Project Report on

Cab Fare Prediction

-Saurabh Shrivastava

Contents

Chapter 1

Introduction:

1.1 Problem Statement –

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.

1.2 Data:

Below is the detail of the independent and dependent variable , we will analyse for solving the problem. Our task will be to analyse each variable distribution, wherever required scale them, study the correlation between independent and target variable .
Using graphical representation , we will study the dependency of independent variable with dependent variable to make our conclusion about what and which factor affecting the Fare amount of the cab. After study, analysing and our final selected predicting Model made from the given data, we have to predict the test data given to us

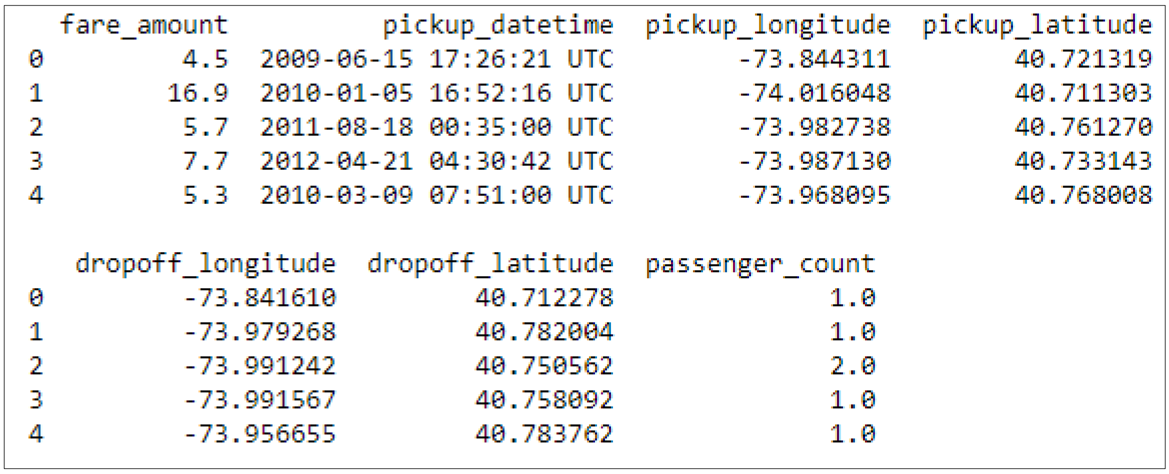
Below is independent variable we have :

1. pickup_datetime - timestamp value indicating when the cab ride started.
2. pickup_longitude - float for longitude coordinate of where the cab ride started.
3. pickup_latitude - float for latitude coordinate of where the cab ride started.
4. dropoff_longitude - float for longitude coordinate of where the cab ride ended.
5. dropoff_latitude - float for latitude coordinate of where the cab ride ended.
6. passenger_count - an integer indicating the number of passengers in the cab ride.

Below is the dependent/Target variable we have :

1. Fare_amount – contains the fare amount taken by cab for the ride

Below are first 5 observations of given data:



```
   fare_amount          pickup_datetime  pickup_longitude  pickup_latitude
0          4.5  2009-06-15 17:26:21 UTC        -73.844311        40.721319
1         16.9  2010-01-05 16:52:16 UTC        -74.016048        40.711303
2          5.7  2011-08-18 00:35:00 UTC        -73.982738        40.761270
3          7.7  2012-04-21 04:30:42 UTC        -73.987130        40.733143
4          5.3  2010-03-09 07:51:00 UTC        -73.968095        40.768008

   dropoff_longitude  dropoff_latitude  passenger_count
0         -73.841610         40.712278              1.0
1         -73.979268         40.782004              1.0
2         -73.991242         40.750562              2.0
3         -73.991567         40.758092              1.0
4         -73.956655         40.783762              1.0
```

Chapter 2

Methodology

2.1 Pre-Processing

Data pre-processing is an integral step in Machine Learning as the quality of data and the useful information that can be derived from it directly affects the ability of our model to learn; therefore, it is extremely important that we pre-process our data before feeding it into our model. This includes multiples process described below. At very first step it involve missing value analysis and checking invalid data which can affect our model accuracy. After which It followed by outlier analysis , feature selection, features scaling.

Summarised steps for Pre-processing

1. Data exploration and Cleaning
2. Missing values treatment
3. Outlier Analysis
4. Feature Selection
5. Features Scaling
6. Visualization

Visualisation includes graphical representation of data in terms of bar graph, histogram, heatmap and other to visualise the data for studying the characteristics of the data, dependency of the data on target variable and their mutual dependency.

Data Exploration and Cleaning:

<u>Datatype Analysis:</u>
<u>Below are the features given to us in dataset:</u>

1.	fare_amount	object
2.	pickup_datetime	object
3.	pickup_longitude	float64
4.	pickup_latitude	float64
5.	dropoff_longitude	float64
6.	dropoff_latitude	float64
7.	passenger_count	float64

On above data, we will convert the pickup_datetime to date type variable and fare_amount to numeric data type.

Treating invalid value :

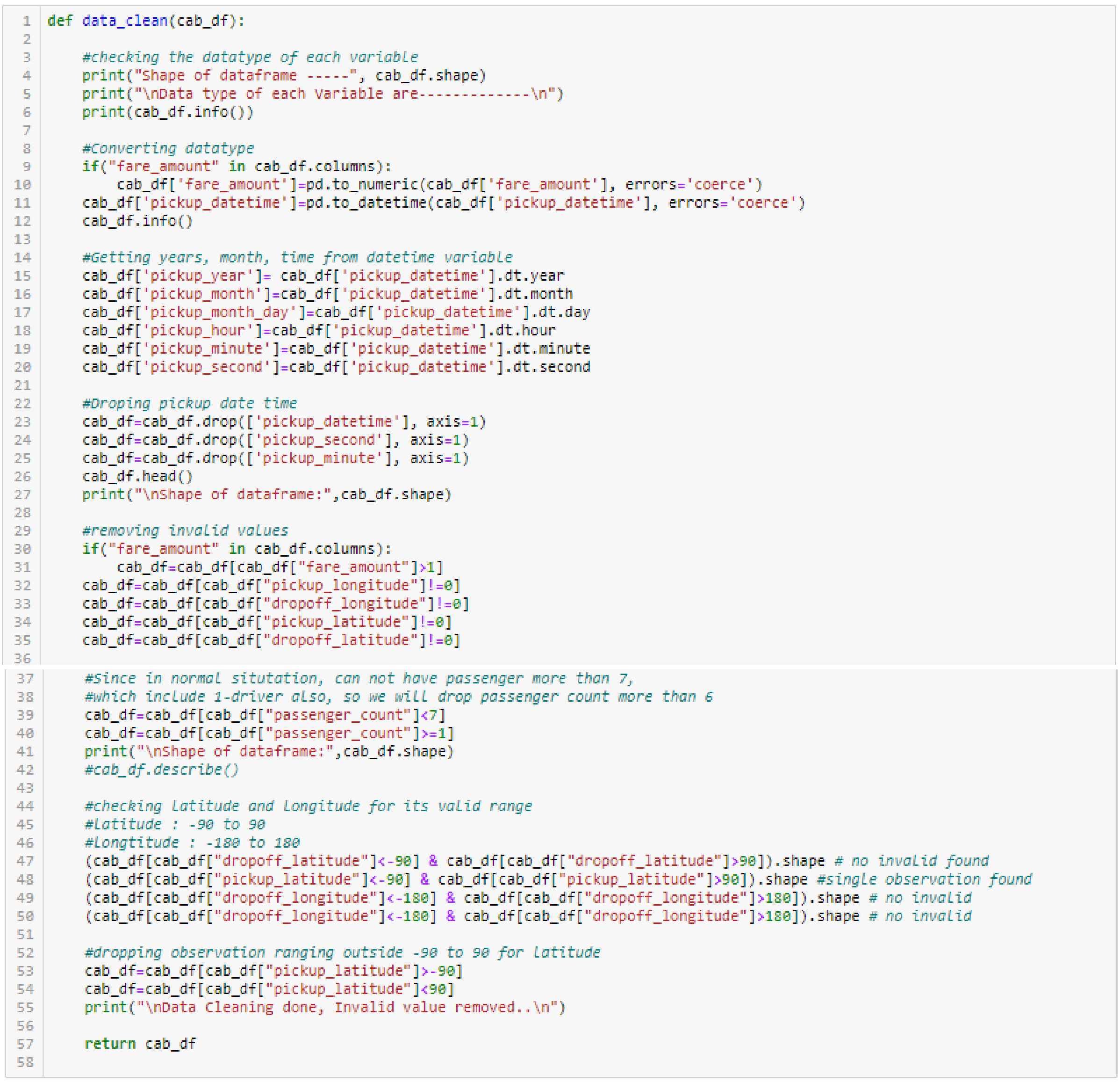
Analysing the data, we understood, there are certain value in certain column which are not the valid value for the column.

We have few fare amount observations which are 0 and below, which is not possible after the Cab ride. Hence, it's invalid entry in our variable, we will remove this observation.

Secondly, the latitude and longitude having 0 in them and we are aware the 0-denotes the region in Atlantic Ocean, which is again not possible for any cab. Hence, all observation having 0's in them, we will remove those observation considering them invalid.
For any standard cab, the maximum person sit inside is 6+1, where 1-denotes the driver. We found passenger count more than 7 and less than 1, hence we remove those observation considering them invalid .

Briefly we perform these step:

- Separate the combined variables.
- As we know we have some negative values in fare amount so we have to remove those values.
- Passenger count would be max 6 if it is a SUV vehicle not more than that. We have to remove the rows having passengers counts more than 6 and less than 1.
- There are some outlier figures in the fare so we need to remove those. e. Latitudes range from -90 to 90. Longitudes range from -180 to 180. We need to remove the rows if any latitude and longitude lies beyond the ranges.



```
def data_clean(cab_df):

    #checking the datatype of each variable
    print("Shape of dataframe -----", cab_df.shape)
    print("\nData type of each Variable are-------------\n")
    print(cab_df.info())

    #Converting datatype
    if("fare_amount" in cab_df.columns):
        cab_df['fare_amount']=pd.to_numeric(cab_df['fare_amount'], errors='coerce')
    cab_df['pickup_datetime']=pd.to_datetime(cab_df['pickup_datetime'], errors='coerce')
    cab_df.info()

    #Getting years, month, time from datetime variable
    cab_df['pickup_year']= cab_df['pickup_datetime'].dt.year
    cab_df['pickup_month']=cab_df['pickup_datetime'].dt.month
    cab_df['pickup_month_day']=cab_df['pickup_datetime'].dt.day
    cab_df['pickup_hour']=cab_df['pickup_datetime'].dt.hour
    cab_df['pickup_minute']=cab_df['pickup_datetime'].dt.minute
    cab_df['pickup_second']=cab_df['pickup_datetime'].dt.second

    #Droping pickup date time
    cab_df=cab_df.drop(['pickup_datetime'], axis=1)
    cab_df=cab_df.drop(['pickup_second'], axis=1)
    cab_df=cab_df.drop(['pickup_minute'], axis=1)
    cab_df.head()
    print("\nShape of dataframe:",cab_df.shape)

    #removing invalid values
    if("fare_amount" in cab_df.columns):
        cab_df=cab_df[cab_df["fare_amount"]>1]
    cab_df=cab_df[cab_df["pickup_longitude"]!=0]
    cab_df=cab_df[cab_df["dropoff_longitude"]!=0]
    cab_df=cab_df[cab_df["pickup_latitude"]!=0]
    cab_df=cab_df[cab_df["dropoff_latitude"]!=0]

    #Since in normal situtation, can not have passenger more than 7,
    #which include 1-driver also, so we will drop passenger count more than 6
    cab_df=cab_df[cab_df["passenger_count"]<7]
    cab_df=cab_df[cab_df["passenger_count"]>=1]
    print("\nShape of dataframe:",cab_df.shape)
    #cab_df.describe()

    #checking latitude and longitude for its valid range
    #latitude : -90 to 90
    #longtitude : -180 to 180
    (cab_df[cab_df["dropoff_latitude"]<-90] & cab_df[cab_df["dropoff_latitude"]>90]).shape # no invalid found
    (cab_df[cab_df["pickup_latitude"]<-90] & cab_df[cab_df["pickup_latitude"]>90]).shape #single observation found
    (cab_df[cab_df["dropoff_longitude"]<-180] & cab_df[cab_df["dropoff_longitude"]>180]).shape # no invalid
    (cab_df[cab_df["dropoff_longitude"]<-180] & cab_df[cab_df["dropoff_longitude"]>180]).shape # no invalid

    #dropping observation ranging outside -90 to 90 for latitude
    cab_df=cab_df[cab_df["pickup_latitude"]>-90]
    cab_df=cab_df[cab_df["pickup_latitude"]<90]
    print("\nData Cleaning done, Invalid value removed..\n")

    return cab_df

```

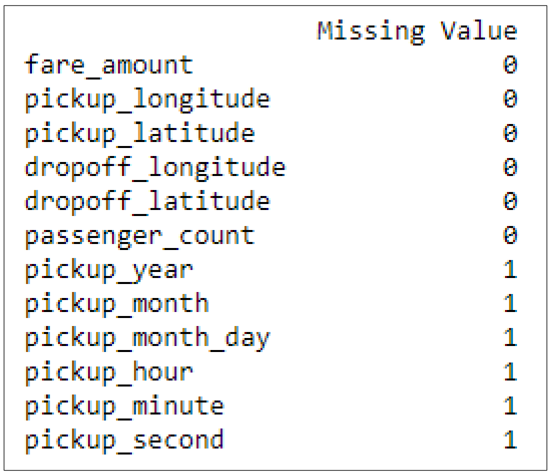
Missing Value Treatment

We are aware, today, we have many sources of data, which usually contains many junk value and missing value. Treating this missing value is crucial process before feeding the data to any machine learning model. It helps in ensuring more accurate and precise prediction from the model.

For imputing the missing value, we have various methods like:

1. Mean Imputation
2. Median imputation
3. Mode Imputation and KNN imputation.

Apart from these, we often use other ways of imputation also, depending on given case. Here we have single missing value from few variables (refer to below missing value table), which we can simply drop along with its corresponding observation, as a treatment for Missing Value.



```
                    Missing Value
fare_amount                     0
pickup_longitude                0
pickup_latitude                 0
dropoff_longitude               0
dropoff_latitude                0
passenger_count                 0
pickup_year                     1
pickup_month                    1
pickup_month_day                1
pickup_hour                     1
pickup_minute                   1
pickup_second                   1
```

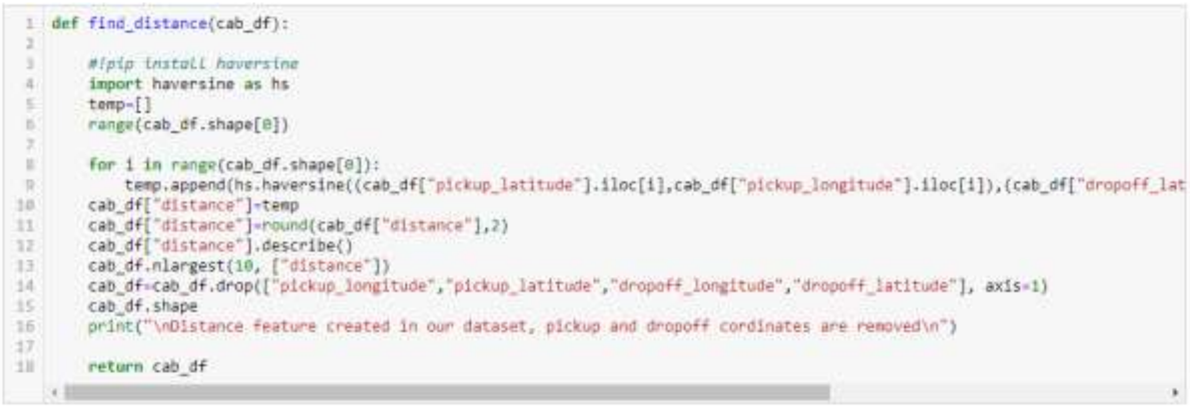
FEATURE SELECTION:

Feature Selection is the process of choosing features that are most useful for your prediction. Whenever required we also do feature creation from the existing given feature, to make our analysis more effective.

Feature selection provides an effective way for removing irrelevant and redundant data, which can reduce computation time, improve learning accuracy, and facilitate a better understanding for the learning model or data.

Here we start with generating “distance” features from pickup and dropoff coordinates. For these, we will use haversine formula, which requires pickup and dropoff longitude and latitude , for calculating distance.

Calculating Distance from Haversine method:

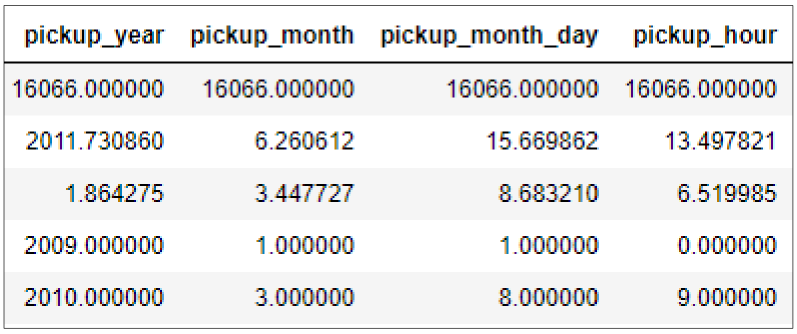


```python
def find_distance(cab_df):

    #!pip install haversine
    import haversine as hs
    temp=[]
    range(cab_df.shape[0])

    for i in range(cab_df.shape[0]):
        temp.append(hs.haversine((cab_df["pickup_latitude"].iloc[i],cab_df["pickup_longitude"].iloc[i]),(cab_df["dropoff_lat
    cab_df["distance"]=temp
    cab_df["distance"]=round(cab_df["distance"],2)
    cab_df["distance"].describe()
    cab_df.nlargest(10, ["distance"])
    cab_df=cab_df.drop(["pickup_longitude","pickup_latitude","dropoff_longitude","dropoff_latitude"], axis=1)
    cab_df.shape
    print("\nDistance feature created in our dataset, pickup and dropoff cordinates are removed\n")

    return cab_df
```

Extracting Date and Time :

From the given data , we see the date and time stamp is given in single format. Using them in their separate entity will provide us better understanding of their effect on our target variable. Hence, we will extract years, month, day and hours from the given time stamp.

Sample of Date and Time extracted from Time Stamp are :

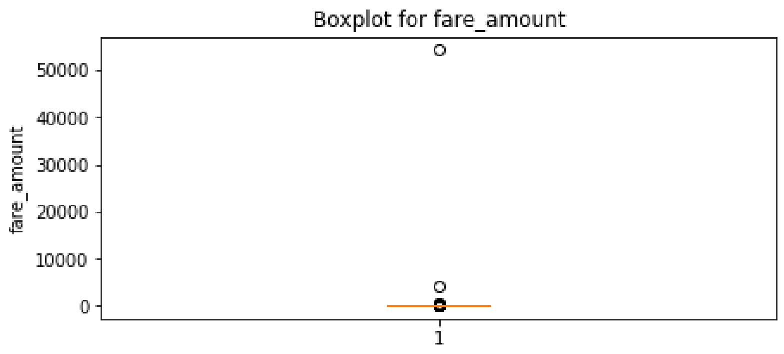


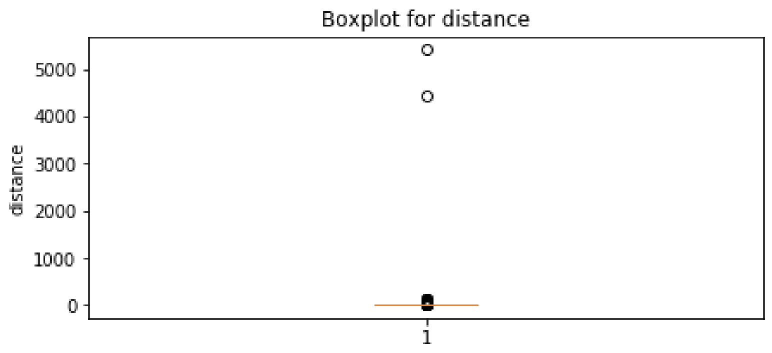
pickup_year	pickup_month	pickup_month_day	pickup_hour
16066.000000	16066.000000	16066.000000	16066.000000
2011.730860	6.260612	15.669862	13.497821
1.864275	3.447727	8.683210	6.519985
2009.000000	1.000000	1.000000	0.000000
2010.000000	3.000000	8.000000	9.000000

After selecting the date and time from the given time stamp, we will drop the time stamp variable from our dataset

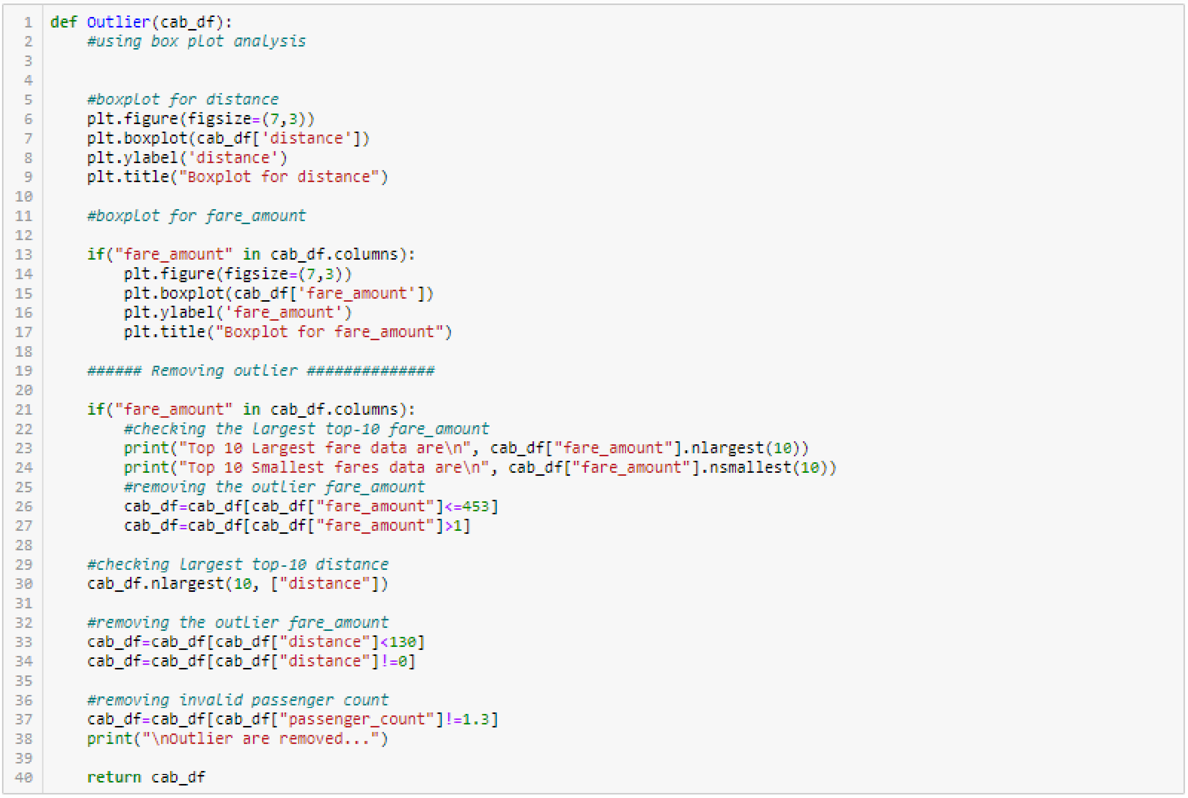
Outlier Analysis:

Here we will check our continuous variable for outliers, we will plot Boxplot for fare amount and distance to check the outlier graphically.





For these we check the number of outlier in total observation by boxplot. We then used manual method to check number of outlier. Since the number of outliers was few as compared to total observation , After getting the outlier in our data we simply removed those outlier along with their corresponding observation. We compared the model accuracy in both way, that is, boxplot removal and manual outlier removal. Manual removal has more accuracy then boxplot removal.

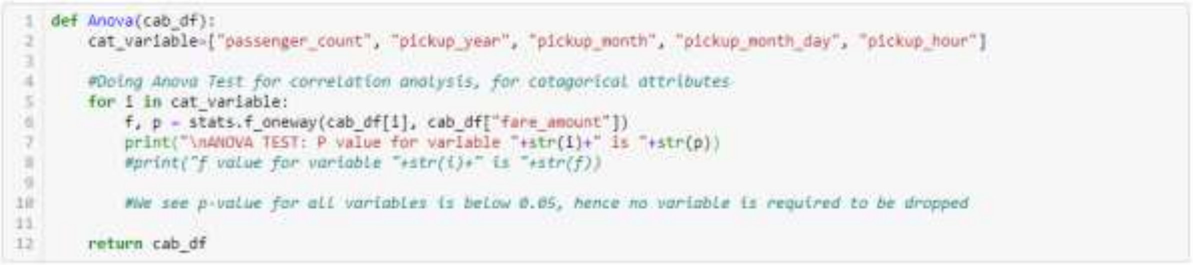


```python
def Outlier(cab_df):
    #using box plot analysis


    #boxplot for distance
    plt.figure(figsize=(7,3))
    plt.boxplot(cab_df['distance'])
    plt.ylabel('distance')
    plt.title("Boxplot for distance")

    #boxplot for fare_amount

    if("fare_amount" in cab_df.columns):
        plt.figure(figsize=(7,3))
        plt.boxplot(cab_df['fare_amount'])
        plt.ylabel('fare_amount')
        plt.title("Boxplot for fare_amount")

    ###### Removing outlier ##############

    if("fare_amount" in cab_df.columns):
        #checking the largest top-10 fare_amount
        print("Top 10 Largest fare data are\n", cab_df["fare_amount"].nlargest(10))
        print("Top 10 Smallest fares data are\n", cab_df["fare_amount"].nsmallest(10))
        #removing the outlier fare_amount
        cab_df=cab_df[cab_df["fare_amount"]<=453]
        cab_df=cab_df[cab_df["fare_amount"]>1]

    #checking largest top-10 distance
    cab_df.nlargest(10, ["distance"])

    #removing the outlier fare_amount
    cab_df=cab_df[cab_df["distance"]<130]
    cab_df=cab_df[cab_df["distance"]!=0]

    #removing invalid passenger count
    cab_df=cab_df[cab_df["passenger_count"]!=1.3]
    print("\nOutlier are removed...")

    return cab_df
```

Correlation Analysis:

Correlation Analysis is statistical method that is used to discover if there is a relationship between two variables/datasets, and how strong that relationship may be. We checks if two or more independent variable are highly correlated with each other, doing so we can avoid the inaccuracy raises from multicollinearity. Here for categorical variable we will use ANOVA.

Anova: We see that p-value <0.05. Hence, we can reject the Null Hypothesis – there are no differences among different density groups. Hence we do not need to drop any categorical column on the basis of ANOVA test.



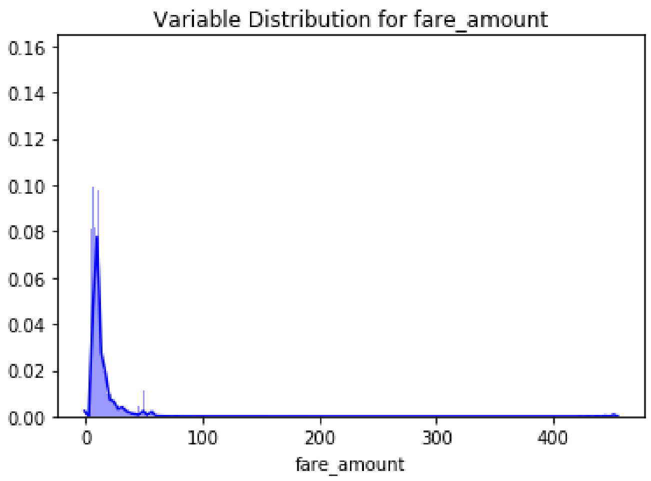
```python
def Anova(cab_df):
    cat_variable=["passenger_count", "pickup_year", "pickup_month", "pickup_month_day", "pickup_hour"]

    #Doing Anova Test for correlation analysis, for catagorical attributes
    for i in cat_variable:
        f, p = stats.f_oneway(cab_df[i], cab_df["fare_amount"])
        print("\nANOVA TEST: P value for variable "+str(i)+" is "+str(p))
        #print("f value for variable "+str(i)+" is "+str(f))

        #We see p-value for all variables is below 0.05, hence no variable is required to be dropped

    return cab_df
```

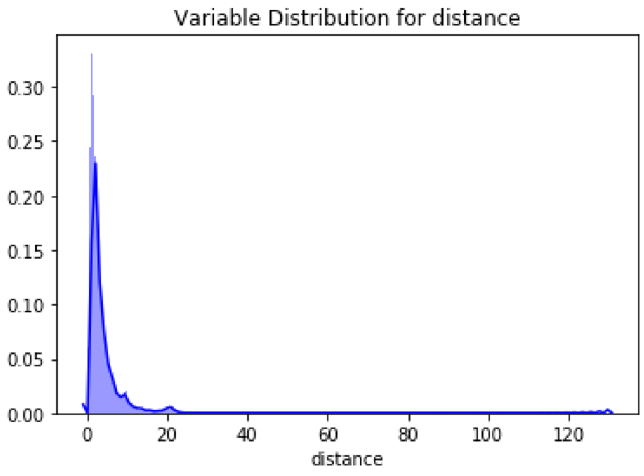
```
ANOVA TEST: P value for variable passenger_count is 0.0

ANOVA TEST: P value for variable pickup_year is 0.0

ANOVA TEST: P value for variable pickup_month is 0.0

ANOVA TEST: P value for variable pickup_month_day is 0.0

ANOVA TEST: P value for variable pickup_hour is 1.0805697402574481e-101
```

Feature Scaling:

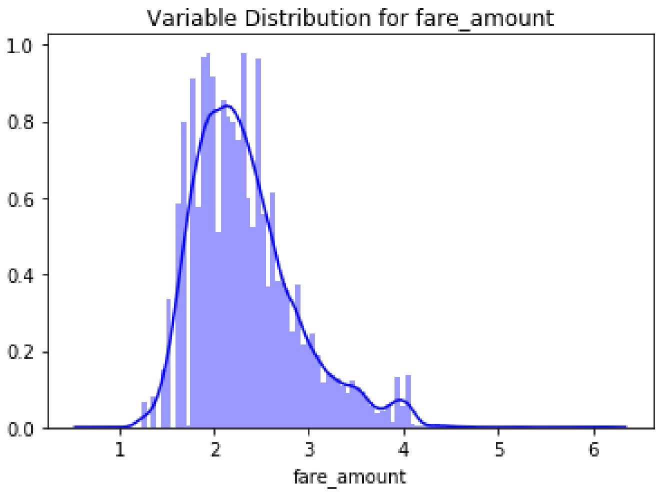
Feature scaling is a method used to normalize the range of independent variables or features of data by reducing the skewness. In data processing, it is also known as data normalization and is generally performed during the data pre-processing step.
Skewness is asymmetry in a statistical distribution, in which the curve appears distorted or skewed either to the left or to the right. Skewness can be quantified to define the extent to which a distribution differs from a normal distribution. Here we tried to show the skewness of our variables and we find that distance and fare_amount one sided skewed so by using log transform technique we tried to reduce the skewness of the same.

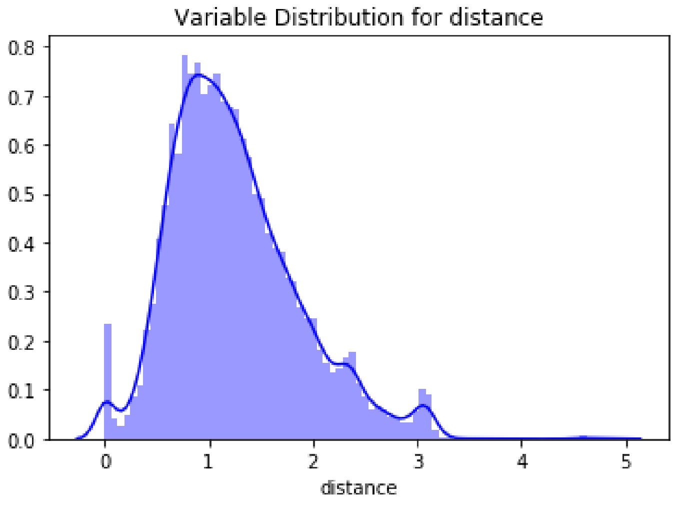
Skewness before log transformation ,by plotting graph, which are :





From figure above, we can see the data is left skewed. We will try to do log transformation to reduce skewness. After log transformation, our data looks like :





Since after log transformation, our data is now less skewed and follow bell curve, we will not apply scaling technique like normalisation and standardisation.

2.2 Modelling

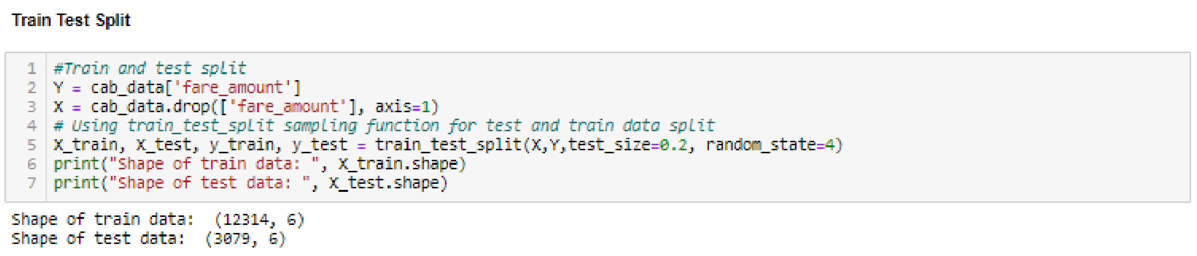
Model Selection:

After a thorough pre-processing, We will use some regression models on our processed data to predict the target variable. Since our target variable is continuous variable we go for Regression model. Following are the models which we have built –

1. Linear Regression
2. Decision Tree Regression
3. Random Forest Regression
4. Gradient Boosting Regression

#Train – Test Split :

Before running any model, we will split our data into two parts which is train and test data. Here in our case we have taken 80% of the data as our train data. Below is the snipped image of the split of train test.



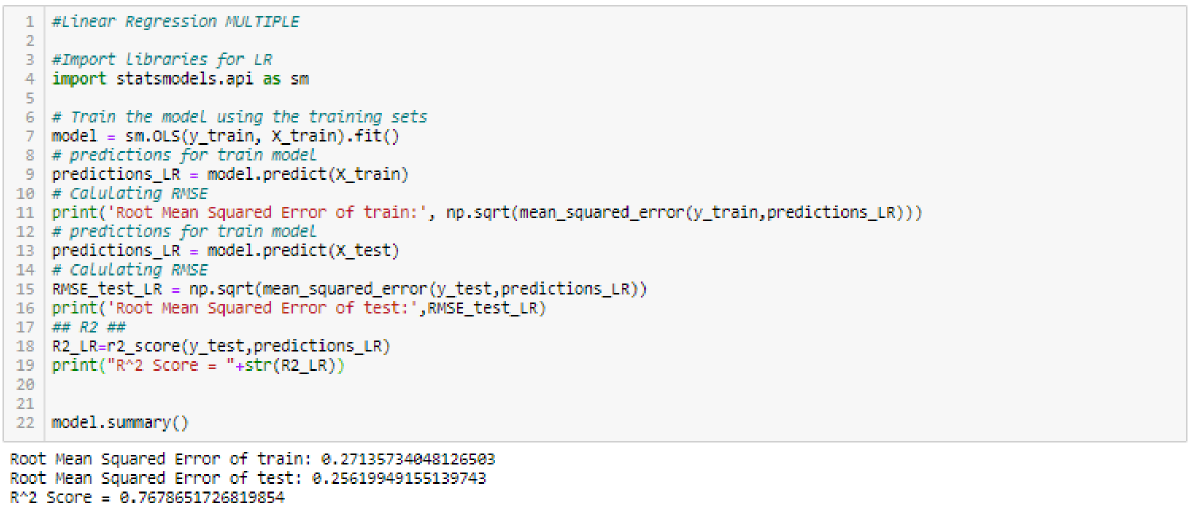
Train Test Split

```
#Train and test split
Y = cab_data['fare_amount']
X = cab_data.drop(['fare_amount'], axis=1)
# Using train_test_split sampling function for test and train data split
X_train, X_test, y_train, y_test = train_test_split(X,Y,test_size=0.2, random_state=4)
print("Shape of train data: ", X_train.shape)
print("Shape of test data: ", X_test.shape)
```

```
Shape of train data:  (12314, 6)
Shape of test data:  (3079, 6)
```

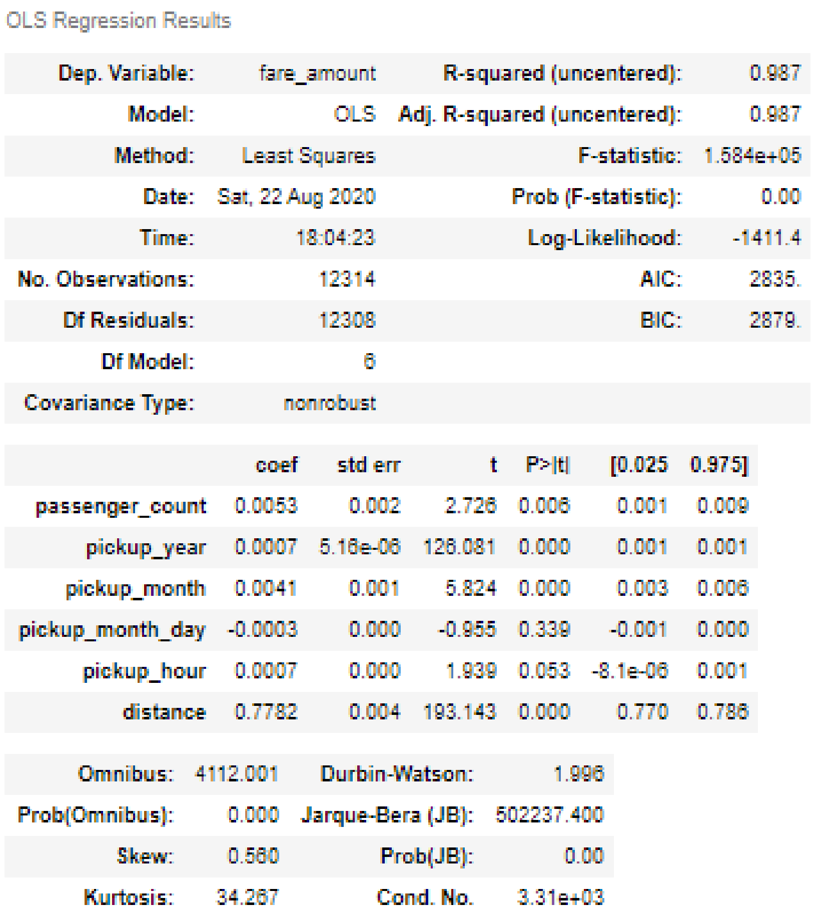
Linear Regression:

Multiple linear regression is the most common form of linear regression analysis. Multiple regression is an extension of simple linear regression. It is used as a predictive analysis, when we want to predict the value of a variable based on the value of two or more other variables. The variable we want to predict is called the dependent variable (or sometimes, the outcome, target or criterion variable). Below is a screenshot of the model we build and its output



```python
#Linear Regression MULTIPLE

#Import libraries for LR
import statsmodels.api as sm

# Train the model using the training sets
model = sm.OLS(y_train, X_train).fit()
# predictions for train model
predictions_LR = model.predict(X_train)
# Calulating RMSE
print('Root Mean Squared Error of train:', np.sqrt(mean_squared_error(y_train,predictions_LR)))
# predictions for train model
predictions_LR = model.predict(X_test)
# Calulating RMSE
RMSE_test_LR = np.sqrt(mean_squared_error(y_test,predictions_LR))
print('Root Mean Squared Error of test:',RMSE_test_LR)
## R2 ##
R2_LR=r2_score(y_test,predictions_LR)
print("R^2 Score = "+str(R2_LR))


model.summary()
```

```
Root Mean Squared Error of train: 0.27135734048126503
Root Mean Squared Error of test: 0.25619949155139743
R^2 Score = 0.7678651726819854
```



OLS Regression Results

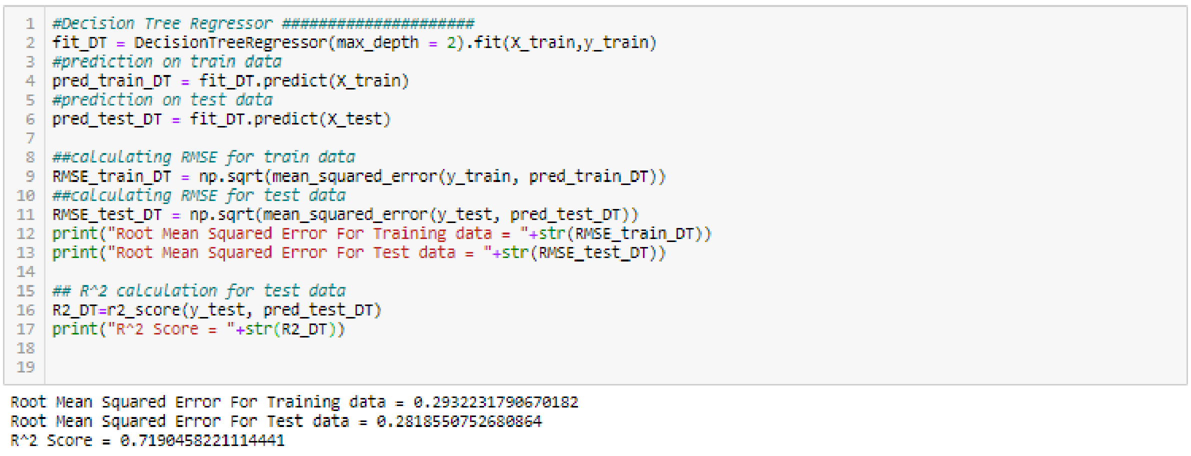
Dep. Variable:	fare_amount	R-squared (uncentered):	0.987
Model:	OLS	Adj. R-squared (uncentered):	0.987
Method:	Least Squares	F-statistic:	1.584e+05
Date:	Sat, 22 Aug 2020	Prob (F-statistic):	0.00
Time:	18:04:23	Log-Likelihood:	-1411.4
No. Observations:	12314	AIC:	2835.
Df Residuals:	12308	BIC:	2879.
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
passenger_count	0.0053	0.002	2.726	0.006	0.001	0.009
pickup_year	0.0007	5.16e-06	128.081	0.000	0.001	0.001
pickup_month	0.0041	0.001	5.824	0.000	0.003	0.006
pickup_month_day	-0.0003	0.000	-0.955	0.339	-0.001	0.000
pickup_hour	0.0007	0.000	1.939	0.053	-8.1e-06	0.001
distance	0.7782	0.004	193.143	0.000	0.770	0.786

Omnibus:	4112.001	Durbin-Watson:	1.996
Prob(Omnibus):	0.000	Jarque-Bera (JB):	502237.400
Skew:	0.560	Prob(JB):	0.00
Kurtosis:	34.267	Cond. No.	3.31e+03

Decision Tree

A tree has many analogies in real life, and turns out that it has influenced a wide area of machine learning, covering both classification and regression. In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. As the name goes, it uses a tree-like model of decisions. Below is the screenshot of the query we executed and the result shown, we will compare the results of each model in a combined table later on.

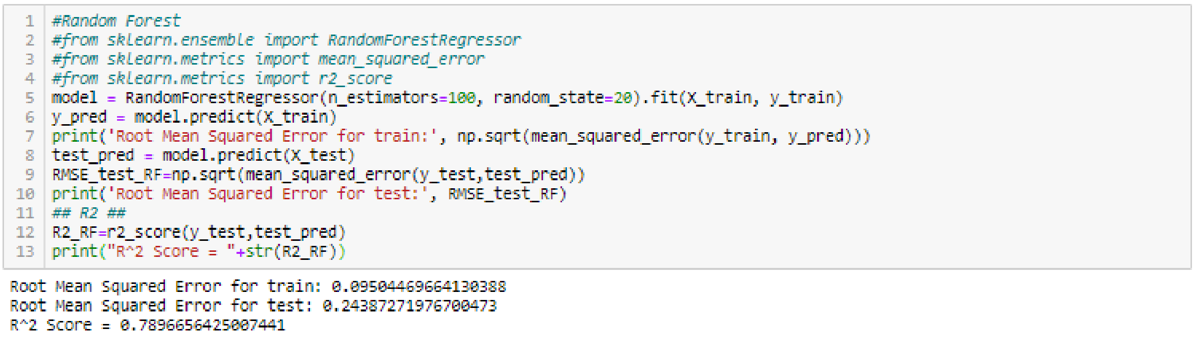


```
#Decision Tree Regressor #####################
fit_DT = DecisionTreeRegressor(max_depth = 2).fit(X_train,y_train)
#prediction on train data
pred_train_DT = fit_DT.predict(X_train)
#prediction on test data
pred_test_DT = fit_DT.predict(X_test)

##calculating RMSE for train data
RMSE_train_DT = np.sqrt(mean_squared_error(y_train, pred_train_DT))
##calculating RMSE for test data
RMSE_test_DT = np.sqrt(mean_squared_error(y_test, pred_test_DT))
print("Root Mean Squared Error For Training data = "+str(RMSE_train_DT))
print("Root Mean Squared Error For Test data = "+str(RMSE_test_DT))

## R^2 calculation for test data
R2_DT=r2_score(y_test, pred_test_DT)
print("R^2 Score = "+str(R2_DT))
```

```
Root Mean Squared Error For Training data = 0.2932231790670182
Root Mean Squared Error For Test data = 0.2818550752680864
R^2 Score = 0.7190458221114441
```

Random Forest:

Random forests or random decision forests are an ensemble learning method for classification, regression and other task, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set. In simple words: Random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction.

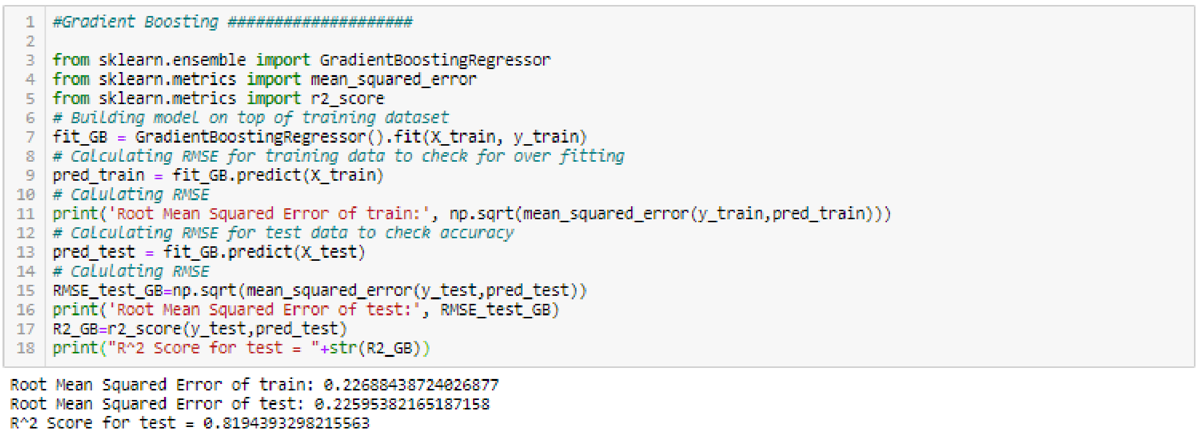


```
#Random Forest
#from sklearn.ensemble import RandomForestRegressor
#from sklearn.metrics import mean_squared_error
#from sklearn.metrics import r2_score
model = RandomForestRegressor(n_estimators=100, random_state=20).fit(X_train, y_train)
y_pred = model.predict(X_train)
print('Root Mean Squared Error for train:', np.sqrt(mean_squared_error(y_train, y_pred)))
test_pred = model.predict(X_test)
RMSE_test_RF=np.sqrt(mean_squared_error(y_test,test_pred))
print('Root Mean Squared Error for test:', RMSE_test_RF)
## R2 ##
R2_RF=r2_score(y_test,test_pred)
print("R^2 Score = "+str(R2_RF))
```

```
Root Mean Squared Error for train: 0.09504469664130388
Root Mean Squared Error for test: 0.24387271976700473
R^2 Score = 0.7896656425007441
```

Gradient Boosting:

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. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.



```python
#Gradient Boosting ####################

from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
# Building model on top of training dataset
fit_GB = GradientBoostingRegressor().fit(X_train, y_train)
# Calculating RMSE for training data to check for over fitting
pred_train = fit_GB.predict(X_train)
# Calulating RMSE
print('Root Mean Squared Error of train:', np.sqrt(mean_squared_error(y_train,pred_train)))
# Calculating RMSE for test data to check accuracy
pred_test = fit_GB.predict(X_test)
# Calulating RMSE
RMSE_test_GB=np.sqrt(mean_squared_error(y_test,pred_test))
print('Root Mean Squared Error of test:', RMSE_test_GB)
R2_GB=r2_score(y_test,pred_test)
print("R^2 Score for test = "+str(R2_GB))
```

```
Root Mean Squared Error of train: 0.22688438724026877
Root Mean Squared Error of test: 0.22595382165187158
R^2 Score for test = 0.8194393298215563
```

Chapter 3

Conclusion

Model Evaluation:

Model Evaluation is an integral part of the model development process. It helps to find the best model that represents our data and how well the chosen model will work in the future.

For Regression model, we have Root mean square error, R-Square (Coefficient of determination), Mean absolute percentage error, mean squared error , mean absolute error

Here we are utilising RMSE and R-Square to evaluate our Model performance and selecting final model.

Choosing Root Mean Square Error over other method

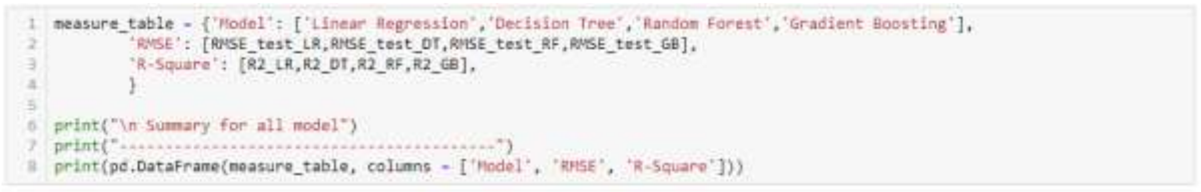
1. In RMSE method -by squaring the errors we can get more accurate results as the negative and positive errors don't cancel out each other and stay in existence till the end of commutation, thus adding more accuracy to the result.
2. Since the errors are squared before they are averaged, the RMSE gives a relatively high weight to large errors.
3. One distinct advantage of RMSE over MAE is that RMSE avoids the use of taking the absolute value, which is undesirable in many mathematical calculations.

R Square

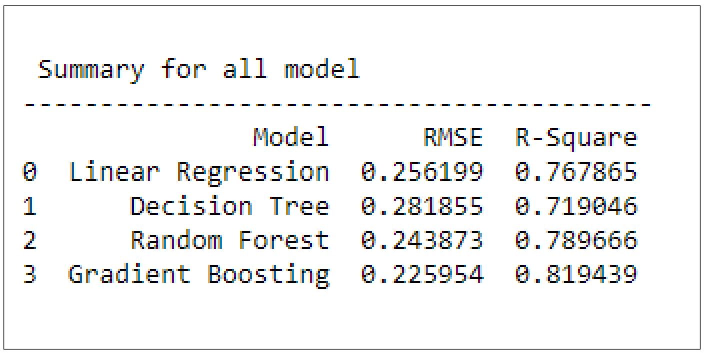
This statistic indicates the percentage of the variance in the dependent variable that the independent variables explain collectively. R-squared measures the strength of the relationship between your model and the dependent variable on a convenient 0 to 1 scale or 0 – 100% scale.

Model Selection:

Here we find out which model has lesser RMSE value and higher R-Square value. We need to refer to below table, which we generated using given codes.



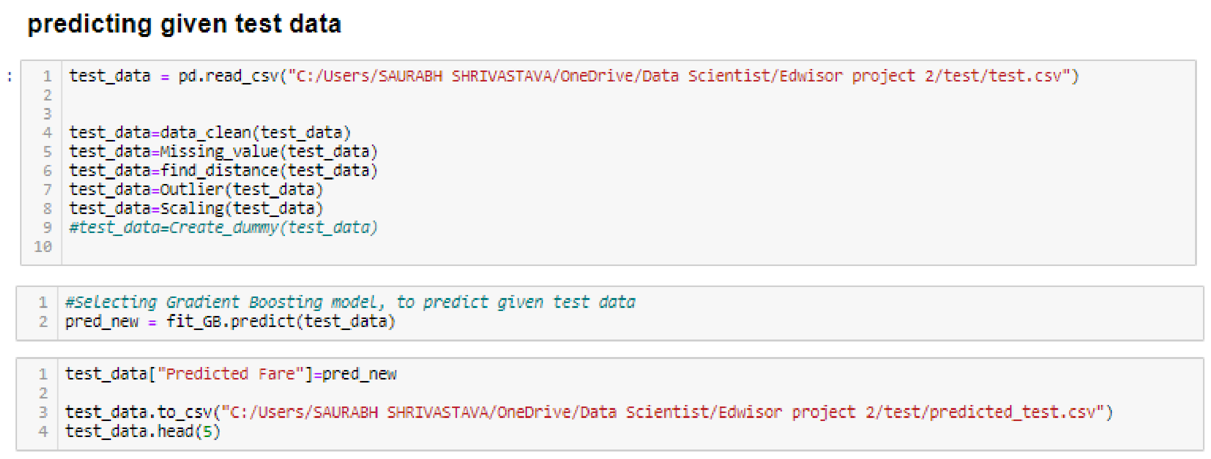
```python
measure_table = {'Model': ['Linear Regression','Decision Tree','Random Forest','Gradient Boosting'],
         'RMSE': [RMSE_test_LR,RMSE_test_DT,RMSE_test_RF,RMSE_test_GB],
         'R-Square': [R2_LR,R2_DT,R2_RF,R2_GB],
         }

print("\n Summary for all model")
print("----------------------------------------")
print(pd.DataFrame(measure_table, columns = ['Model', 'RMSE', 'R-Square']))
```



```

 Summary for all model
----------------------------------------
               Model      RMSE  R-Square
0  Linear Regression  0.256199  0.767865
1      Decision Tree  0.281855  0.719046
2      Random Forest  0.243873  0.789666
3  Gradient Boosting  0.225954  0.819439
```

From the table above, we can clearly see, RMSE value is least for Gradient Boosting and R-Square Value is highest for Gradient Boosting**..**

Hence our desirable best model is Gradient Boosting..



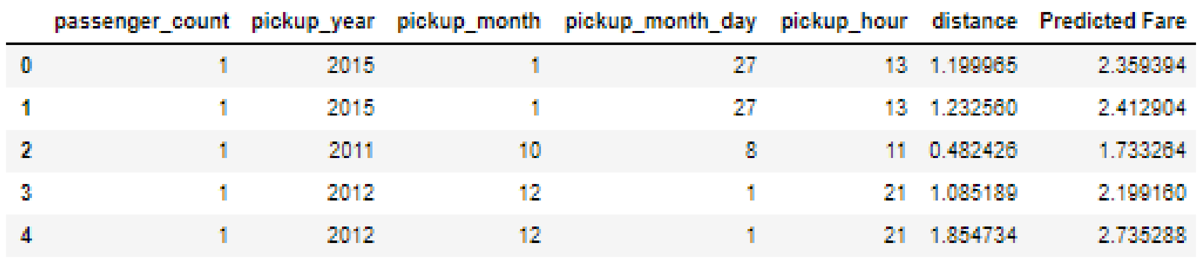
predicting given test data

```python
test_data = pd.read_csv("C:/Users/SAURABH SHRIVASTAVA/OneDrive/Data Scientist/Edwisor project 2/test/test.csv")


test_data=data_clean(test_data)
test_data=Missing_value(test_data)
test_data=find_distance(test_data)
test_data=Outlier(test_data)
test_data=Scaling(test_data)
#test_data=Create_dummy(test_data)

```

```python
#Selecting Gradient Boosting model, to predict given test data
pred_new = fit_GB.predict(test_data)
```

```python
test_data["Predicted Fare"]=pred_new

test_data.to_csv("C:/Users/SAURABH SHRIVASTAVA/OneDrive/Data Scientist/Edwisor project 2/test/predicted_test.csv")
test_data.head(5)
```

Top five rows after predicting the fare amount:



	passenger_count	pickup_year	pickup_month	pickup_month_day	pickup_hour	distance	Predicted Fare
0	1	2015	1	27	13	1.199965	2.359394
1	1	2015	1	27	13	1.232560	2.412904
2	1	2011	10	8	11	0.482426	1.733264
3	1	2012	12	1	21	1.085189	2.199160
4	1	2012	12	1	21	1.854734	2.735288

Running in DOS:

For running in DOS/Command Prompt , we save our python coding in .py format. Then we have to follow below steps :

1. Open Command Prompt.
2. Set to your working directory. (where you have your .py file stored)
3. Type below command:
   ➔ python cab_fare.py

Above method may not work if the python path is not registered in “environment variable”. In that case we use below:

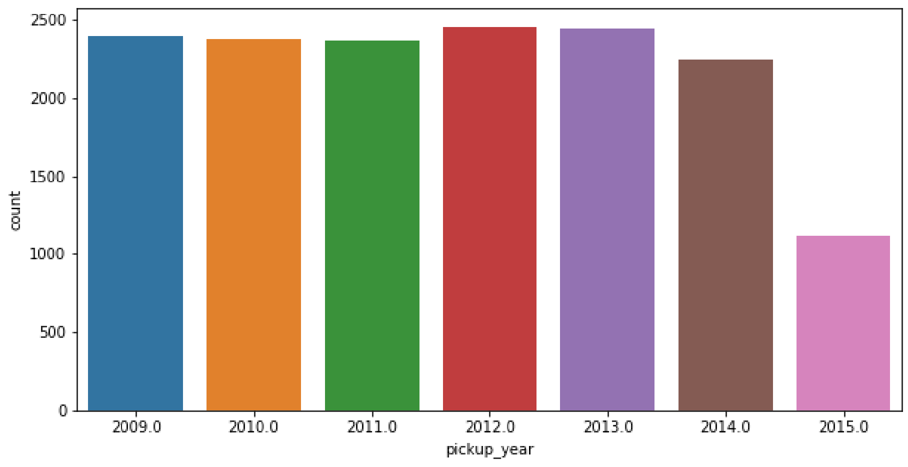
1. Anaconda prompt
2. Jupyter Notebook terminal etc.

Chapter 4

Data Visualisation

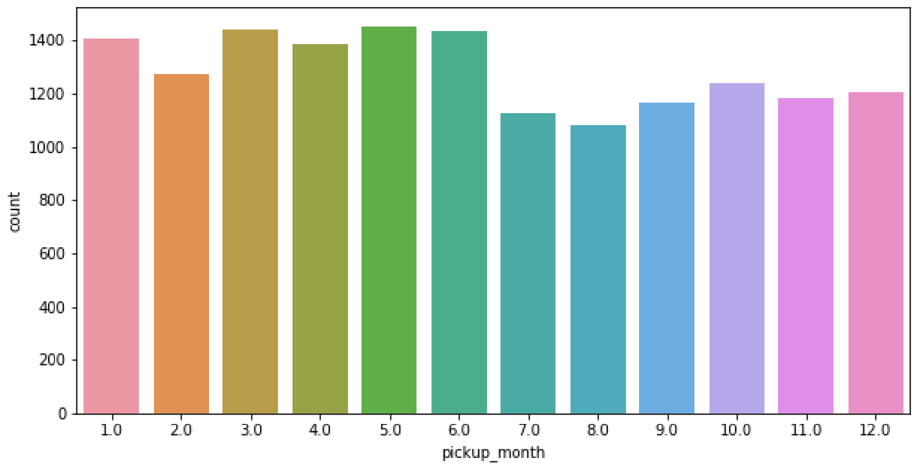
#Years vs Cab Count

Here we found on year 2015 cab count was low as compared to other year.



#Month vs Cab Count

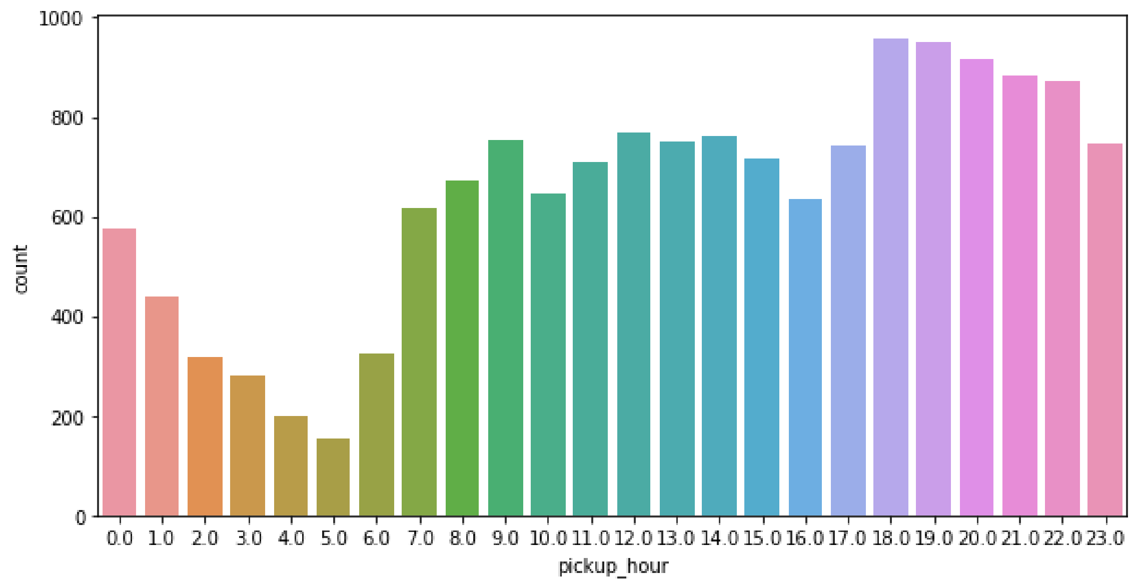
Here we analysed cab demand is more in first half of the year i.e. from January to June.



#Pickup Hour vs Cab Count

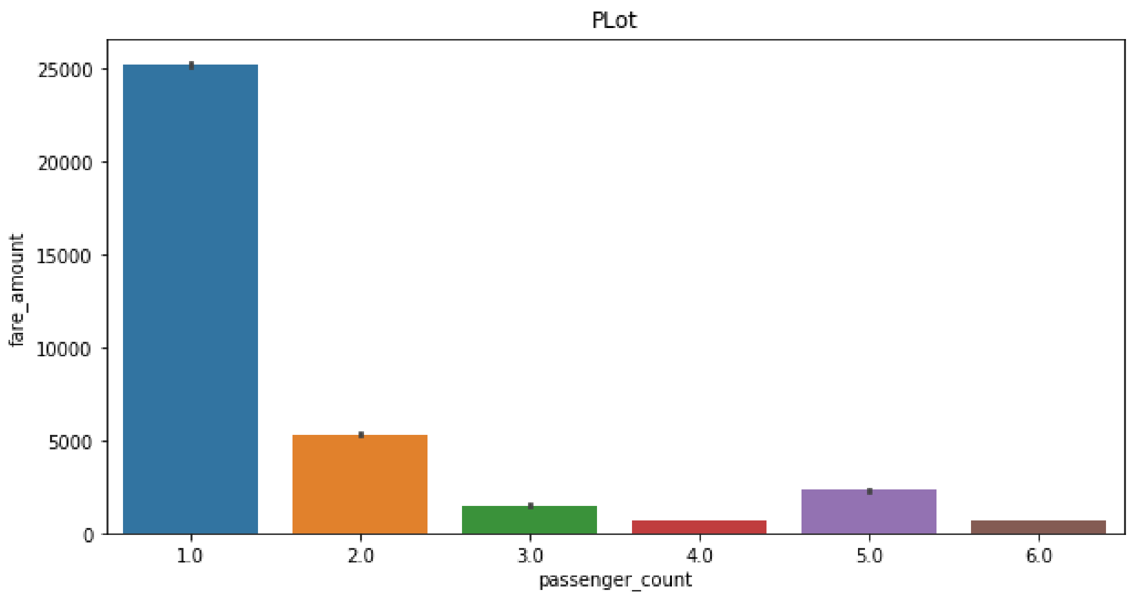
Here we see the cab demand is more in night time from 6.PM to 11P.M, probably because job employee would be going back to their home after job.

Cab Demand is less during early morning and increases gradually afterwards.



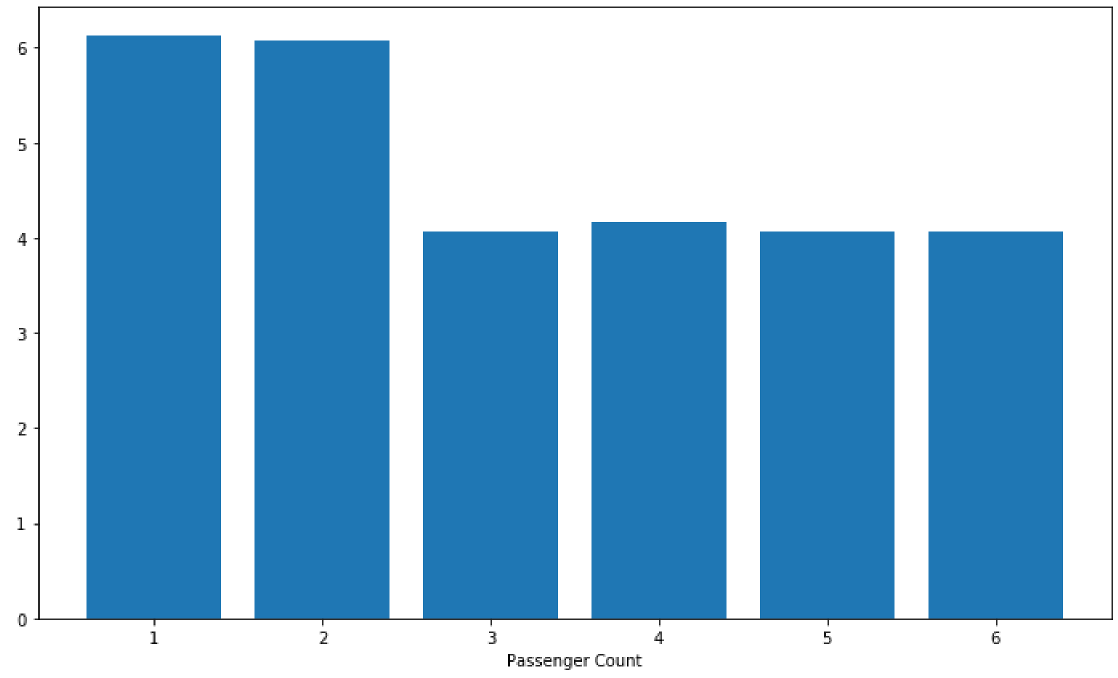
#Passenger Count vs Fare Amount:

Fare amount is more for the cab having single passenger.



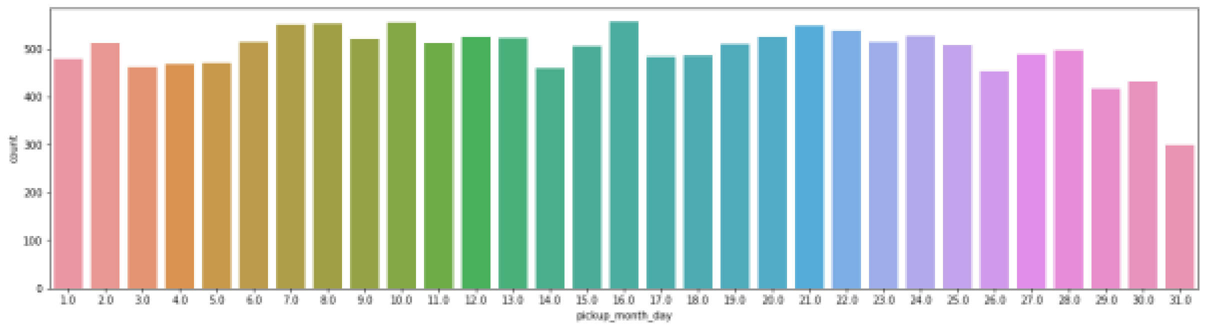
Passenger Count vs Cab Count

Here we found single and double passengers takes more cab.



#Pickup Day vs Cab Count

Cab demand is slightly more in mid of the month compare to be beginning and end of the month.



Reference:

1. https://www.wikipedia.org/
2. https://towardsdatascience.com/understanding-gradient-boosting-machines-9be756fe76ab

Python Code:

```
import os
import pandas as pd
import numpy as np
import math

#------ for model evaluation -----
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

#----- for preprocessing
from sklearn.preprocessing import Normalizer
from sklearn.preprocessing import StandardScaler

#---- for model building
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor

#---- for visualization---
import matplotlib.pyplot as plt
import seaborn as sns

from scipy import stats
from sklearn.model_selection import train_test_split

os.chdir("C:/Users/SAURABH SHRIVASTAVA/OneDrive/Data Scientist/Edwisor project 2")
os.getcwd()

cab_data = pd.read_csv("C:/Users/SAURABH SHRIVASTAVA/OneDrive/Data Scientist/Edwisor project 2/train_cab.csv")
```

```
print(cab_data.head())

def data_clean(cab_df):

    #checking the datatype of each variable
    print("Shape of dataframe -----", cab_df.shape)
    print("\nData type of each Variable are-------------\n")
    print(cab_df.info())

    #Converting datatype
    if("fare_amount" in cab_df.columns):
        cab_df['fare_amount']=pd.to_numeric(cab_df['fare_amount'], errors='coerce')
    cab_df['pickup_datetime']=pd.to_datetime(cab_df['pickup_datetime'], errors='coerce')
    cab_df.info()

    #Getting years, month, time from datetime variable
    cab_df['pickup_year']= cab_df['pickup_datetime'].dt.year
    cab_df['pickup_month']=cab_df['pickup_datetime'].dt.month
    cab_df['pickup_month_day']=cab_df['pickup_datetime'].dt.day
    cab_df['pickup_hour']=cab_df['pickup_datetime'].dt.hour
    cab_df['pickup_minute']=cab_df['pickup_datetime'].dt.minute
    cab_df['pickup_second']=cab_df['pickup_datetime'].dt.second

    #Droping pickup date time
    cab_df=cab_df.drop(['pickup_datetime'], axis=1)
    cab_df=cab_df.drop(['pickup_second'], axis=1)
    cab_df=cab_df.drop(['pickup_minute'], axis=1)
    cab_df.head()
    print("\nShape of dataframe:",cab_df.shape)

    #removing invalid values
    if("fare_amount" in cab_df.columns):
        cab_df=cab_df[cab_df["fare_amount"]>1]
    cab_df=cab_df[cab_df["pickup_longitude"]!=0]
    cab_df=cab_df[cab_df["dropoff_longitude"]!=0]
    cab_df=cab_df[cab_df["pickup_latitude"]!=0]
    cab_df=cab_df[cab_df["dropoff_latitude"]!=0]

    #Since in normal situtation, can not have passenger more than 7,
    #which include 1-driver also, so we will drop passenger count more than 6
    cab_df=cab_df[cab_df["passenger_count"]<7]
    cab_df=cab_df[cab_df["passenger_count"]>=1]
    print("\nShape of dataframe:",cab_df.shape)
    #cab_df.describe()

    #checking latitude and longitude for its valid range
    #latitude : -90 to 90
    #longtitude : -180 to 180
    (cab_df[cab_df["dropoff_latitude"]<-90] & cab_df[cab_df["dropoff_latitude"]>90]).shape # no
invalid found
    (cab_df[cab_df["pickup_latitude"]<-90] & cab_df[cab_df["pickup_latitude"]>90]).shape #single
observation found
    (cab_df[cab_df["dropoff_longitude"]<-180] & cab_df[cab_df["dropoff_longitude"]>180]).shape #
no invalid
    (cab_df[cab_df["dropoff_longitude"]<-180] & cab_df[cab_df["dropoff_longitude"]>180]).shape #
no invalid

    #dropping observation ranging outside -90 to 90 for latitude
    cab_df=cab_df[cab_df["pickup_latitude"]>-90]
    cab_df=cab_df[cab_df["pickup_latitude"]<90]
    print("\nData Cleaning done, Invalid value removed..\n")

    return cab_df



def Missing_value(cab_df):
    #checking missing value
    print(pd.DataFrame(cab_df.isna().sum(), columns={'Missing Value'}))

    #For missing value in current scenario , we can use mode/median.
    #But since single observation only have all missing value we can just drop that single observation
    cab_df=cab_df.dropna()
    print("\nShape of data after missing value removal",cab_df.shape)

    return cab_df



def find_distance(cab_df):

    #!pip install haversine
    import haversine as hs
    temp=[]
    range(cab_df.shape[0])

    for i in range(cab_df.shape[0]):

temp.append(hs.haversine((cab_df["pickup_latitude"].iloc[i],cab_df["pickup_longitude"].iloc[i]),(cab
_df["dropoff_latitude"].iloc[i],cab_df["dropoff_longitude"].iloc[i])))
    cab_df["distance"]=temp
    cab_df["distance"]=round(cab_df["distance"],2)
    cab_df["distance"].describe()
    cab_df.nlargest(10, ["distance"])

cab_df=cab_df.drop(["pickup_longitude","pickup_latitude","dropoff_longitude","dropoff_latitude"],
axis=1)
    cab_df.shape
    print("\nDistance feature created in our dataset, pickup and dropoff cordinates are removed\n")

    return cab_df
```

```
def Outlier(cab_df):
    #using box plot analysis



    #boxplot for distance
    plt.figure(figsize=(7,3))
    plt.boxplot(cab_df['distance'])
    plt.ylabel('distance')
    plt.title("Boxplot for distance")

    #boxplot for fare_amount

    if("fare_amount" in cab_df.columns):
        plt.figure(figsize=(7,3))
        plt.boxplot(cab_df['fare_amount'])
        plt.ylabel('fare_amount')
        plt.title("Boxplot for fare_amount")

    ###### Removing outlier ##############

    if("fare_amount" in cab_df.columns):
        #checking the largest top-10 fare_amount
        print("Top 10 Largest fare data are\n", cab_df["fare_amount"].nlargest(10))
        print("Top 10 Smallest fares data are\n", cab_df["fare_amount"].nsmallest(10))
        #removing the outlier fare_amount
        cab_df=cab_df[cab_df["fare_amount"]<=453]
        cab_df=cab_df[cab_df["fare_amount"]>1]

    #checking largest top-10 distance
    cab_df.nlargest(10, ["distance"])

    #removing the outlier fare_amount
    cab_df=cab_df[cab_df["distance"]<130]
    cab_df=cab_df[cab_df["distance"]!=0]

    #removing invalid passenger count
    cab_df=cab_df[cab_df["passenger_count"]!=1.3]
    print("\nOutlier are removed...")

    return cab_df

def Anova(cab_df):
    cat_variable=["passenger_count", "pickup_year", "pickup_month", "pickup_month_day",
"pickup_hour"]

    #Doing Anova Test for correlation analysis, for catagorical attributes
    for i in cat_variable:
        f, p = stats.f_oneway(cab_df[i], cab_df["fare_amount"])
        print("\nANOVA TEST: P value for variable "+str(i)+" is "+str(p))
        #print("f value for variable "+str(i)+" is "+str(f))

        #We see p-value for all variables is below 0.05, hence no variable is required to be dropped

    return cab_df

def Scaling(cab_df):

    #checking varible distribution for continous varible
    sns.distplot(cab_df['distance'],bins='auto',color='blue')
    plt.title("Variable Distribution for distance")
    plt.xlabel('distance')
    plt.show()

    if "fare_amount" in cab_df.columns:
        #checking varible distribution for continous varible
        sns.distplot(cab_df['fare_amount'],bins='auto',color='blue')
        plt.title("Variable Distribution for fare_amount")
        plt.xlabel('fare_amount')
        plt.show()



    #Removing skewness of data
    cnames=["fare_amount", "distance"]
    #using logarithm transformation for reducing skewness

    if("fare_amount" in cab_df.columns):
        cab_df['fare_amount'] = np.log1p(cab_df['fare_amount'])
    cab_df['distance'] = np.log1p(cab_df['distance'])

    #rechecking the skewness after log transformation,
    #since our skewness reduced, we do not requires scaling technique further like normalisation and
standardisation

    print("\nDistribution after Scaling..\n")

    #checking varible distribution for continous varible
    sns.distplot(cab_df['distance'],bins='auto',color='blue')
    plt.title("Variable Distribution for distance")
    plt.xlabel('distance')
    plt.show()

    if "fare_amount" in cab_df.columns:
        #checking varible distribution for continous varible
        sns.distplot(cab_df['fare_amount'],bins='auto',color='blue')
        plt.title("Variable Distribution for fare_amount")
        plt.xlabel('fare_amount')
        plt.show()
    return cab_df

def Create_dummy(cab_df):

    #Creating dummy variable
    cab = pd.get_dummies(data = cab_df, columns = cat_variable)
    return cab

#Using all fucntion defined above

cab_data=data_clean(cab_data)
cab_data=Missing_value(cab_data)
cab_data=find_distance(cab_data)
cab_data=Outlier(cab_data)
cab_data=Anova(cab_data)
cab_data=Scaling(cab_data)
#cab_data=Create_dummy(cab_data) #Adding or removing dummy variable not affecting the model
much.

#Train and test split
Y = cab_data['fare_amount']
X = cab_data.drop(['fare_amount'], axis=1)
# Using train_test_split sampling function for test and train data split
X_train, X_test, y_train, y_test = train_test_split(X,Y,test_size=0.2, random_state=4)
print("Shape of train data: ", X_train.shape)
print("Shape of test data: ", X_test.shape)

#Checking for parameter giving the best accuracy in terms R-Square value

#r2=[]
#random=[]

#for i in range(34):
```

```
#    X_train, X_test, y_train, y_test = train_test_split(X,Y,test_size=0.2, random_state=i)
#    fit_GB = GradientBoostingRegressor().fit(X_train, y_train)
#    pred_test = fit_GB.predict(X_test)
#    r2.append(r2_score(y_test,pred_test))
#    random.append(i)

#for test size 0.2, Gradeient Boosting
# Create the pandas DataFrame
#df = pd.DataFrame(r2, columns = ['r2_Score'])
#df["Random"]=random
#df.nlargest(5, ["r2_Score"])

#Linear Regression MULTIPLE

#Import libraries for LR
import statsmodels.api as sm

# Train the model using the training sets
model = sm.OLS(y_train, X_train).fit()
# predictions for train model
predictions_LR = model.predict(X_train)
# Calulating RMSE
print('Root Mean Squared Error of train:', np.sqrt(mean_squared_error(y_train,predictions_LR)))
# predictions for train model
predictions_LR = model.predict(X_test)
# Calulating RMSE
RMSE_test_LR = np.sqrt(mean_squared_error(y_test,predictions_LR))
print('Root Mean Squared Error of test:',RMSE_test_LR)
## R2 ##
R2_LR=r2_score(y_test,predictions_LR)
print("R^2 Score = "+str(R2_LR))
```

```
model.summary()

#Decision Tree Regressor #####################
fit_DT = DecisionTreeRegressor(max_depth = 2).fit(X_train,y_train)
#prediction on train data
pred_train_DT = fit_DT.predict(X_train)
#prediction on test data
pred_test_DT = fit_DT.predict(X_test)

##calculating RMSE for train data
RMSE_train_DT = np.sqrt(mean_squared_error(y_train, pred_train_DT))
##calculating RMSE for test data
RMSE_test_DT = np.sqrt(mean_squared_error(y_test, pred_test_DT))
print("Root Mean Squared Error For Training data = "+str(RMSE_train_DT))
print("Root Mean Squared Error For Test data = "+str(RMSE_test_DT))

## R^2 calculation for test data
R2_DT=r2_score(y_test, pred_test_DT)
print("R^2 Score = "+str(R2_DT))



#Random Forest
#from sklearn.ensemble import RandomForestRegressor
#from sklearn.metrics import mean_squared_error
#from sklearn.metrics import r2_score
model = RandomForestRegressor(n_estimators=100, random_state=20).fit(X_train, y_train)
y_pred = model.predict(X_train)
print('Root Mean Squared Error for train:', np.sqrt(mean_squared_error(y_train, y_pred)))
test_pred = model.predict(X_test)
RMSE_test_RF=np.sqrt(mean_squared_error(y_test,test_pred))
```

```
print('Root Mean Squared Error for test:', RMSE_test_RF)
## R2 ##
R2_RF=r2_score(y_test,test_pred)
print("R^2 Score = "+str(R2_RF))

#Gradient Boosting ###################

from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
# Building model on top of training dataset
fit_GB = GradientBoostingRegressor().fit(X_train, y_train)
# Calculating RMSE for training data to check for over fitting
pred_train = fit_GB.predict(X_train)
# Calulating RMSE
print('Root Mean Squared Error of train:', np.sqrt(mean_squared_error(y_train,pred_train)))
# Calculating RMSE for test data to check accuracy
pred_test = fit_GB.predict(X_test)
# Calulating RMSE
RMSE_test_GB=np.sqrt(mean_squared_error(y_test,pred_test))
print('Root Mean Squared Error of test:', RMSE_test_GB)
R2_GB=r2_score(y_test,pred_test)
print("R^2 Score for test = "+str(R2_GB))

#Out of all below model, we see Gradient Bossting giving better R2-Square and lesser RMSE.
#Hence we select Gradient Boosting model.

measure_table = {'Model': ['Linear Regression','Decision Tree','Random Forest','Gradient Boosting'],
    'RMSE': [RMSE_test_LR,RMSE_test_DT,RMSE_test_RF,RMSE_test_GB],
    'R-Square': [R2_LR,R2_DT,R2_RF,R2_GB],
    }
```

```
print("\n Summary for all model")

print("----------------------------------------")

print(pd.DataFrame(measure_table, columns = ['Model', 'RMSE', 'R-Square']))


test_data = pd.read_csv("C:/Users/SAURABH SHRIVASTAVA/OneDrive/Data Scientist/Edwisor
project 2/test.csv")


test_data=data_clean(test_data)

test_data=Missing_value(test_data)

test_data=find_distance(test_data)

test_data=Outlier(test_data)

test_data=Scaling(test_data)

#test_data=Create_dummy(test_data)



#Selecting Gradient Boosting model, to predict given test data

pred_new = fit_GB.predict(test_data)


test_data["Predicted Fare"]=pred_new


test_data.to_csv("C:/Users/SAURABH SHRIVASTAVA/OneDrive/Data Scientist/Edwisor project
2/predicted_test.csv")

test_data.head(5)


############################### Bar plot of fare amount vs passenger count ####

############################### Bar plot of fare amount vs Pickup year ########

# Trip distance vs fare amount

#plt.subplots(111, figsize=(12,6))

#fig.figure()


plt.figure(figsize=(10,5))

sns.countplot(cab_data['pickup_year'])

#Cab Count for Year 2015 is less as compared to other previous year.
```

```
plt.figure(figsize=(10,5))

sns.countplot(cab_data['pickup_month'])

#Cab counts are less from 7th to 12th Month i.e. July to December Month


plt.figure(figsize=(20,5))

sns.countplot(cab_data['pickup_month_day'])

#Last Month days cab taken are less


plt.figure(figsize=(10,5))

sns.countplot(cab_data['pickup_hour'])

#Cab requirement is in peak from 6P.M to 11P.M, whereas cab count are less in morning i.e 2A.M to
6A.M


#Plot between passenger count and fare amount

plt.figure(figsize=(10,5))

sns.barplot(x='passenger_count', y='fare_amount', data=cab_data, estimator=sum).set_title("PLot")


#Cab fare are highest for single passengers


plt.figure(figsize=(12,7))

plt.bar(cab_data["passenger_count"], cab_data['fare_amount'])

plt.xlabel("Passenger Count")
```

R-Code

```
library("readxl")
library("DMwR")
library("ggplot2")
library("randomForest")
library("corrgram")
library("rpart")
library("plyr")
library("dplyr")
library("caret")
library("vctrs")
library(CSTools)
library(Metrics)
library(e1071)
library(caret)
library(mlr)
library(dummies)

# Set working directory
setwd("C:/Users/SAURABH SHRIVASTAVA/OneDrive/Data Scientist/Edwisor project 2")

#loading data
cab = read.csv('train_cab.csv')

cab = as.data.frame(cab)
head(cab)

#Splitting Date and time
cab$Date <- as.Date(cab$pickup_datetime)
cab$Year <- substr(as.character(cab$Date),1,4)
cab$Month <- substr(as.character(cab$Date),6,7)
cab$Date <- substr(as.character(cab$Date),9,10)
```

```
cab$Time <- substr(as.factor(cab$pickup_datetime),12,13)

#droping pickup_datetime as we have different columns
cab = subset(cab, select = -c(pickup_datetime))

dim(cab)
###########dropping invalid value

#removing passenger more than 6 and less than 1 number
cab=subset(cab, passenger_count < 7)
cab=subset(cab, passenger_count >= 1)

#checking Datatype
cab$fare_amount <- as.numeric(cab$fare_amount)
cab=subset(cab, fare_amount > 1)



unique(cab$passenger_count) # found 1.3 is invalid, removing those
cab=subset(cab, passenger_count != 1.3)

dim(cab)
#removing longitude, latitude which is zero
cab=subset(cab, pickup_longitude != 0)
cab=subset(cab, pickup_latitude != 0)
cab=subset(cab, dropoff_longitude != 0)
cab=subset(cab, dropoff_latitude != 0)

#Longitude should be between -180 and 180
cab=subset(cab, pickup_longitude > -180)
cab=subset(cab, pickup_longitude < 180)
cab=subset(cab, dropoff_longitude > -180)
cab=subset(cab, dropoff_longitude < 180)
```

```
#Latitude should be between -90 and 90
cab=subset(cab, pickup_latitude > -90)
cab=subset(cab, pickup_latitude < 90)
cab=subset(cab, dropoff_latitude > -90)
cab=subset(cab, dropoff_latitude < 90)



dim(cab)
############Missing Value Analysis##########

#checking whether missing value present
sum(is.na(cab))
#Since the missing value is just 3 nos., we can simply drop those observation
cab = na.omit(cab)

dim(cab)
##############################################

# Calculates the geodesic distance between two points specified by
# radian latitude/longitude using the Haversine formula
lat1 = cab['pickup_latitude']
lat2 = cab['dropoff_latitude']
long1 = cab['pickup_longitude']
long2 = cab['dropoff_longitude']

##### Function to calculate distance ######
gcd_hf <- function(long1, lat1, long2, lat2) {
  R <- 6371.145 # Earth mean radius [km]
  delta.long <- (long2 - long1)
  delta.lat <- (lat2 - lat1)
  a <- sin(delta.lat/2)^2 + cos(lat1) * cos(lat2) * sin(delta.long/2)^2
  c <- 2 * atan2(sqrt(a),sqrt(1-a))
  d = R * c
  return(d) # Distance in km
}
#Running the function for all rows in dataframe
for (i in 1:nrow(cab)){
  cab$distance[i]= gcd_hf(cab$pickup_longitude[i], cab$pickup_latitude[i], cab$dropoff_longitude[i],
                       cab$dropoff_latitude[i])
}

cab$distance=round(cab$distance, digits = 2)
############################################################################################
########

###########Outlier Analysis##################

###for outlier for distance########
boxplot(cab$distance)
tail(sort(cab$distance, decreasing = FALSE),1000)#found value above 500 looks outlier, increasing
suddenly
tail(sort(cab$distance, decreasing = TRUE),1000)#found value less 1

#tail(sort(unique(cab$distance)),2000)
cab=subset(cab, distance < 500)
cab=subset(cab, distance > 1)

dim(cab)
###for outlier in fare_amount######
boxplot(cab$fare_amount)
tail(sort(cab$fare_amount, decreasing = FALSE),1000) #nothing seems outlier
tail(sort(cab$fare_amount, decreasing = TRUE),1000) #nothing seems outlier
#cab=subset(cab, fare_amount < 453)
#cab=subset(cab, fare_amount > 1)
```

```
dim(cab)

##########Feature Selection

#After distance conversion from longitute and latitude, we can drop the columns for latitude/longitude

cab = subset(cab, select = -c(pickup_latitude,dropoff_latitude,pickup_longitude,dropoff_longitude))

corrgram(cab[,],order = FALSE,upper.panel = panel.pie,text.panel = panel.txt,main= 'Correlation Plot')

###########Feature Scaling ###########

hist(cab$distance)

hist(cab$fare_amount)

#plot shows data has high skewness, we will try to reduce the skewness by log transformation, if

#skewness reduced, we will not go with Scaling technique like normalisation or standardisation

#Log transformation

cab$distance <- log10(cab$distance)

#cab$fare_amount <- log10(cab$fare_amount)

#Normalization

#cab[,"distance"] = (cab[,"distance"] - min(cab[,"distance"]))/(max(cab[,"distance"])-min(cab[,"distance"]))

cab[,"fare_amount"] = (cab[,"fare_amount"] - min(cab[,"fare_amount"]))/(max(cab[,"fare_amount"])-min(cab[,"fare_amount"]))

#############Creating Dummy variable for categorical variable
```

```
cata = c('passenger_count','Date','Year','Month','Time')

#cab1 = dummy.data.frame(cab, cata)
#################Train and Test Split

# #Divide data into trainset and testset using stratified sampling method
#install.packages('caret')

set.seed(101)
split_index = createDataPartition(cab$fare_amount, p = 0.8, list = FALSE)
trainset = cab[split_index,]
testset = cab[-split_index,]

dim(trainset)
######################Model Builing

##linear Regression

#Develop Model on training data
fit_LR = lm(fare_amount ~ ., data = trainset)
#Lets predict for test data
pred_LR_test = predict(fit_LR, testset)
# For test data
print(postResample(pred = pred_LR_test, obs = testset$fare_amount))



## Random Forest

#Develop Model on training data
fit_RF = randomForest(fare_amount~., data = trainset, ntree=300)
#Lets predict for test data
pred_RF_test = predict(fit_RF, testset)
```

```
# For test data
print(postResample(pred = pred_RF_test, obs = testset$fare_amount))


###--------------------------------------------XGBoost------------------------------------------#
### for xgboost it is required to make date variable as factor.
trainset$Date <- as.numeric(trainset$Date)
trainset$Year <- as.numeric(trainset$Year)
trainset$Month <- as.numeric(trainset$Month)
trainset$Time <- as.numeric(trainset$Time)
trainset$passenger_count <- as.numeric(trainset$passenger_count)
trainset$distance <- as.numeric(trainset$distance)
#trainset$fare_amount <- as.numeric(trainset$fare_amount)


library(gbm)
#Develop Model on training data
fit_XGB = gbm(fare_amount~., data = trainset, n.trees = 300, interaction.depth = 2)
#Lets predict for test data
pred_XGB_test = predict(fit_XGB, testset, n.trees = 300)
# For test data
print(postResample(pred = pred_XGB_test, obs = testset$fare_amount))
```

End of Project Report