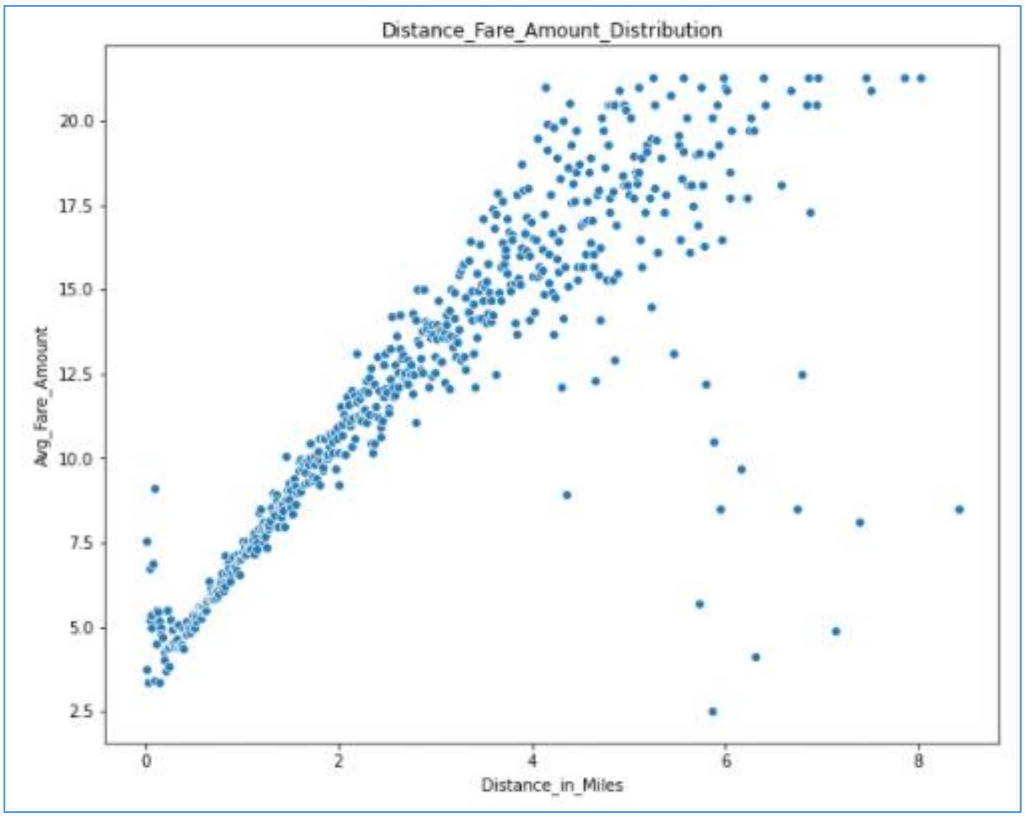
E. Year effect on Fare amount



Observations from above:

Year shows a slight increasing effect on Fare Amount.

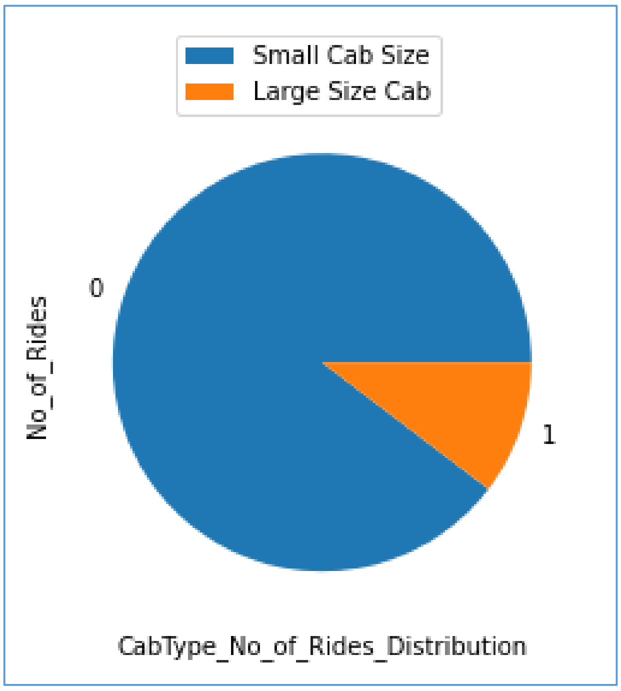
F. Distance effect on Fare amount



Observations from above:

Distance shows a highly linear increasing effect on Fare Amount.

G. Cab Type effect on Number of Rides

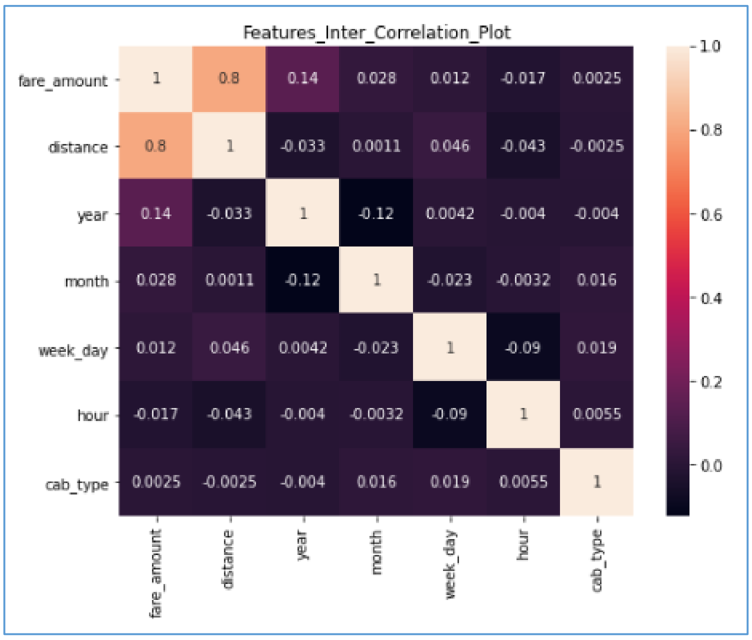


Observations from above:

Most of the cabs rides are of small size cab.

3. Feature Selection:

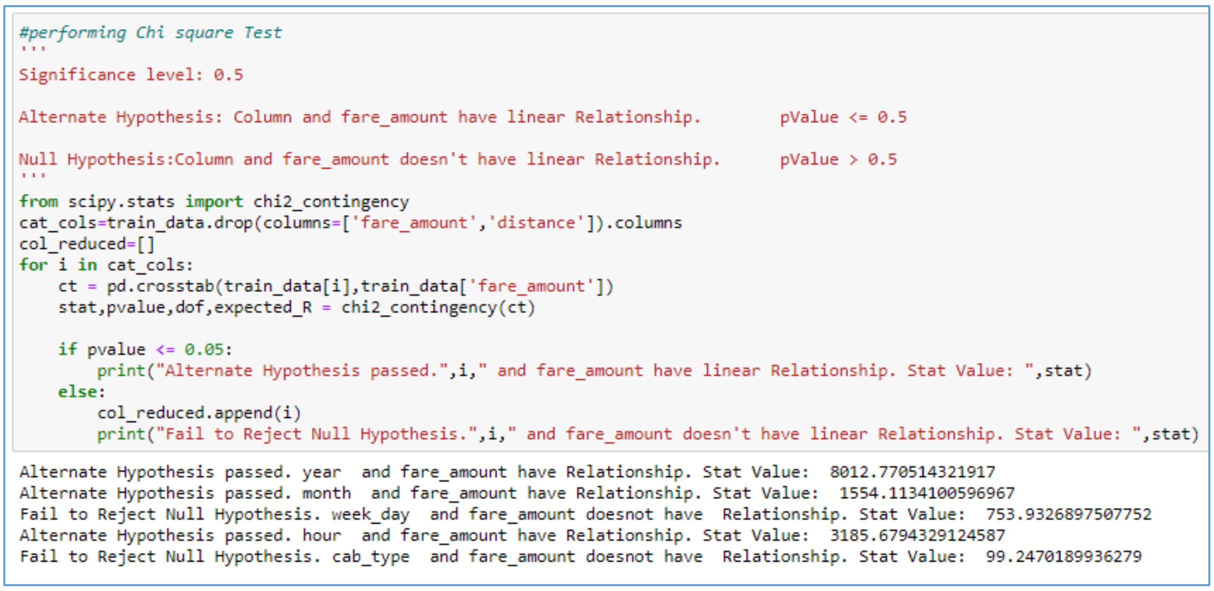
A. Features Multi Collinearity Test



Observations from above:

1. The correlation values between feature variables is less than 0.15. Thus, multicollinearity doesn’t exists between the feature variables.
2. The Distance column have a high coorelation value with the target variable

B. Categorical Features – Target Chi Square Test



```
#performing Chi square Test
'''
Significance level: 0.5

Alternate Hypothesis: Column and fare_amount have linear Relationship.        pValue <= 0.5

Null Hypothesis:Column and fare_amount doesn't have linear Relationship.      pValue > 0.5
'''
from scipy.stats import chi2_contingency
cat_cols=train_data.drop(columns=['fare_amount','distance']).columns
col_reduced=[]
for i in cat_cols:
    ct = pd.crosstab(train_data[i],train_data['fare_amount'])
    stat,pvalue,dof,expected_R = chi2_contingency(ct)

    if pvalue <= 0.05:
        print("Alternate Hypothesis passed.",i," and fare_amount have linear Relationship. Stat Value: ",stat)
    else:
        col_reduced.append(i)
        print("Fail to Reject Null Hypothesis.",i," and fare_amount doesn't have linear Relationship. Stat Value: ",stat)
```

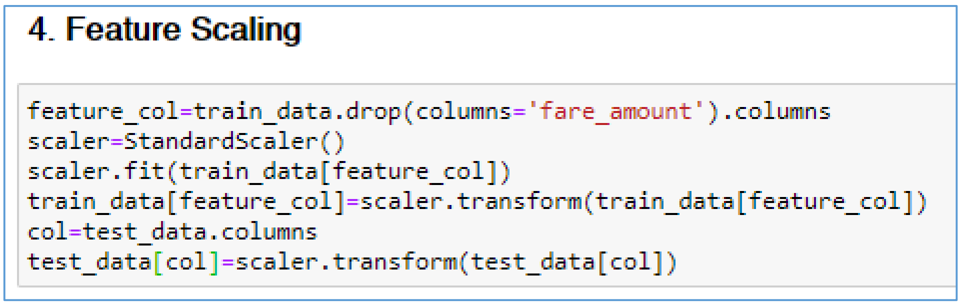
```
Alternate Hypothesis passed. year  and fare_amount have Relationship. Stat Value:  8012.770514321917
Alternate Hypothesis passed. month  and fare_amount have Relationship. Stat Value:  1554.1134100596967
Fail to Reject Null Hypothesis. week_day  and fare_amount doesnot have  Relationship. Stat Value:  753.9326897507752
Alternate Hypothesis passed. hour  and fare_amount have Relationship. Stat Value:  3185.6794329124587
Fail to Reject Null Hypothesis. cab_type  and fare_amount doesnot have  Relationship. Stat Value:  99.2470189936279
```

Observations from above:

We Fail to reject null hypothesis for week_day & cab_type columns. But we are not going to drop any columns as cab_type is the only column to provide a relationship with passenger count & week_day have a slight impact on number of rides.

4. Feature Scaling:

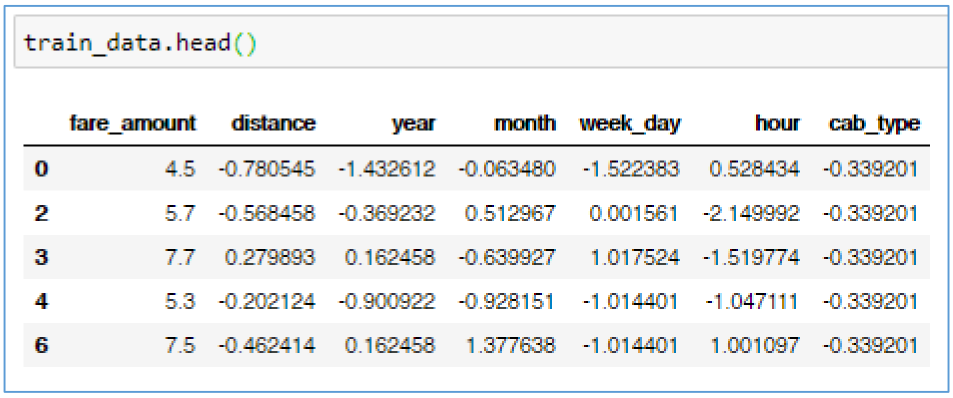
As the numerical feature almost follows normalized distribution, performing Standardization on the dataset to scale down the features values in an appropriate range which is centred to zero to lower down the variance of features to same scale. This is done to equalize the magnitude of features values so that no feature will dominate the calculations during modelling like Logistic Regression etc. We are using StandardScaler from scikit learn module to standardized the data.



4. Feature Scaling

```
feature_col=train_data.drop(columns='fare_amount').columns
scaler=StandardScaler()
scaler.fit(train_data[feature_col])
train_data[feature_col]=scaler.transform(train_data[feature_col])
col=test_data.columns
test_data[col]=scaler.transform(test_data[col])
```

Data after Standardization

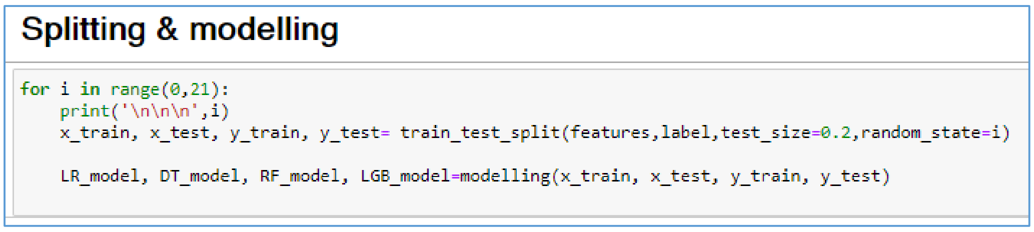


```
train_data.head()
```

	fare_amount	distance	year	month	week_day	hour	cab_type
0	4.5	-0.780545	-1.432612	-0.063480	-1.522383	0.528434	-0.339201
2	5.7	-0.568458	-0.369232	0.512967	0.001561	-2.149992	-0.339201
3	7.7	0.279893	0.162458	-0.639927	1.017524	-1.519774	-0.339201
4	5.3	-0.202124	-0.900922	-0.928151	-1.014401	-1.047111	-0.339201
6	7.5	-0.462414	0.162458	1.377638	-1.014401	1.001097	-0.339201

Splitting & Modelling:

Using train_test_split from scikit learn module, we split the data into 80-20 ratio & modelling on train & validation data using 4 different algorithms i.e. Linear Regression, Decision Tree Regressor, Random Forest Regressor & LightGBM Regressor.
Here we are trying to hypertune the best random state for train, validation data split.



Splitting & modelling

```
for i in range(0,21):
    print('\n\n\n',i)
    x_train, x_test, y_train, y_test= train_test_split(features,label,test_size=0.2,random_state=i)

    LR_model, DT_model, RF_model, LGB_model=modelling(x_train, x_test, y_train, y_test)
```

The best random state is 12, the modelling results are:

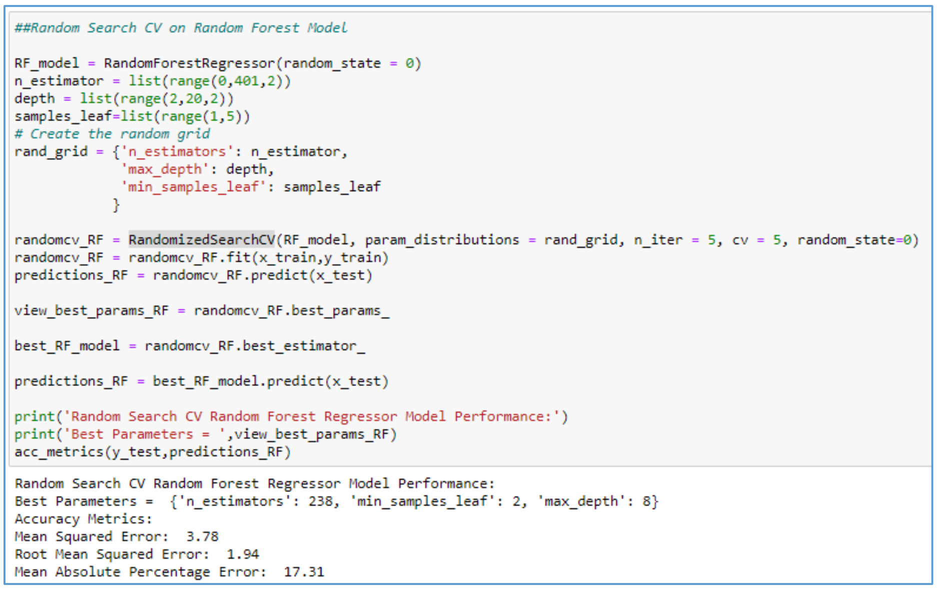
Metrics	Linear Regression	Decision Tree	Random Forest	LightGBM
MSE	4.43	4.19	4.06	3.59
RMSE	2.1	2.05	2.01	1.89
MAPE	19.08	18.21	18.5	16.94

From above the two best performing model is Random Forest & LightGBM Regressor.

Hyper Parameters Tuning & Best Model

Using **RandomSearchCV** from **sklearn.model_selection,** we are trying to optimise the hyper parameters of Random Forest & LightGBM algorithms & develop the best model.

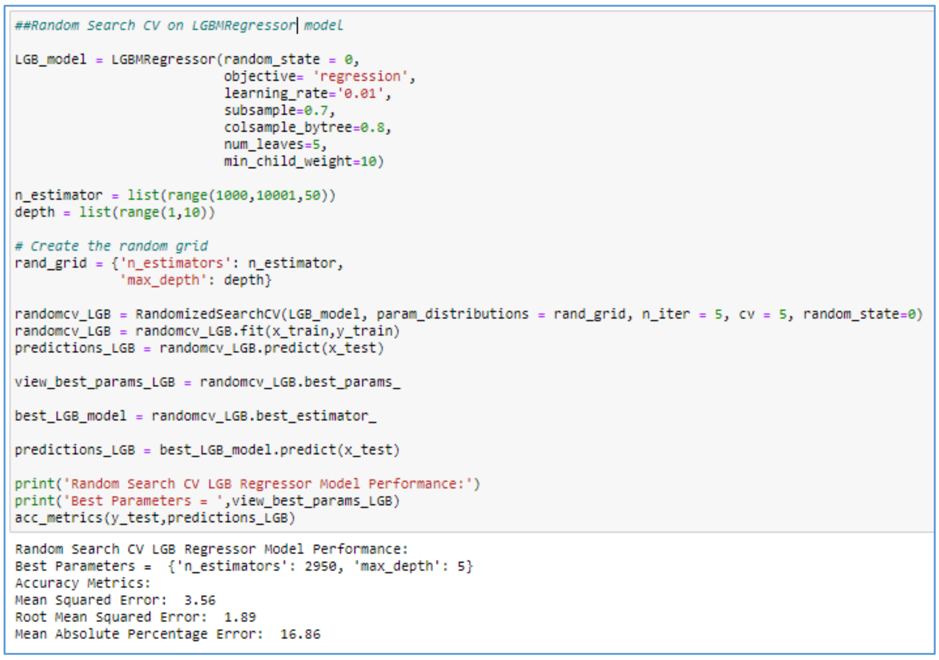
Random Forest Hyper parameters tuning:



```
##Random Search CV on Random Forest Model

RF_model = RandomForestRegressor(random_state = 0)
n_estimator = list(range(0,401,2))
depth = list(range(2,20,2))
samples_leaf=list(range(1,5))
# Create the random grid
rand_grid = {'n_estimators': n_estimator,
             'max_depth': depth,
             'min_samples_leaf': samples_leaf
            }

randomcv_RF = RandomizedSearchCV(RF_model, param_distributions = rand_grid, n_iter = 5, cv = 5, random_state=0)
randomcv_RF = randomcv_RF.fit(x_train,y_train)
predictions_RF = randomcv_RF.predict(x_test)

view_best_params_RF = randomcv_RF.best_params_

best_RF_model = randomcv_RF.best_estimator_

predictions_RF = best_RF_model.predict(x_test)

print('Random Search CV Random Forest Regressor Model Performance:')
print('Best Parameters = ',view_best_params_RF)
acc_metrics(y_test,predictions_RF)
```

```
Random Search CV Random Forest Regressor Model Performance:
Best Parameters =  {'n_estimators': 238, 'min_samples_leaf': 2, 'max_depth': 8}
Accuracy Metrics:
Mean Squared Error:  3.78
Root Mean Squared Error:  1.94
Mean Absolute Percentage Error:  17.31
```

LightGBM Regressor Hyper parameters tuning:



```
##Random Search CV on LGBMRegressor model

LGB_model = LGBMRegressor(random_state = 0,
                          objective= 'regression',
                          learning_rate='0.01',
                          subsample=0.7,
                          colsample_bytree=0.8,
                          num_leaves=5,
                          min_child_weight=10)

n_estimator = list(range(1000,10001,50))
depth = list(range(1,10))

# Create the random grid
rand_grid = {'n_estimators': n_estimator,
             'max_depth': depth}

randomcv_LGB = RandomizedSearchCV(LGB_model, param_distributions = rand_grid, n_iter = 5, cv = 5, random_state=0)
randomcv_LGB = randomcv_LGB.fit(x_train,y_train)
predictions_LGB = randomcv_LGB.predict(x_test)

view_best_params_LGB = randomcv_LGB.best_params_

best_LGB_model = randomcv_LGB.best_estimator_

predictions_LGB = best_LGB_model.predict(x_test)

print('Random Search CV LGB Regressor Model Performance:')
print('Best Parameters = ',view_best_params_LGB)
acc_metrics(y_test,predictions_LGB)
```

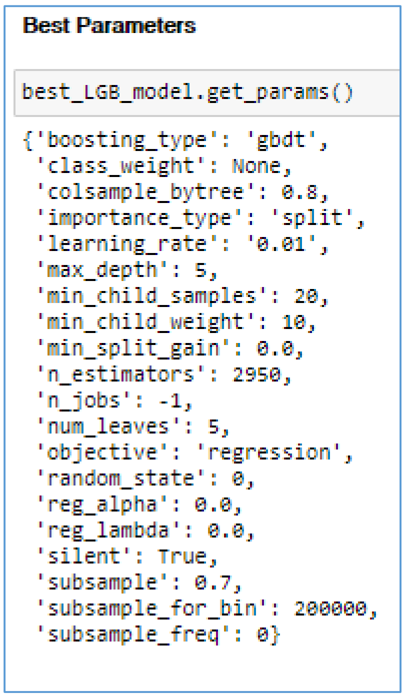
```
Random Search CV LGB Regressor Model Performance:
Best Parameters =  {'n_estimators': 2950, 'max_depth': 5}
Accuracy Metrics:
Mean Squared Error:  3.56
Root Mean Squared Error:  1.89
Mean Absolute Percentage Error:  16.86
```

After parameter tuning, the modelling results are:

Metrics	Random Forest	LightGBM
MSE	3.78	3.56
RMSE	1.94	1.89
MAPE	17.31	16.86

From above the best performing model is LightGBM Regressor.

The Best Parameters are:



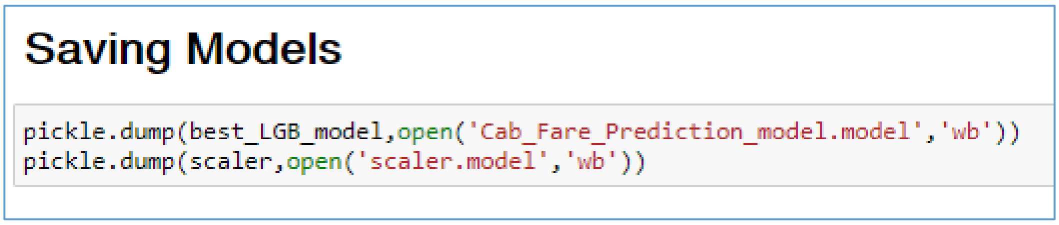
Best Parameters

```
best_LGB_model.get_params()
```

```
{'boosting_type': 'gbdt',
 'class_weight': None,
 'colsample_bytree': 0.8,
 'importance_type': 'split',
 'learning_rate': '0.01',
 'max_depth': 5,
 'min_child_samples': 20,
 'min_child_weight': 10,
 'min_split_gain': 0.0,
 'n_estimators': 2950,
 'n_jobs': -1,
 'num_leaves': 5,
 'objective': 'regression',
 'random_state': 0,
 'reg_alpha': 0.0,
 'reg_lambda': 0.0,
 'silent': True,
 'subsample': 0.7,
 'subsample_for_bin': 200000,
 'subsample_freq': 0}
```

Dumping Models

The best performing model is LightGBM model. Thus saving the LightGBM model along with the StandardScaler model by using pickle module to be used for the deployment of the prediction model.



Saving Models

```
pickle.dump(best_LGB_model,open('Cab_Fare_Prediction_model.model','wb'))
pickle.dump(scaler,open('scaler.model','wb'))
```

Conclusion

- Distance shows the max correlation with Fare amount. Thus it is the most important feature in predicting fare amount.
- Fare amount shows a slight linear relation with the year, i.e. the fare rates are increasing year by year.
- Months & Weekdays Doesn’t shows any major impact on fare amount. Fare rates is mostly uniform.
- Late Night & early morning hours shows lower number of rides but higher fare rates, while rest of the day the number of rides are higher but with lower fare rates.
- Passenger count doesn’t have any major impact on fare rates.
- On the basis of RMSE & MAPE the performing model is LightGBM Regressor.
- Best model RMSE = 1.89, MAPE = 16.86 %.