Project Name:
Cab Fare Prediction

Prepared by:
Kriti Upadhyay

Submission Date: 27.12.2019

CONTENT TABLE:

PART 1: IMPLEMENTATION BY R

Introduction

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.

Data Set provided:

1) train_cab.zip
2) test.zip

Data set contains below number of attributes:

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

Train Dataset Summary:

```
 $ distance       : num  1.03 8.45 1.39 2.8 2 ...
> summary( Cab_Train_Data)
  fare_amount    passenger_count pickup_weekday  pickup_month  pickup_year
 Min.   : 1.14   1:11070         1:2079         05     :1476   2009:2429
 1st Qu.: 6.00   2: 2286         2:2224         03     :1473   2010:2432
 Median : 8.50   3:  663         3:2256         06     :1461   2011:2413
 Mean   : 9.40   4:  320         4:2269         01     :1435   2012:2500
 3rd Qu.:12.00   5: 1025         5:2323         04     :1406   2013:2474
 Max.   :22.10   6:  296         6:2397         02     :1297   2014:2279
                                 7:2112         (Other):7112   2015:1133
  pickup_hour         distance
 18      :  970   Min.   :   0.000
 19      :  967   1st Qu.:   1.257
 20      :  936   Median :   2.172
 21      :  894   Mean   :   4.037
 22      :  884   3rd Qu.:   3.905
 12      :  780   Max.   :5420.989
 (Other):10229
>
```

Fig 1: Summary of train dataset

Workflow of the Project:

1. Prepare Problem (Load libraries, Load dataset)
2. Summarize Data (Descriptive statistics, Data visualizations)
3. Prepare Data (Data Cleaning, Feature Selection, Data Transforms)
4. Evaluate Algorithms (Split-out validation dataset, Testing and evaluating metric, Checking Algorithms, Comparing Algorithms)
5. Improve Accuracy (Algorithm Tuning, Ensembles)
6. Selecting Model (Predictions on validation dataset, create standalone model on entire training dataset, Predicting cab fare amount for test dataset.

PART 1: IMPLEMENTATION BY R

Step-1: Data Loading

Data Set is taken from the link provided and then loaded into the R environment for performing Data pre-processing techniques which are the necessary steps in the data science to organize data into proper format before feeding it to the model, because every Model accepts the data in a specific format form only.

Following commands used to perform above task:

```
#Loading datasets
 Cab_Train_Data = read.csv("train_cab.csv", header = T, na.strings = c(" ", "", "NA"))
 Cab_Test_Data = read.csv("test.csv")
 test_pickup_datetime = Cab_Test_Data["pickup_datetime"]


#Viewing Structure of both train and test data
 str( Cab_Train_Data)
 str( Cab_Test_Data)
```

Step-2 : Exploratory Data Analysis

Exploratory Data Analysis is a process of analyzing data in detail and Removing values which are not within desired range depending upon basic understanding of dataset. In our case of dataset, we explored data upon following understanding that are as follows:

1. Fare_amount variable has a negative value, which doesn't make sense because price amount cannot be negative and also cannot be 0. So, we will remove the observations having negative fare amount.

2. 20 observations of passenger_count variable is consistently above from 6,7,8,9,10,11 passenger_counts, we need to remove these fields as cab contain maximum number of passenger 6 and also Removing 58 observations having passenger_count = 0

3. Latitudes range should be from -90 to 90 and Longitudes range should be from -180 to 180.Removing which does not satisfy these ranges

Following codes used to perform above task:

##############EXPLORATORY DATA ANALYSIS ###################

```
#Changing the data types of variables
Cab_Train_Data$fare_amount=
as.numeric(as.character(Cab_Train_Data$fare_amount))
Cab_Train_Data$passenger_count=round(Cab_Train_Data$passenger_count)

#Removing values which are not within desired range depending upon basic
understanding of dataset.

#1. Fare amount has a negative value, which doesn't make sense. A price amount
cannot be negative and also cannot be 0. So, we will remove these fields.
#Rows with negative Fare amount
Cab_Train_Data[which(Cab_Train_Data$fare_amount < 1 ),]

#Count of rows having negative fare amount
nrow(Cab_Train_Data[which(Cab_Train_Data$fare_amount < 1 ),])

#Removing rows from data containing negative fare amount
Cab_Train_Data= Cab_Train_Data[-which(Cab_Train_Data$fare_amount < 1 ),]


#2. Passenger_count variable
for (i in seq(4,11,by=1))
{
print(paste('passenger_count above '
,i,nrow(Cab_Train_Data[which(Cab_Train_Data$passenger_count > i ),])))
}

#So 20 observations of passenger_count is consistenly above from 6,7,8,9,10,11
passenger_counts, checking them.
Cab_Train_Data[which(Cab_Train_Data$passenger_count > 6 ),]

#We need to see if there are any passenger_count = 0
Cab_Train_Data[which(Cab_Train_Data$passenger_count <1 ),]

#Getting number of observation having passenger_count==0
nrow(Cab_Train_Data[which(Cab_Train_Data$passenger_count <1 ),])
```

#Removing 58 observations having passenger_count==0 and 20 observation which are above 6 because a cab cannot hold these number of passengers.

```
Cab_Train_Data = Cab_Train_Data[-which(Cab_Train_Data$passenger_count < 1 ),]
Cab_Train_Data = Cab_Train_Data[-which(Cab_Train_Data$passenger_count > 6),]
```

#3. Latitudes range from -90 to 90. Longitudes range from -180 to 180. Removing which does not satisfy these ranges

#Getting rows of Latitudes range from -90 to 90. Longitudes range from -180 to 180.

```
print(paste('pickup_longitude above 180=',nrow(Cab_Train_Data[which(Cab_Train_Data$pickup_longitude >180 ),])))
print(paste('pickup_longitude above -180=',nrow(Cab_Train_Data[which(Cab_Train_Data$pickup_longitude < -180 ),])))
print(paste('pickup_latitude above 90=',nrow(Cab_Train_Data[which(Cab_Train_Data$pickup_latitude > 90 ),])))
print(paste('pickup_latitude above -90=',nrow(Cab_Train_Data[which(Cab_Train_Data$pickup_latitude < -90 ),])))
print(paste('dropoff_longitude above 180=',nrow(Cab_Train_Data[which(Cab_Train_Data$dropoff_longitude > 180 ),])))
print(paste('dropoff_longitude above -180=',nrow(Cab_Train_Data[which(Cab_Train_Data$dropoff_longitude < -180 ),])))
print(paste('dropoff_latitude above -90=',nrow(Cab_Train_Data[which(Cab_Train_Data$dropoff_latitude < -90 ),])))
print(paste('dropoff_latitude above 90=',nrow(Cab_Train_Data[which(Cab_Train_Data$dropoff_latitude > 90 ),])))
```

#Removing pickup_latitude above 90

```
Cab_Train_Data= Cab_Train_Data[-which(Cab_Train_Data$pickup_latitude > 90),]
```

#Getting rows having longitude and latitude values which are equal to 0
```
nrow(Cab_Train_Data[which(Cab_Train_Data$pickup_longitude == 0 ),])
nrow(Cab_Train_Data[which(Cab_Train_Data$pickup_latitude == 0 ),])
nrow(Cab_Train_Data[which(Cab_Train_Data$dropoff_longitude == 0 ),])
nrow(Cab_Train_Data[which(Cab_Train_Data$pickup_latitude == 0 ),])
```

#Removing longitude and latitude values which are equal to 0
```
Cab_Train_Data = Cab_Train_Data[-which(Cab_Train_Data$pickup_longitude == 0),]
Cab_Train_Data = Cab_Train_Data[-which(Cab_Train_Data$dropoff_longitude == 0),]
```

Step-3: Missing Value Analysis

After loading the data, the first step performed is missing value analysis where we compute the missing values or information present in the data set. Here the missing value analysis is computed by using the method of KNN Imputation, because it computes the missing values by using Euclidean Distance formula and this method gives accurate results mostly than compared to mean median method.

Following codes used to perform above task:

############### MISSING VALUE ANALYSIS ###################

#Missing value detection in each variable of the dataset by using apply function that uses Sum(is.na()) function as an argument and will return total count of the missing values present in each variable of the dataset.
```
missing_val = data.frame(apply(Cab_Train_Data,2,function(x){sum(is.na(x))}))
missing_val
```

#Converting row names into column and renaming variable name
```
missing_val$Columns = row.names(missing_val)
row.names(missing_val) = NULL
names(missing_val)[1] =  "Missing_percentage"
```

#Calculating Missing_percentage
```
missing_val$Missing_percentage = (missing_val$Missing_percentage/nrow(Cab_Train_Data)) * 100
```

#Sorting variable
  missing_val = missing_val[order(-missing_val$Missing_percentage),]

#Rearranging columns
  missing_val = missing_val[,c(2,1)]
  missing_val

#Getting unqiue passenger count for train and test data
  unique(Cab_Train_Data$passenger_count)
  unique(Cab_Test_Data$passenger_count)

#Converting datatype of passenger_count variable as factor for both test and train data
 Cab_Train_Data[,'passenger_count'] = factor(Cab_Train_Data[,'passenger_count'], labels=(1:6))
 Cab_Test_Data[,'passenger_count'] = factor(Cab_Test_Data[,'passenger_count'], labels=(1:6))

#Applying different methods for computing missing values
  #Mode Method
    #Cab_Train_Data$passenger_count[1000]
    #Cab_Train_Data$passenger_count[1000] = NA
    #getmode = function(v)
    #{
    #uniqv = unique(v)
    #uniqv[which.max(tabulate(match(v, uniqv)))]
    #}
    #getmode(Cab_Train_Data$passenger_count)
      #For Passenger_count variable:
      #Actual value = 1
      #Mode = 1
      #We can't use mode method because data is more biased towards passenger_count=1

   # Mean Method
    #Cab_Train_Data$fare_amount[1000]
    #Cab_Train_Data$fare_amount[1000]= NA

    #mean(Cab_Train_Data$fare_amount, na.rm = T)

```
# Median Method
  #Cab_Train_Data$fare_amount[1000]
  #Cab_Train_Data$fare_amount[1000]= NA

  #median(train$fare_amount, na.rm = T)

# kNN Imputation
  #Cab_Train_Data$fare_amount[1000]
  #Cab_Train_Data$fare_amount[1000]=NA

  #Cab_Train_Data = knnImputation(Cab_Train_Data, k = 181)

    # For fare_amount variable:
    # Actual value = 18.1,
    # Mean = 15.117,
    # Median = 8.5,
    # KNN = 18.28
```

#Since the missing values computed by different methods but only KNN imputation method would be good choice as the value computed by it is more closer to the actual values.

```
#Computing missing values using KNN imputation method
  Cab_Train_Data = knnImputation(Cab_Train_Data, k = 181)

#Check for missing values
  sum(is.na(Cab_Train_Data))

 #Getting structure and summary of the train data
  str(Cab_Train_Data)
  summary(Cab_Train_Data)
```

Step-4: Outliers Detection & Removal
Outliers Detection:

Once, the missing values are computed, the next step is to detect and remove outliers present in the data set. These outliers are nothing but the extreme values present in the data set.

In this project, missing values are detected using Box-plot method because it gives the graphical representation of the presence of the outliers in the data set and completely distinguish the outliers present in the lower and upper fence of the box plot. Using replace outlier by NA method in our dataset as deleting outliers can cause information loss which might be important for the computation of the model.

Following codes used to perform above task:

```
################## OUTLIER ANALYSIS ###################

#Selecting only numeric variables
  Numeric_index= sapply(Cab_Train_Data,is.numeric)
  Numeric_data=Cab_Train_Data[,Numeric_index]

#Variables names containing numeric data
  Cnames=colnames(Numeric_data)
  Cnames

#Performing Outlier Analysis only on Fare_amount because other variables will be required to perform feature engineering.
  # Boxplot for fare_amount
    pl1 = ggplot(Cab_Train_Data,aes(x = factor(passenger_count),y = fare_amount))
    pl1 + geom_boxplot(outlier.colour="red", fill = "grey" ,outlier.shape=18,outlier.size=1, notch=FALSE)+ylim(0,100)
```

Detection of the outliers present in the fare_amount with respect to the passenger_count variable using Box plots are shown below:

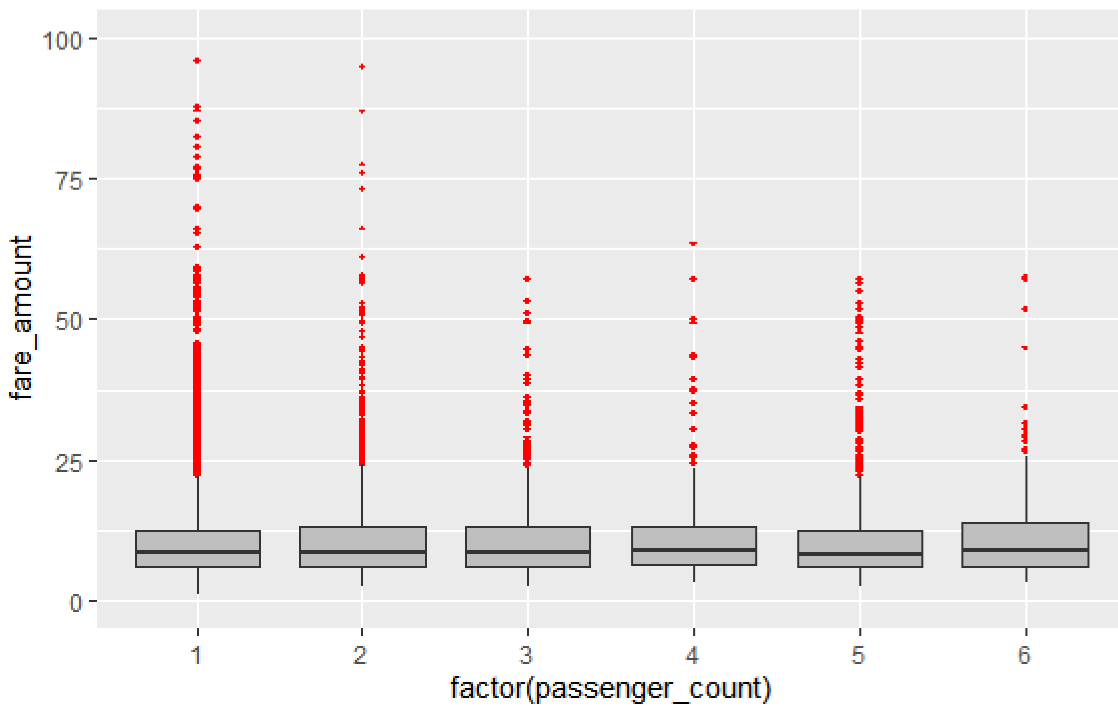

Fig 2: fare_amount variable with outliers

Outlier Removal:

Outlier replaced by NA method: This method is chosen because we cannot afford deletion of the outliers as sometimes outliers contains important information. After replacing outliers with NA, the NA values are further computed by the method KNN imputation. Replacing outliers with NA present in the dataset as this is the best method to deal with the NA as taught in lecture.

Following codes used to perform above task:

```
# Replace all outliers with NA and impute
vals=Cab_Train_Data[,"fare_amount"]%in%
boxplot.stats(Cab_Train_Data[,"fare_amount"])$out
Cab_Train_Data[which(vals),"fare_amount"] = NA


#lets check the NA's
sum(is.na(Cab_Train_Data$fare_amount))
```

#Imputing with KNN
Cab_Train_Data = knnImputation(Cab_Train_Data,k=3)

#Checking the missing values
sum(is.na(Cab_Train_Data$fare_amount))
str(Cab_Train_Data)

Boxplot for outliers present in the fare_amount with respect to the passenger_count variable after outlier removal are shown below:

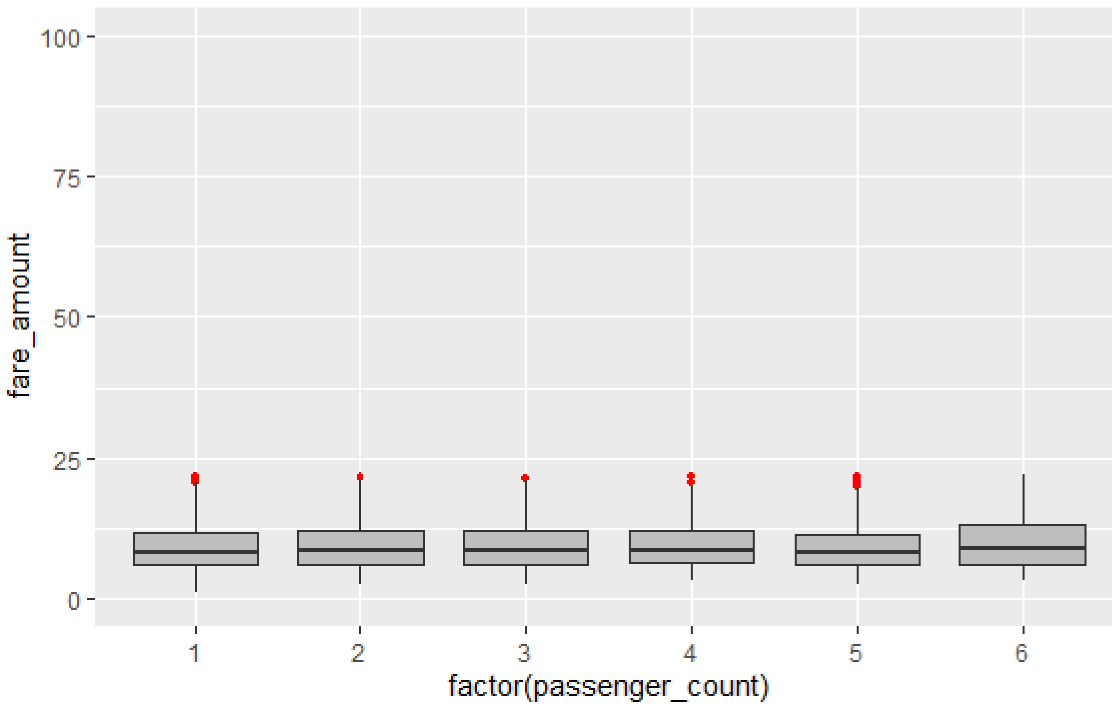

Fig 3: fare_amount variable without outliers

Step 5: Feature Engineering
Below features are derived from the existing variable pickup_datetime for efficient understanding of data and for good modelling purpose too:
1. pickup_date
2. pickup_weekday
3. pickup_month
4. pickup_year
5. pickup_hour
6. distance

Following codes used to perform above task:

```
################## FEATURE ENGINEERING ###################

#1. Feature Engineering for timestamp variable
#we will derive new features from pickup_datetime variable and new features will be year,month,day_of_week,hour
#Convert pickup_datetime from factor to date time
   Cab_Train_Data$pickup_date =
as.Date(as.character(Cab_Train_Data$pickup_datetime))
   Cab_Train_Data$pickup_weekday =
as.factor(format(Cab_Train_Data$pickup_date,"%u"))
   Cab_Train_Data$pickup_month =
as.factor(format(Cab_Train_Data$pickup_date,"%m"))
   Cab_Train_Data$pickup_year =
as.factor(format(Cab_Train_Data$pickup_date,"%Y"))
   pickup_time = strptime(Cab_Train_Data$pickup_datetime,"%Y-%m-%d
%H:%M:%S")
   Cab_Train_Data$pickup_hour = as.factor(format(pickup_time,"%H"))

  #Adding similar features to test dataset
   Cab_Test_Data$pickup_date =
as.Date(as.character(Cab_Test_Data$pickup_datetime))
   Cab_Test_Data$pickup_weekday =
as.factor(format(Cab_Test_Data$pickup_date,"%u"))
   Cab_Test_Data$pickup_month = as.factor(format(
Cab_Test_Data$pickup_date,"%m"))
   Cab_Test_Data$pickup_year = as.factor(format(
Cab_Test_Data$pickup_date,"%Y"))
```

```
pickup_time = strptime( Cab_Test_Data$pickup_datetime,"%Y-%m-%d %H:%M:%S")
Cab_Test_Data$pickup_hour = as.factor(format(pickup_time,"%H"))
```

#Check for NA values

```
missing_val = data.frame(apply(Cab_Train_Data,2,function(x){sum(is.na(x))}))
missing_val
```

#One NA was present in variable pickup_datetime which created NA in above new feature engineered variables.

#Removing that 1 row of NA's

```
Cab_Train_Data= na.omit(Cab_Train_Data)
```

#Displaying Cab_Train_Data

```
Cab_Train_Data
```

#Removing the variables which were used to engineer new variables from both train and test dataset

```
Cab_Train_Data = subset(Cab_Train_Data,select = -c(pickup_datetime,pickup_date))
Cab_Test_Data= subset( Cab_Test_Data,select = -c(pickup_datetime,pickup_date))
```

#2. Calculate the distance travelled using longitude and latitude

#function to convert degree into radian

```
deg_to_rad = function(deg)
  {
   (deg * pi) / 180
  }
```

#Using haversine formula to calculate distance for both train and test data

```
haversine = function(long1,lat1,long2,lat2)
  {
   #long1rad = deg_to_rad(long1)
   phi1 = deg_to_rad(lat1)
   #long2rad = deg_to_rad(long2)
   phi2 = deg_to_rad(lat2)
   delphi = deg_to_rad(lat2 - lat1)
   dellamda = deg_to_rad(long2 - long1)
```

```
    a = sin(delphi/2) * sin(delphi/2) + cos(phi1) * cos(phi2) *
    sin(dellamda/2) * sin(dellamda/2)

    c = 2 * atan2(sqrt(a),sqrt(1-a))
    R = 6371e3
    #1000 is used to convert to meters
    R * c / 1000
  }

Cab_Train_Data$distance = haversine( Cab_Train_Data$pickup_longitude, Cab_Train_Data$pickup_latitude, Cab_Train_Data$dropoff_longitude, Cab_Train_Data$dropoff_latitude)
  Cab_Test_Data$distance = haversine(Cab_Test_Data$pickup_longitude,Cab_Test_Data$pickup_latitude,Cab_Test_Data$dropoff_longitude,Cab_Test_Data$dropoff_latitude)

#Removing the variables which were used to engineer new variables from both train and test dataset
  Cab_Train_Data = subset( Cab_Train_Data,select = -c(pickup_longitude,pickup_latitude,dropoff_longitude,dropoff_latitude))
  Cab_Test_Data = subset(Cab_Test_Data,select = -c(pickup_longitude,pickup_latitude,dropoff_longitude,dropoff_latitude))

#Getting summary and structure of the train data
 str( Cab_Train_Data)
 summary( Cab_Train_Data)
```

Step-6: Feature selection

This stage involves the process of reducing variables on the basis of correlation present in the variables of the dataset. Correlation plot is being used to analyze the correlation among the variables and reduce the dimensionality on the basis of correlation between variables. In this step we would allow only to pass relevant features to further steps. We remove irrelevant features from the dataset. We do this by some statistical techniques, like we look for features which will not be helpful in predicting the target variables. In this dataset we have to predict the fare_amount. Further below are some types of test involved for feature selection:

1. **Correlation analysis** – This requires only numerical variables. Therefore, we will filter out only numerical variables and feed it to correlation analysis. We do this by plotting correlation plot for all numerical variables. There should be no correlation between independent variables but there should be high correlation between independent variable and dependent variable. So, we plot the correlation plot.

Following codes used to perform above task:

```
################## FEATURE SELECTION ###################

################## CORRELATION PLOT ####################

#Selecting only numeric variables
 Numeric_index= sapply(Cab_Train_Data,is.numeric)
 Numeric_data=Cab_Train_Data[,Numeric_index]

#Variables names containing numeric data
 Cnames=colnames(Numeric_data)
 Cnames

#Correlation plot
corrgram(Cab_Train_Data[,Cnames],order=F,upper.panel=panel.pie,text.panel=panel.txt,main="CORRELATION PLOT")
```

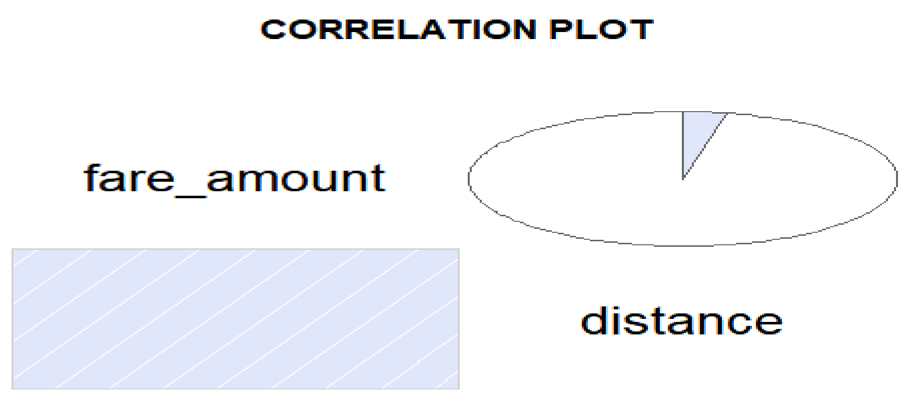

Fig 4: Correlation plot

From above correlation plot we see that:
• 'fare_amount' and 'distance' are very highly correlated with each other.
• As fare_amount is the target variable and 'distance' is independent variable we will keep 'distance' because it will help to explain variation in fare_amount.

2. Analysis of Variance (Anova) Test – It is carried out to compare between each group in a categorical variable. ANOVA only lets us know the means for different groups are same or not. It doesn't help us identify which mean is different.
Hypothesis testing:
- **Null Hypothesis**: mean of all categories in a variable are same.
- **Alternate Hypothesis**: mean of at least one category in a variable is different.

If p-value is less than 0.05 then we reject the null hypothesis and if p-value is greater than 0.05 then we accept the null hypothesis.

Following codes used to perform above task:

```
################## ANOVA TEST ###################

#ANOVA for categorical variables with target numeric variable
Anova_results = aov(fare_amount ~ passenger_count + pickup_hour + pickup_weekday + pickup_month + pickup_year,data = Cab_Train_Data)
```

#Summary of anova result

```
summary(Anova_results)
```

#pickup_weekday has p value greater than 0.05, so rejecting this variable

```
Cab_Train_Data = subset(Cab_Train_Data,select=-pickup_weekday)
```

#Also remove that variable from from test dataset

```
Cab_Test_Data = subset(Cab_Test_Data,select=-pickup_weekday)
```

Below is the Anova analysis table for each categorical variable:

```
>   summary(Anova_results)
                   Df Sum Sq Mean Sq F value   Pr(>F)    
passenger_count     5    252    50.4   2.651   0.0211 *  
pickup_hour        23   2552   111.0   5.841  < 2e-16 ***
pickup_weekday      6     61    10.2   0.535   0.7817    
pickup_month       11    976    88.7   4.669 3.67e-07 ***
pickup_year         6   7111  1185.1  62.372  < 2e-16 ***
Residuals       15608 296562    19.0                     
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
> 
```

Fig 5: ANOVA test on Variables

pickup_weekday variable has p value greater than 0.05, so rejecting this variable

Step-7: Feature Scaling
Data Scaling methods are used when we want our variables in data to scaled on common ground. It is performed only on continuous variables.

• **Normalization**: Normalization refer to the dividing of a vector by its length. normalization normalizes the data in the range of 0 to 1. It is generally used when we are planning to use distance method for our model development purpose such as KNN. Normalizing the data improves convergence of such algorithms. Normalization of data scales the data to a very small interval, where outliers can be loosed.

• **Standardization**: Standardization refers to the subtraction of mean from individual point and then dividing by its SD. Z is negative when the raw score is below the mean and Z is positive when above mean. When the data is distributed normally you should go for standardization.

Linear Models assume that the data you are feeding are related in a linear fashion, or can be measured with a linear distance metric. Also, our independent numerical variable 'distance' is not distributed normally so we had chosen normalization over standardization. High variance will affect the accuracy of the model. So, we want to normalise that variance. It is performed only on Continuous variables.

Following codes used to perform above task:

```
################## FEATURE SCALING ###################

#Normality check
  qqnorm(Cab_Train_Data$fare_amount)
  histogram(Cab_Train_Data$fare_amount)
  qqnorm(Cab_Train_Data$distance)
  histogram(Cab_Train_Data$distance)
```

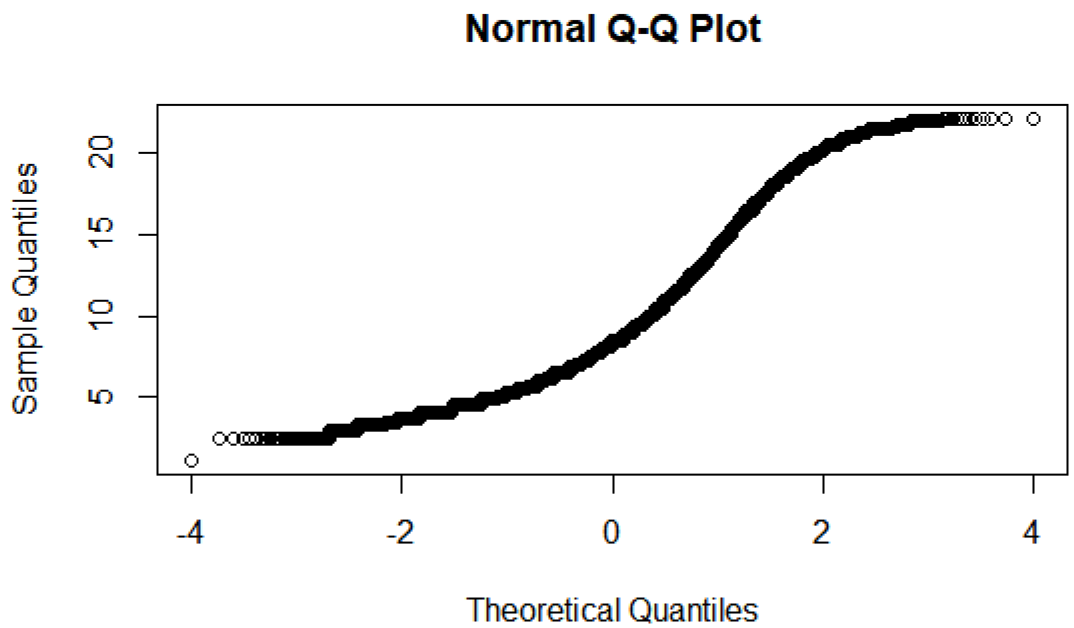

Fig 6: Normality check on fare_amount variable

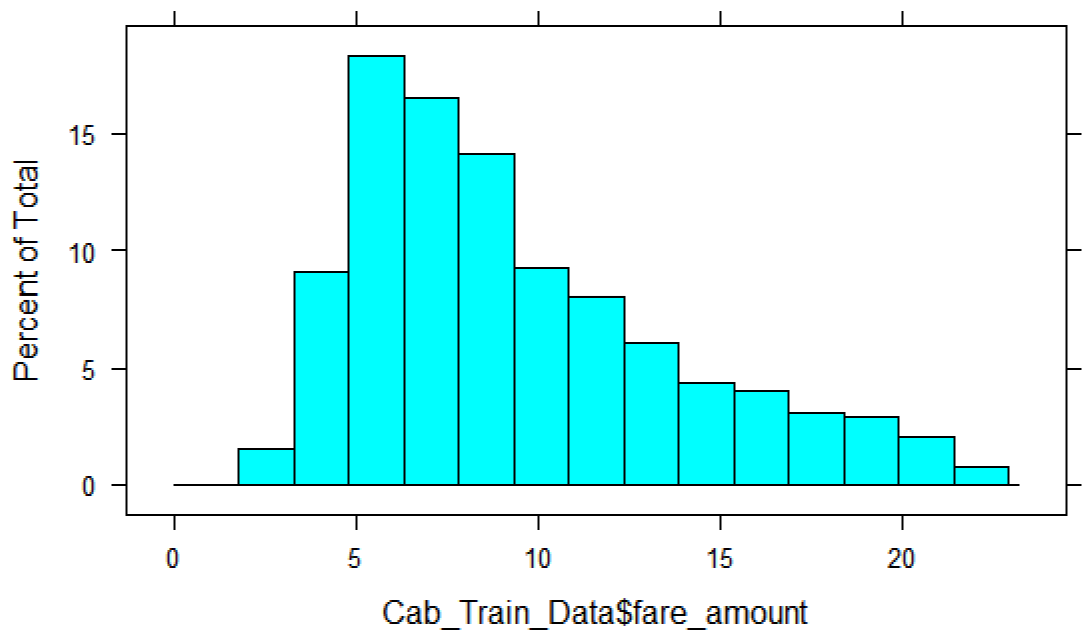

Fig 7: Normality check on fare_amount variable using histogram

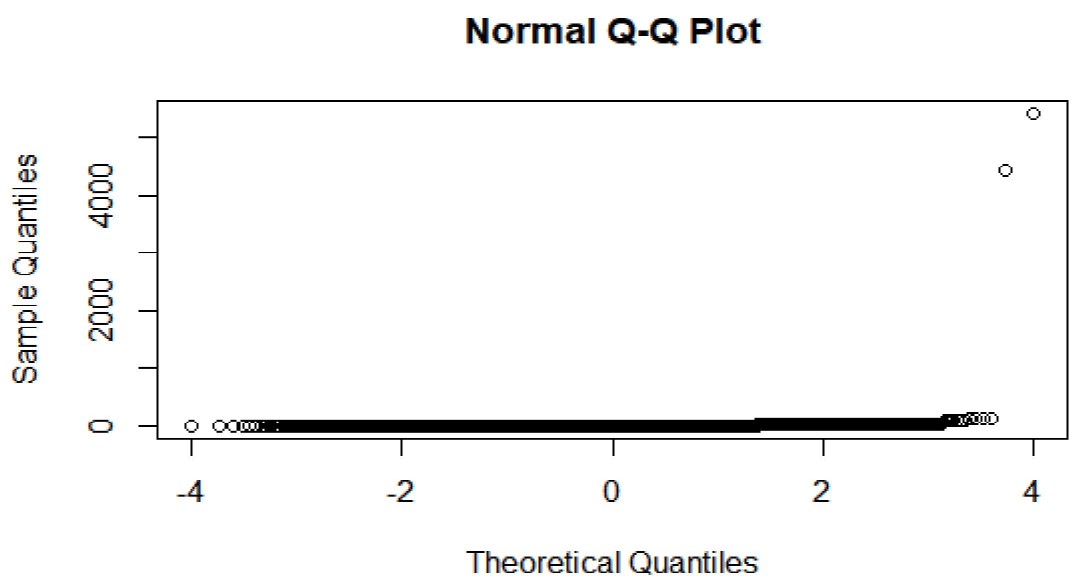

Fig 8: Normality check on distance variable

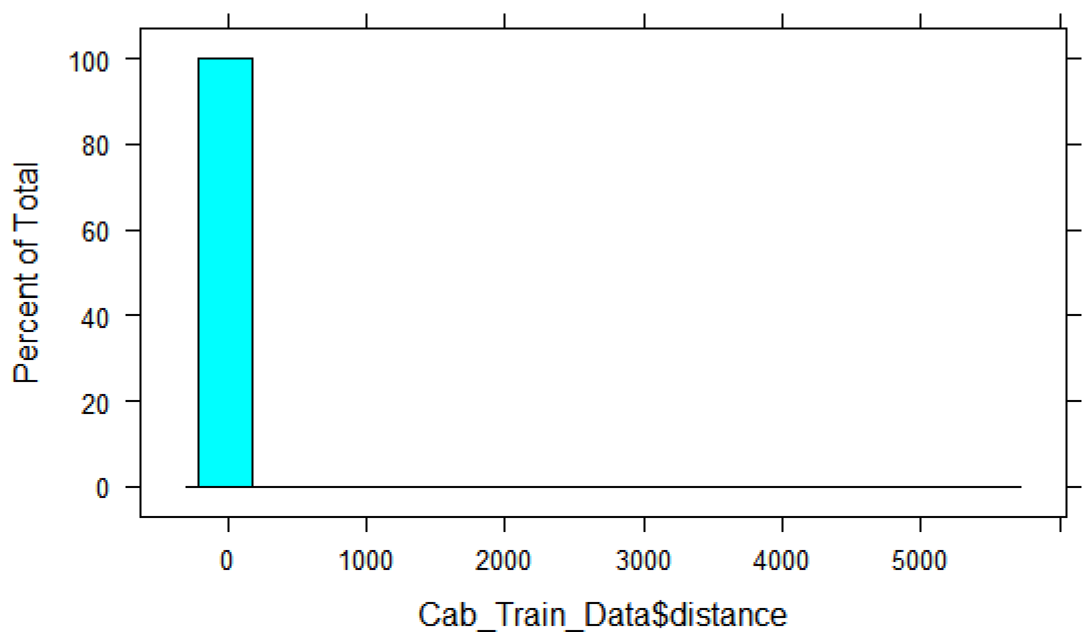

Fig 9: Normality check on distance variable using histogram

On observing plots, it is clear that the data of the distance variable is not uniformly distributed. Hence in this applying normalization for proper scaling of the distance variable.

```
#Normalisation (distance variable is not uniformly distributed)
Cab_Train_Data[,'distance'] = (Cab_Train_Data[,'distance'] - min(Cab_Train_Data[,'distance']))/
(max(Cab_Train_Data[,'distance'] - min(Cab_Train_Data[,'distance'])))
```

Step-8: Multicollinearity

In regression, "multicollinearity" refers to predictors that are correlated with other predictors. Multicollinearity occurs when your model includes multiple factors that are correlated not just to your response variable, but also to each other. Multicollinearity increases the standard errors of the coefficients. It Increases standard errors in turn means that coefficients for some independent variables may be found not to be significantly different from 0.

In other words, by overinflating the standard errors, multicollinearity makes some variables statistically insignificant when they should be significant. Without multicollinearity (and thus, with lower standard errors), those coefficients might be significant.

VIF is always greater or equal to 1.
if VIF is 1: Not correlated to any of the variables.
if VIF is between 1-5: Moderately correlated.
if VIF is above 5: Highly correlated.
If there are multiple variables with VIF greater than 5, only remove the variable with the highest VIF and if the VIF goes above 10, you can assume that the regression coefficients are poorly estimated due to multicollinearity.

Following codes used to perform above task:

```
#check for multicollinearity

  #vif(Cab_Train_Data[,2:6])
  #vifcor(Cab_Train_Data[,-1], th = 0.9)
```

Below is the table for VIF analysis for each independent variable:

```
> vif(Cab_Train_Data[,2:6])
        Variables     VIF
1 passenger_count      NA
2    pickup_month      NA
3     pickup_year      NA
4     pickup_hour      NA
5        distance 1.02389
```

Fig 10: VIF analysis

Step-9: Splitting train and Validation Dataset

We have used **createDataPartition()** method to divide whole Dataset into train and validation dataset. 20% is in validation dataset and 80% is in training data. We will test the performance of model on validation dataset. The model which performs best will be chosen to perform on test dataset provided along with original train dataset.

- X_train y_train--are train subset.
- X_test y_test--are validation subset.

Following codes used to perform above task:

```
################## SPLITTING DATA ###################

set.seed(1000)
#Splitting 80% of Cab-Train_Data in train_data and 20% in Validation Dataset(test_data)
train_index = createDataPartition(Cab_Train_Data$fare_amount,p=0.80,list = FALSE)
 train_data = Cab_Train_Data[train_index,]
 test_data = Cab_Train_Data[-train_index,]
```

Step- 10: Model Development

Our problem statement wants us to predict the fare_amount. This is a Regression problem. So, we are going to build regression models on training data and predict it on test data. In this project I have built models using four Algorithms:

- Linear Regression
- Decision Tree
- Random Forest
- Xgboost Regression

We will evaluate performance on validation dataset which was generated using Sampling. We will deal with specific error metrics like –

Regression metrics for our Models:

- MAE (Mean Absolute Error)
- MAPE (Mean Absolute Percentage Error)
- MSE (Mean square Error)
- RMSE (Root Mean Square Error)

################## LINEAR REGRESSION ###################

```
#Running Regression model
 lm_model = lm(fare_amount ~.,data=train_data)

#Summary of the model
 summary(lm_model)
 str(train_data)

#Predicting test_data using predict() method
 lm_predictions = predict(lm_model,test_data[,2:6])

#plotting regression model on the basis of test_data
 qplot(x = test_data[,1], y = lm_predictions, data = test_data, color = I("blue"), geom
= "point")
```

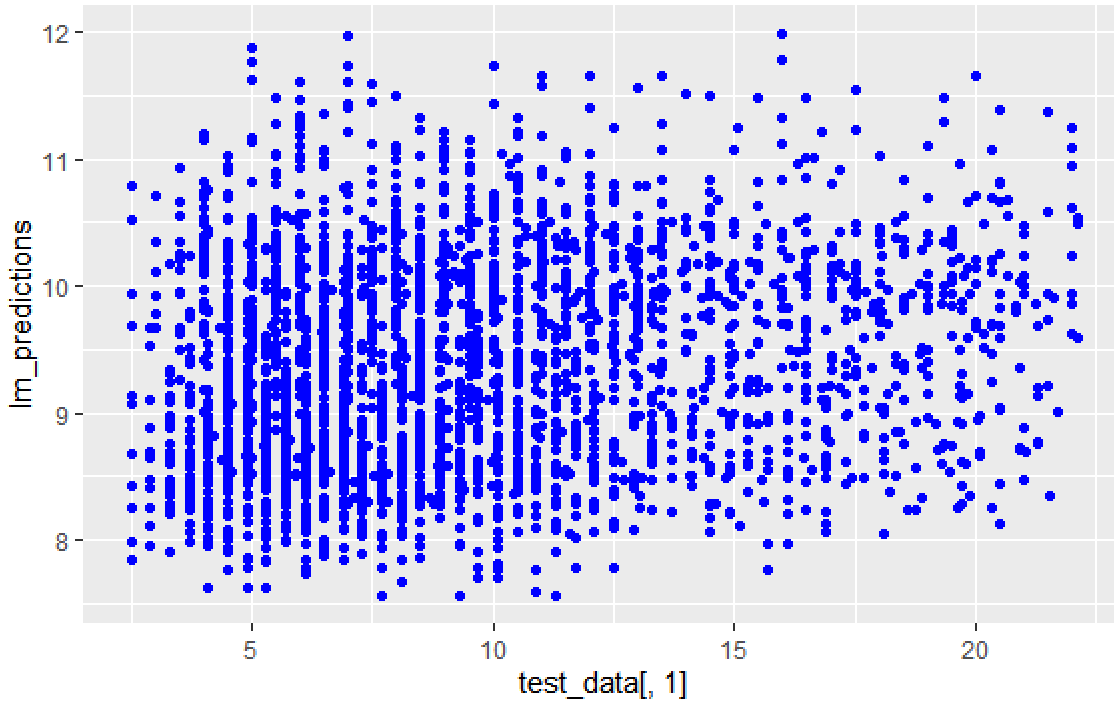

Fig 11: Linear Regression predicted plot

#Evaluation of the linear regression model on the basis of test_data

```
regr.eval(test_data[,1],lm_predictions)
```

 # mae mse rmse mape
 # 3.5120358 19.0085569 4.3598804 0.4544157

#Error rate =0.45
#Accuracy=55%

################### DECISION TREE ####################

#Running Decision tree model

```
Dt_model = rpart(fare_amount ~ ., data = train_data, method = "anova")
```

#Summary of the model

```
summary(Dt_model)
```

#Predicting test cases using predict() method

```
predictions_DT = predict(Dt_model, test_data[,2:6])
```

#plotting Decision tree model on the basis predicted test_data
qplot(x = test_data[,1], y = predictions_DT, data = test_data, color = I("blue"), geom= "point")

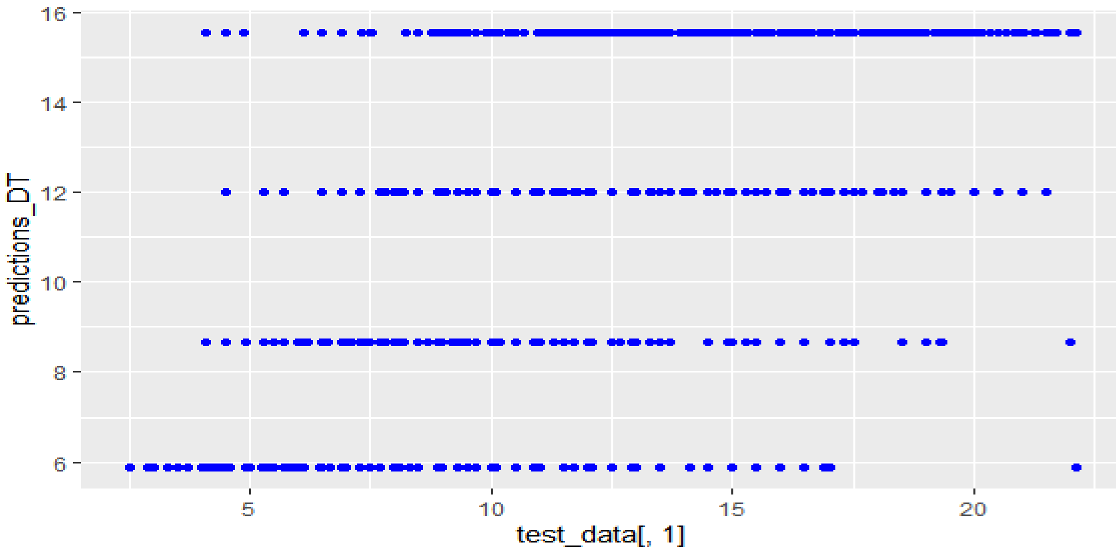

Fig 12: Decision tree predicted plot

#Evaluation of Decision tree model on the basis of test_data
regr.eval(test_data[,1],predictions_DT)

 # mae mse rmse mape
 # 1.9674997 7.0442171 2.6540944 0.2317727

#Error rate =0.23
#Accuracy=77%

################### RANDOM FOREST ###################

#Running Random Forest model
rf_model = randomForest(fare_amount ~.,data=train_data)

#Summary of the model
summary(rf_model)

#Predicting test cases using predict() method
rf_predictions = predict(rf_model,test_data[,2:6])

#plotting Random Forest model on the basis predicted test_data

```
qplot(x = test_data[,1], y = rf_predictions, data = test_data, color = I("blue"), geom
= "point")
```

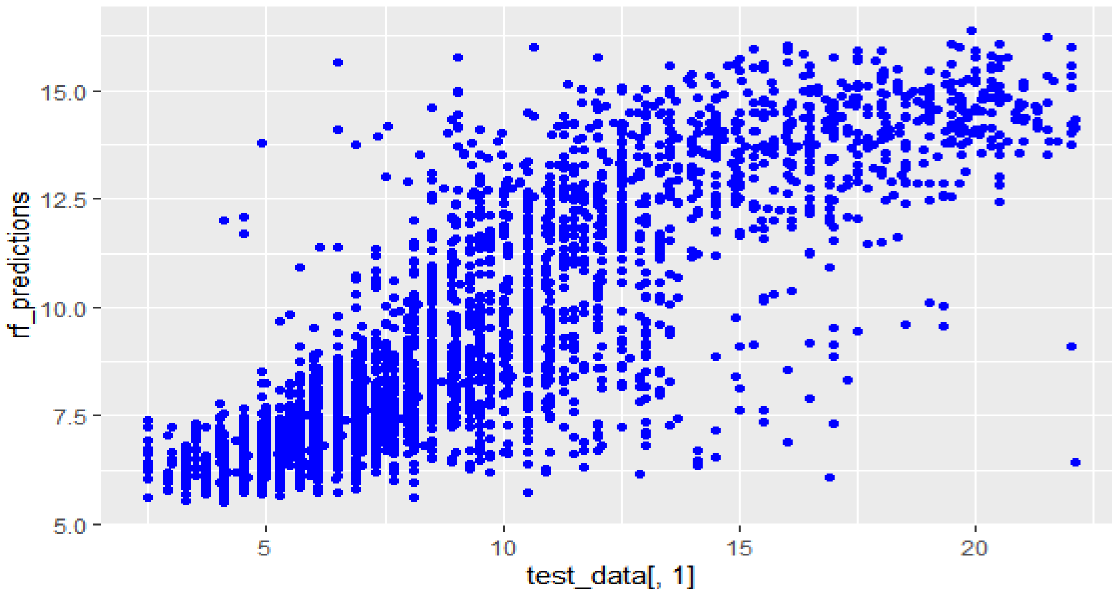

Fig 13: Random Forest predicted plot

#Evaluation of Random Forest model on the basis of test_data

```
regr.eval(test_data[,1],rf_predictions)
```

 # mae mse rmse mape
 # 1.8838831 6.2805142 2.5060954 0.2312715

#Error rate =0.23
#Accuracy=77%

################### XGBOOST MODEL ###################

#Running xgboost model

```
train_data_matrix = as.matrix(sapply(train_data[-1],as.numeric))
test_data_matrix = as.matrix(sapply(test_data[-1],as.numeric))
xgboost_model =xgboost(data = train_data_matrix,label =
train_data$fare_amount,nrounds = 15,verbose = FALSE)
```

#Summary of the model

```
summary(xgboost_model)
```

#Predicting test cases using predict() method

```
xgb_predictions = predict(xgboost_model,test_data_matrix)
```

#plotting xgboost model on the basis predicted test_data

```
qplot(x = test_data[,1], y = xgb_predictions, data = test_data, color = I("blue"),
geom = "point")
```

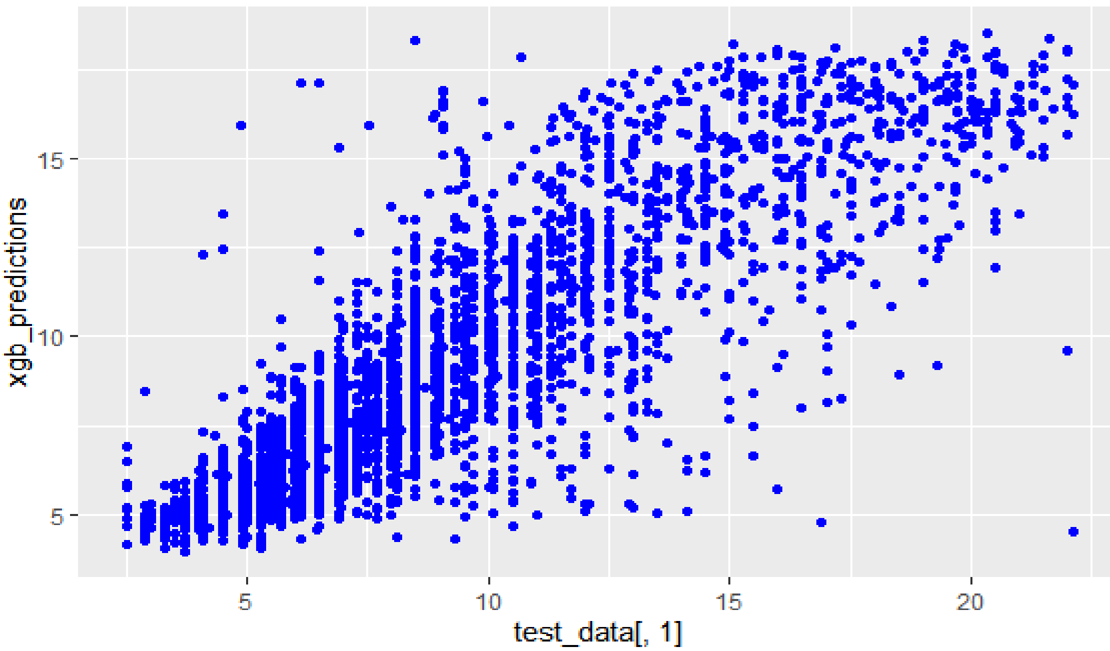

Fig 14: XGBOOST model predicted plot

#Evaluation of Random Forest model on the basis of test_data

```
regr.eval(test_data[,1],xgb_predictions)
```

 # mae mse rmse mape
 # 1.6206727 5.2563892 2.2926817 0.1835422

#Error rate =0.18
#Accuracy=82%

Step 11: Model Selection

After developing model, we will evaluate the performance of the model by considering generated Regression metrics for our Models in development stage:

- MAE (Mean Absolute Error)
- MAPE (Mean Absolute Percentage Error)
- MSE (Mean square Error)
- RMSE (Root Mean Square Error)

The lesser the error rate better will be the performance and the accuracy of the model. On comparing these regression metrics of different models applied on the dataset we will select that model who has low error rate and better accuracy compared to other models for cab fare prediction.

Models	MAE	MAPE	MSE	RMSE	Error Rate	Accuracy
Linear Regression	3.51	0.45	19.00	4.35	0.45	55%
Decision Tree	1.96	0.23	7.04	2.65	0.23	77%
Random Forest	1.88	0.23	6.28	2.50	0.23	77%
Xgboost model	1.62	0.18	5.2	2.2	0.18	82%

On Comparing different regression metrics Xgboost model found to be better model than other models and have good performance and accuracy as well as low error rate.So, selecting XGBOOST model for cab fare prediction .

Step-12:Conclusion/Result:

```
################## CAB FARE PREDICTION ####################

#Training model on whole training Dataset and saving model using xgboost model has it has more accuracy when compared to other models
  train_data_matrix1 = as.matrix(sapply(Cab_Train_Data[-1],as.numeric))
  test_data_matrix1 = as.matrix(sapply(Cab_Test_Data,as.numeric))

#Running xgboost model on enire train data
 xgboost_model1 = xgboost(data = train_data_matrix1,label = Cab_Train_Data$fare_amount,nrounds = 15,verbose = FALSE)
```

#Saving the trained model
```
saveRDS(xgboost_model1, "./Trained_Xgboost_model_using_R.rds")
```

#loading the saved model
```
Final_Trained_model= readRDS("./Trained_Xgboost_model_using_R.rds")
print(Final_Trained_model)
```

#Predicting fare_amount on test dataset
```
xgb = predict(Final_Trained_model,test_data_matrix1)
xgb_pred = data.frame(test_pickup_datetime,"predictions" = xgb)
```

#Writing the predicted fare_amount in disk in .csv format
```
write.csv(xgb_pred,"Cab_Fare_Prediction_By_R.csv",row.names = FALSE)
```

Note: The predicted fare_amount for the test data is present in the file "Cab_Fare_Prediction_By_R.csv" which is attached with the report.

PART 2: IMPLEMENTATION BY PYTHON

Step-1: Data Loading
Data Set is taken from the link provided and then loaded into the python environment for performing Data pre-processing techniques which are the necessary steps in the data science to organize data into proper format before feeding it to the model, because every Model accepts the data in a specific format form only.

Following commands used to perform above task:

```
# loading the required libraries
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
from geopy.distance import geodesic
from scipy.stats import chi2_contingency
import statsmodels.api as sm
from statsmodels.formula.api import ols
from patsy import dmatrices
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
import xgboost as xgb

# Importing dataset
Cab_Train_Data =
pd.read_csv('train_cab.csv',dtype={'fare_amount':np.float64},na_values={'fare_am
ount':'430-'})
Cab_Test_Data = pd.read_csv('test.csv')

#Summary of the train dataset
Cab_Train_Data.describe()
```

memory usage: 878.8 KB

```
In [6]: #Summary of the train dataset
        Cab_Train_Data.describe()
```

Out[6]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
count	16042.000000	16067.000000	16067.000000	16067.000000	16067.000000	16012.000000
mean	15.015004	-72.462787	39.914725	-72.462328	39.897906	2.625070
std	430.460945	10.578384	6.826587	10.575062	6.187087	60.844122
min	-3.000000	-74.438233	-74.006893	-74.429332	-74.006377	0.000000
25%	6.000000	-73.992156	40.734927	-73.991182	40.734651	1.000000
50%	8.500000	-73.981698	40.752603	-73.980172	40.753567	1.000000
75%	12.500000	-73.966838	40.767381	-73.963643	40.768013	2.000000
max	54343.000000	40.766125	401.083332	40.802437	41.366138	5345.000000

Fig 1: Summary of train dataset

#Summary of the test dataset
Cab_Test_Data.describe()

```
In [9]: #Summary of the test dataset
        Cab_Test_Data.describe()
```

Out[9]:

	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

Fig 2: Summary of test dataset

```
######################## DATA VISUALIZATION ###############

# setting up the sns for plots
sns.set(style='darkgrid',palette='Set1')

# Plotting histogram for variables of the train dataset
plt.figure(figsize=(20,20))
plt.subplot(321)
_ = sns.distplot(Cab_Train_Data['fare_amount'],bins=50)
plt.subplot(322)
_ = sns.distplot(Cab_Train_Data['pickup_longitude'],bins=50)
plt.subplot(323)
_ = sns.distplot(Cab_Train_Data['pickup_latitude'],bins=50)
plt.subplot(324)
_ = sns.distplot(Cab_Train_Data['dropoff_longitude'],bins=50)
plt.subplot(325)
_ = sns.distplot(Cab_Train_Data['dropoff_latitude'],bins=50)

#Saving Plots
plt.savefig('histogrambypython.png')

#Displaying plots
plt.show()
```

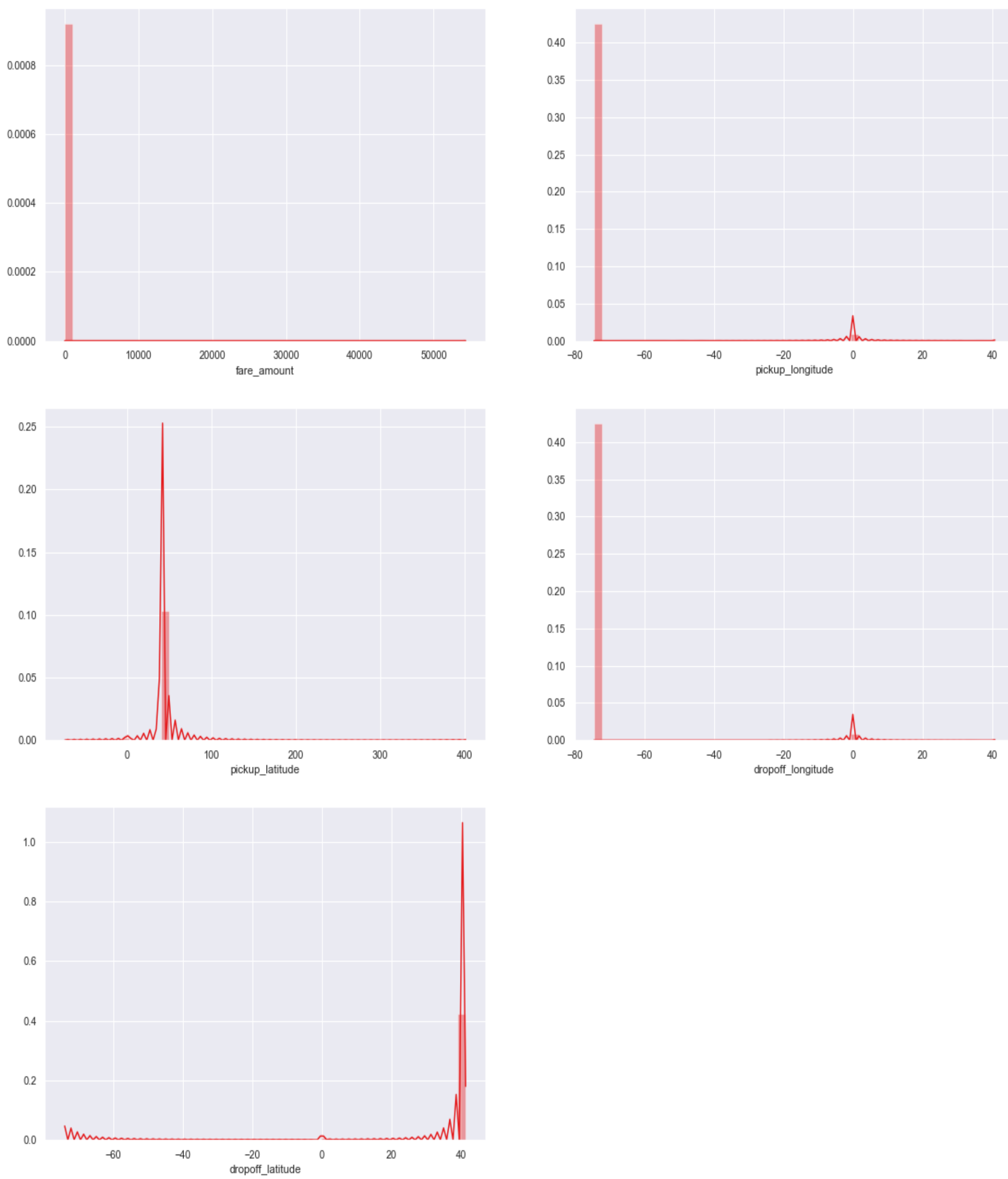

Fig 3: Data visualization

Step-2: Exploratory Data Analysis
Exploratory Data Analysis is a process of analyzing data in detail and Removing values which are not within desired range depending upon basic understanding of dataset. In our case of dataset, we explored data upon following understanding that are as follows:

4. Fare_amount variable has a negative value, which doesn't make sense because price amount cannot be negative and also cannot be 0. So, we will remove the observations having negative fare amount.

5. 20 observations of passenger_count variable is consistently above from 6,7,8,9,10,11 passenger_counts, we need to remove these fields as cab contain maximum number of passenger 6 and also Removing 58 observations having passenger_count = 0

6. Latitudes range should be from -90 to 90 and Longitudes range should be from -180 to 180.Removing which does not satisfy these ranges

Following codes used to perform above task:

```
##############EXPLORATORY DATA ANALYSIS ###################

#Removing values which are not within desired range(outlier) depending upon basic understanding of dataset.

#1. Fare amount has a negative value, which do not make sense. A price amount cannot be negative and also cannot be 0. So, we will remove these fields.

# Finding total number of observations have fare amount negative
sum(Cab_Train_Data['fare_amount']<1)

#Getting records having fare amount in negatives
Cab_Train_Data[Cab_Train_Data['fare_amount']<1]

#Removing records having fare amount in negatives
Cab_Train_Data=
Cab_Train_Data.drop(Cab_Train_Data[Cab_Train_Data['fare_amount']<1].index, axis=0)
Cab_Train_Data.loc[Cab_Train_Data['fare_amount'] < 1,'fare_amount'] = np.nan
```

#Checking for negative value in fare amount variable

```
sum(Cab_Train_Data['fare_amount']<1)
```

#2. Passenger_count variable needs to convert into a categorical variable because passenger_count is not a continuous variable.
passenger_count cannot take continuous values also they are limited in number if it’s a cab that is maximum number of passengers can sit in a cab is 6.

#Fetching records having passenger count greater than 4

```
for i in range(4,11):
   print('passenger_count above' +str(i)+'={}'.format(sum(Cab_Train_Data['passenger_count']>i)))
```

Output:

```
passenger_count above4=1367
passenger_count above5=322
passenger_count above6=20
passenger_count above7=20
passenger_count above8=20
passenger_count above9=20
passenger_count above10=20
```

#20 observations of passenger_count is consistenly above from 6,7,8,9,10 passenger_counts, let's check them.

```
Cab_Train_Data[Cab_Train_Data['passenger_count']>6]
```

#Checking for any passenger_count<1

```
Cab_Train_Data[Cab_Train_Data['passenger_count']<1]
```

#Finding total number of records having passenger_count<1

```
len(Cab_Train_Data[Cab_Train_Data['passenger_count']<1])
```

#passenger_count variable conains value equal to 0 but test data does not contain passenger_count=0. So, we will remove those 0 values.

#passenger_count variable conains value equal to 0 but test data does not contain passenger_count=0. So, we will remove those 0 values.
#Also dropping 20 observations which are above 6 because a cab cannot hold number of passengers greater than 6 .

```
Cab_Train_Data = Cab_Train_Data.drop(Cab_Train_Data[Cab_Train_Data['passenger_count']>6].index, axis=0)
```

```
Cab_Train_Data = Cab_Train_Data.drop(Cab_Train_Data[Cab_Train_Data['passenger_count']<1].index, axis=0)

#Check for passenger_count>6
sum(Cab_Train_Data['passenger_count']>6)
#3.Latitudes range should be  from -90 to 90 and Longitudes range should be from -180 to 180. Removing which does not satisfy these ranges

#Getting data for checking ranges
print('pickup_longitude above 180={}'.format(sum(Cab_Train_Data['pickup_longitude']>180)))
print('pickup_longitude below -180={}'.format(sum(Cab_Train_Data['pickup_longitude']<-180)))
print('pickup_latitude above 90={}'.format(sum(Cab_Train_Data['pickup_latitude']>90)))
print('pickup_latitude below -90={}'.format(sum(Cab_Train_Data['pickup_latitude']<-90)))
print('dropoff_longitude above 180={}'.format(sum(Cab_Train_Data['dropoff_longitude']>180)))
print('dropoff_longitude below -180={}'.format(sum(Cab_Train_Data['dropoff_longitude']<-180)))
print('dropoff_latitude below -90={}'.format(sum(Cab_Train_Data['dropoff_latitude']<-90)))
print('dropoff_latitude above 90={}'.format(sum(Cab_Train_Data['dropoff_latitude']>90)))
```

Output:

```
pickup_longitude above 180=0
pickup_longitude below -180=0
pickup_latitude above 90=1
pickup_latitude below -90=0
dropoff_longitude above 180=0
dropoff_longitude below -180=0
dropoff_latitude below -90=0
dropoff_latitude above 90=0
```

```
#Only one outlier which is in variable pickup_latitude.So we will remove it with nan.
#Checking for any values equal to 0.
for i in ['pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitude']:
```

```
 print(i,'equal to 0={}'.format(sum(Cab_Train_Data[i]==0)))

#Removing values =0 with NA
Cab_Train_Data=
Cab_Train_Data.drop(Cab_Train_Data[Cab_Train_Data['pickup_latitude']>90].index, axis=0)
for i in ['pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitude']:
    Cab_Train_Data =
Cab_Train_Data.drop(Cab_Train_Data[Cab_Train_Data[i]==0].index, axis=0)

for i in ['pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitude']:
    Cab_Train_Data.loc[Cab_Train_Data[i]==0,i] = np.nan
Cab_Train_Data.loc[Cab_Train_Data['pickup_latitude']>90,'pickup_latitude'] =
np.nan
```

Step-3: Missing Value Analysis

After loading the data, the first step performed is missing value analysis where we compute the missing values or information present in the data set. Here the missing value analysis is computed by using the mean method, because it computes the missing values closest to the actual value when compared to the other methods like mean and mode .Since KNN imputation method is not working in my python environment because it is no supported by my version of python also tried a lot to import it many times, So used median method.

In this step we look for missing values in the dataset like empty row column cell which was left after removing special characters and punctuation marks. Some missing values are in form of NA. missing values left behind after outlier analysis; missing values can be in any form. Unfortunately, in this dataset we have found some missing values. Therefore, we will do some missing value analysis. Before imputed we selected random row no-1000 and made it NA, so that we will compare original value with imputed value and choose best method which will impute value closer to actual value.

Following codes used to perform above task:

```
############### MISSING VALUE ANALYSIS ###################

#Create dataframe with missing percentage
missing_val = pd.DataFrame(Cab_Train_Data.isnull().sum())
```

#Reset index
```
missing_val = missing_val.reset_index()
missing_val
```

Output:

	index	0
0	fare_amount	22
1	pickup_datetime	1
2	pickup_longitude	0
3	pickup_latitude	0
4	dropoff_longitude	0
5	dropoff_latitude	0
6	passenger_count	55

we can see there are some missing values in the data.
#We will impute missing values for fare_amount,passenger_count variables & will drop that 1 row which has missing value in pickup_datetime.

#Rename variable
```
missing_val = missing_val.rename(columns = {'index': 'Variables', 0: 'Missing_percentage'})
missing_val
```

#Calculate percentage
```
missing_val['Missing_percentage'] = (missing_val['Missing_percentage']/len(Cab_Train_Data))*100
```

#descending order
```
missing_val = missing_val.sort_values('Missing_percentage', ascending = False).reset_index(drop = True)
missing_val
```

Output:

	Variables	Missing_percentage
0	passenger_count	0.351191
1	fare_amount	0.140476
2	pickup_datetime	0.006385
3	pickup_longitude	0.000000
4	pickup_latitude	0.000000
5	dropoff_longitude	0.000000
6	dropoff_latitude	0.000000

#Imputing missing values of the variable passenger_count and fare_amount variable using mean and median method
#Not using KNN method for computation because unable to install fancyimpute package. I tried a lot even perform many times of installing and uninstalling of the python
#Not using Mode method for passenger_count variable because it was showing biasing towards passenger_count =1

Choosing a random value to replace it as NA
Cab_Train_Data['fare_amount'].loc[1000]

Replacing 7.0 with NA
#Cab_Train_Data['fare_amount'].loc[1000] = np.nan
#Cab_Train_Data['fare_amount'].loc[1000]

#Imputing by mean method
#print('Value imputed by mean:{}'.format(Cab_Train_Data['fare_amount'].fillna(Cab_Train_Data['fare_amount'].mean()).loc[1000]))

Imputing by median method
#print('Value imputed by median:{}'.format(Cab_Train_Data['fare_amount'].fillna(Cab_Train_Data['fare_amount'].median()).loc[1000]))

#Missing value computed by different methods:

 # Actual value = 7.0
 # Mean = 15.117
 # Median = 8.5

#we will separate pickup_datetime into a different dataframe and then merge with train in feature engineering step.
pickup_datetime=pd.DataFrame(Cab_Train_Data['pickup_datetime'])

#dropping pickup_datetime variable from train dataset
Cab_Train_Data=Cab_Train_Data.drop('pickup_datetime',axis=1)

#Since value computed by median method is closer to the actual value when compared to mean method so will be using median method to compute missing values present in the dataset.
Cab_Train_Data["fare_amount"]=Cab_Train_Data["fare_amount"].fillna(Cab_Train_Data["fare_amount"].median())
Cab_Train_Data["passenger_count"]=Cab_Train_Data["passenger_count"].fillna(Cab_Train_Data["passenger_count"].median())

#Checking the number of missing values in the variables after computing missing value.
Miss_value=pd.DataFrame(Cab_Train_Data.isnull().sum())
Miss_value

Output:

	0
fare_amount	0
pickup_longitude	0
pickup_latitude	0
dropoff_longitude	0
dropoff_latitude	0
passenger_count	0

Step-4: Outliers Detection & Removal
Outliers Detection:

Once, the missing values are computed, the next step is to detect and remove outliers present in the data set. These outliers are nothing but the extreme values present in the data set.

In this project, missing values are detected using Box-plot method because it gives the graphical representation of the presence of the outliers in the data set and completely distinguish the outliers present in the lower and upper fence of the box plot. Using replace outlier by NA method in out dataset as deleting outliers can cause information loss which might be important for the computation of the model.

Following codes used to perform above task:

```
################## OUTLIER ANALYSIS ###################

#Outlier anaysis on fare_amount variable
plt.figure(figsize=(20,5))
plt.xlim(0,100)
sns.boxplot(x=Cab_Train_Data['fare_amount'],data=Cab_Train_Data,orient='h')
plt.title('Boxplot of fare_amount')
plt.savefig('bp_of_fare_amount_python.png')
plt.show()
```

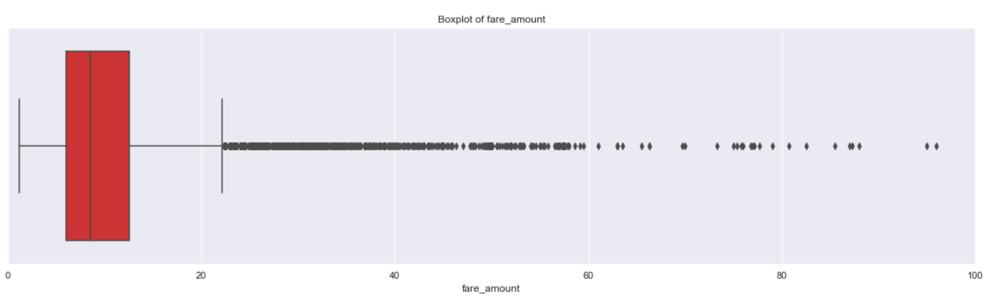

Fig 4: Boxplot for fare_amount variable

```
#Bivariate Boxplots (Boxplot for Numerical Variable Vs Categorical Variable)
plt.figure(figsize=(20,10))
plt.xlim(0,100)
_ =
sns.boxplot(x=Cab_Train_Data['fare_amount'],y=Cab_Train_Data['passenger_cou
nt'],data=Cab_Train_Data,orient='h')
```

```
plt.title('Boxplot of fare_amount w.r.t passenger_count')
plt.savefig('Boxplot_of_fare_amount_w.r.t_passenger_count_python.png')
plt.show()
```

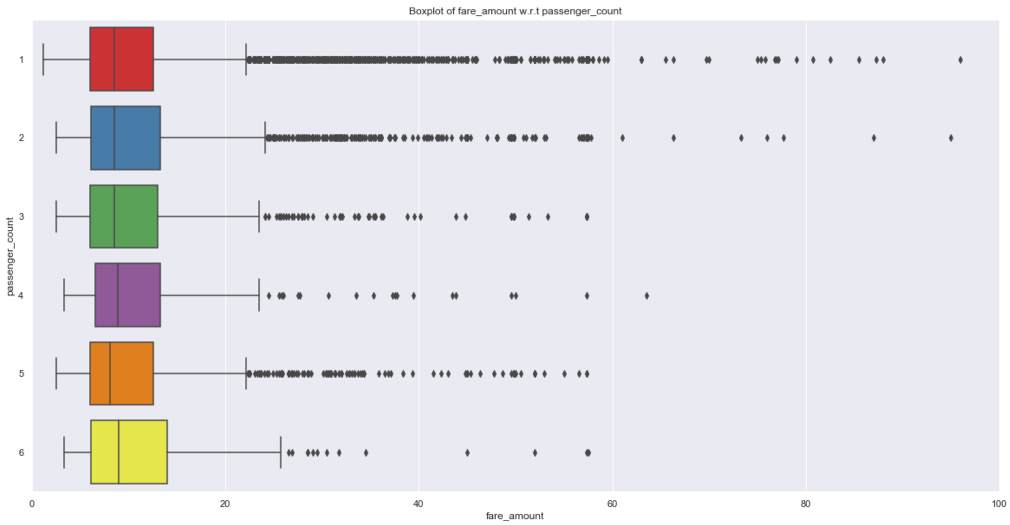

Fig 5: Boxplot of fare_amount variable with respect to passenger_count variable

Outlier Removal:

Outlier replaced by NA method: This method is chosen because we cannot afford deletion of the outliers as sometimes outliers contains important information. After replacing outliers with NA, the NA values are further computed by the median method. Replacing outliers with NA present in the dataset as this is the best method to deal with the NA as taught in lecture.

Following codes used to perform above task:

```
#Calculating outlier and replacing them with NA

def outlier_treatment(col):

    #Extract quartiles
    q75, q25 = np.percentile(Cab_Train_Data[col], [75 ,25])
    print(q75,q25)
```

```
    #Calculate IQR
    iqr = q75 - q25
    #Calculate inner and outer fence
    minimum = q25 - (iqr*1.5)
    maximum = q75 + (iqr*1.5)
    print(minimum,maximum)
    #Replace with NA
    Cab_Train_Data.loc[Cab_Train_Data[col] < minimum,col] = np.nan
    Cab_Train_Data.loc[Cab_Train_Data[col] > maximum,col] = np.nan
```

```
#Outlier analysis
outlier_treatment('fare_amount')
outlier_treatment('pickup_longitude')
outlier_treatment('pickup_latitude')
outlier_treatment('dropoff_longitude')
outlier_treatment('dropoff_latitude')
```

```
#Check for null values generated by outliers
pd.DataFrame(Cab_Train_Data.isnull().sum())
```

Output:

fare_amount	1358
pickup_longitude	805
pickup_latitude	510
dropoff_longitude	920
dropoff_latitude	754
passenger_count	0

#Computing missing values generated by outlier analysis.
#Since value computed by median method is closer to the actual value when compared to mean method so will be using median method to compute missing values present in the dataset.

```
Cab_Train_Data["fare_amount"]=Cab_Train_Data["fare_amount"].fillna(Cab_Train_Data["fare_amount"].median())
Cab_Train_Data["pickup_longitude"]=Cab_Train_Data["pickup_longitude"].fillna(Cab_Train_Data["pickup_longitude"].median())
```

Cab_Train_Data["pickup_latitude"]=Cab_Train_Data["pickup_latitude"].fillna(Cab_Train_Data["pickup_latitude"].median())
Cab_Train_Data["dropoff_latitude"]=Cab_Train_Data["dropoff_latitude"].fillna(Cab_Train_Data["dropoff_latitude"].median())
Cab_Train_Data["dropoff_longitude"]=Cab_Train_Data["dropoff_longitude"].fillna(Cab_Train_Data["dropoff_longitude"].median())

#Checking the number of missing values in the variables after computing missing value.
Miss_value=pd.DataFrame(Cab_Train_Data.isnull().sum())
Miss_value

Output:

	0
fare_amount	0
pickup_longitude	0
pickup_latitude	0
dropoff_longitude	0
dropoff_latitude	0
passenger_count	0

Step-5: Feature Engineering

1. ‘pickup_datetime’ variable:

Feature Engineering is used to drive new features from existing features. We will use this timestamp variable to create new variables. New features will be year, month, day_of_week, hour. ‘year’ will contain only years from pickup_datetime. For ex. 2009, 2010, 2011, etc. ‘month’ will contain only months from pickup_datetime. For ex. 1 for January, 2 for February, etc. ‘day_of_week’ will contain only week from pickup_datetime. For ex. 1 which is for Monday,2 for Tuesday,etc. ‘hour’ will contain only hours from pickup_datetime. For ex. 1, 2, 3, etc.We will categorize them to new variables like Session from hour column, seasons from month column, week:weekday/weekend from day_of_week variable.

So, session variable which will contain categories—morning, afternoon, evening, night_PM, night_AM. Seasons variable will contain categories—spring, summer, fall, winter. Week will contain categories—weekday, weekend.

2. ‘passenger_count’ variable:

As passenger_count is a categorical variable we will one-hot-encode it.

3. ‘Latitudes’ and ‘Longitudes’ variables:

As we have latitude and longitude data for pickup and dropoff, we will find the distance the cab travelled from pickup and dropoff location using geodesic method from geopy library.

Following codes used to perform above task:

```
################## FEATURE ENGINEERING ###################

#Deriving new features from pickup_datetime variables like year,month,day_of_week,hour

#Joining two Dataframes pickup_datetime and train
Cab_Train_Data = pd.merge(pickup_datetime,Cab_Train_Data,right_index=True,left_index=True)
Cab_Train_Data.head()

#Check for NA values
pd.DataFrame(Cab_Train_Data.isna().sum())
```

Ouptut:

	0
pickup_datetime	1
fare_amount	0
pickup_longitude	0
pickup_latitude	0
dropoff_longitude	0
dropoff_latitude	0
passenger_count	0

```
#Droping NA's
Cab_Train_Data=Cab_Train_Data.dropna()

#Driving new feature
data=[Cab_Train_Data,Cab_Test_Data]
for i in data:
    i["year"] = i["pickup_datetime"].apply(lambda row: row.year)
    i["month"] = i["pickup_datetime"].apply(lambda row: row.month)
    i["day_of_week"] = i["pickup_datetime"].apply(lambda row: row.dayofweek)
    i["hour"] = i["pickup_datetime"].apply(lambda row: row.hour)


#Plotting new features
plt.figure(figsize=(20,10))
sns.countplot(Cab_Train_Data['year'])
plt.savefig('year_python.png')
```

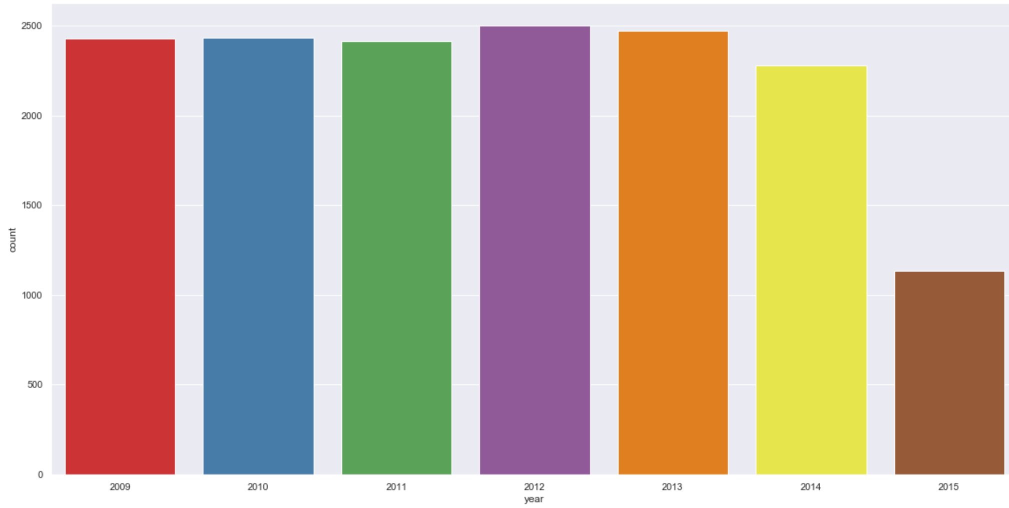

Fig 6: Plot for year variable

```
plt.figure(figsize=(20,10))
sns.countplot(Cab_Train_Data['month'])
plt.savefig('month_python.png')
```

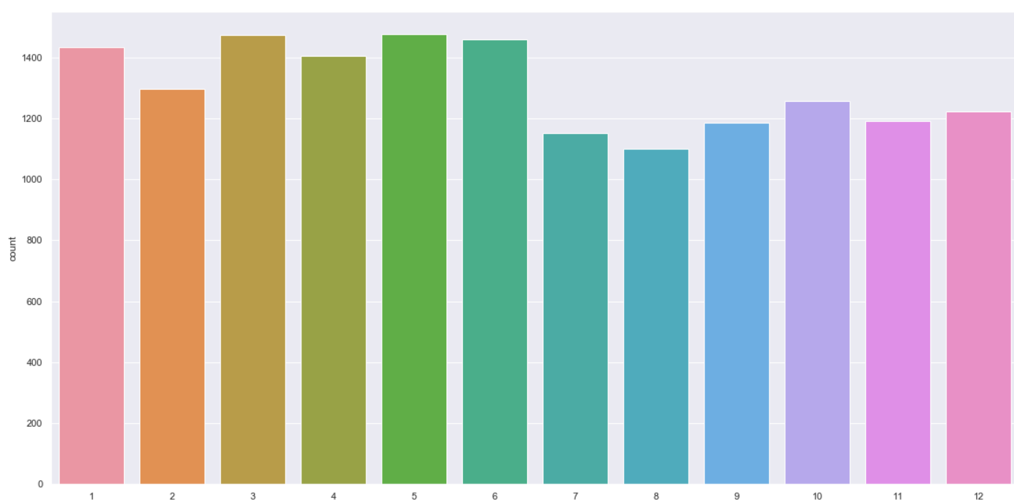

Fig 7: Plot for month variable

```
plt.figure(figsize=(20,10))
sns.countplot(Cab_Train_Data['day_of_week'])
plt.savefig('day_of_week_python.png')
```

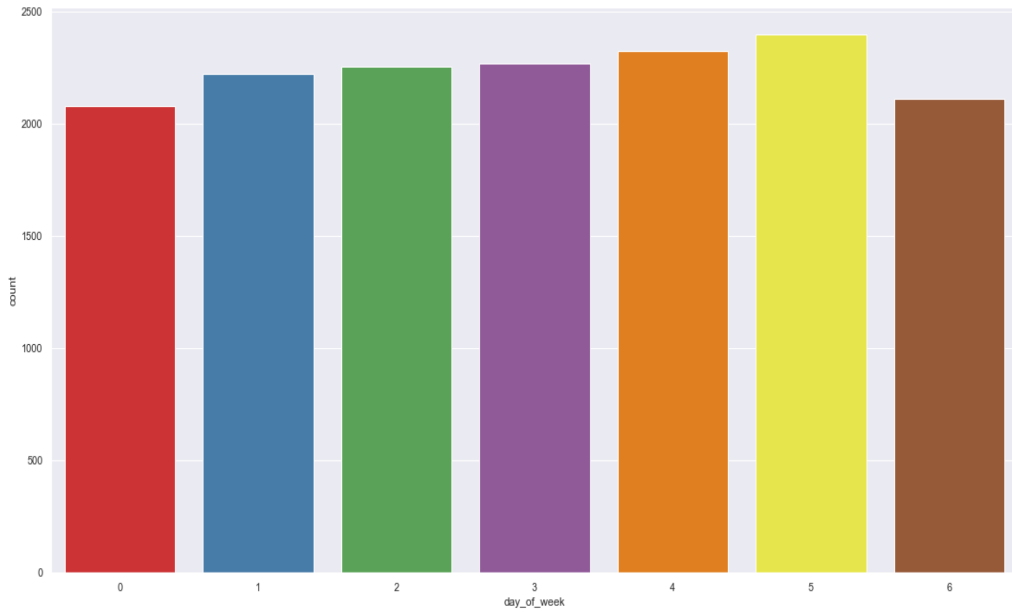

Fig 8: Plot for month variable

```
plt.figure(figsize=(20,10))
sns.countplot(Cab_Train_Data['hour'])
plt.savefig('hour_python.png')
```

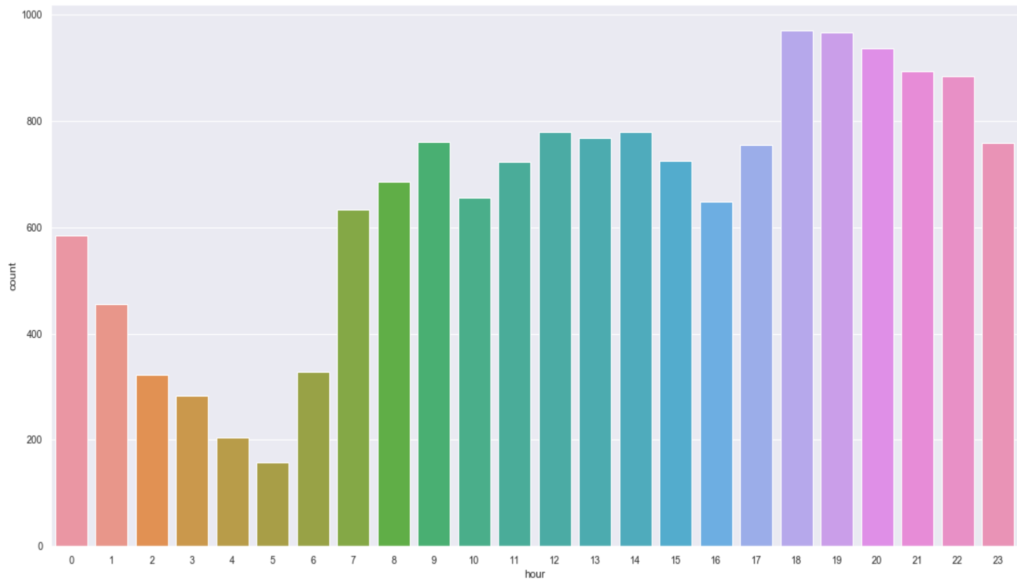

Fig 9: Plot for month variable

#Using month,day_of_week,hour for deriving new features like sessions in a day,seasons in a year,week:weekend/weekday

#function for sessions in a day using hour variable

```
def session(x):
    if (x >=5) and (x <= 11):
        return 'morning'
    elif (x >=12) and (x <=16 ):
        return 'afternoon'
    elif (x >= 17) and (x <= 20):
        return'evening'
    elif (x >=21) and (x <= 23) :
        return 'night_PM'
    elif (x >=0) and (x <=4):
        return'night_AM'
```

#function for seasons in a year using month variable

```
def season(x):
    if (x >=3) and (x <= 5):
        return 'spring'
    elif (x >=6) and (x <=8):
        return 'summer'
```

```
    elif (x >= 9) and (x <= 11):
        return 'fall'
    elif (x >=12)|(x <= 2) :
        return 'winter'
```

#function for weekday/weekend in a day_of_week variable

```
def day(x):
    if (x >=0) and (x <= 4):
        return 'weekday'
    elif (x >=5) and (x <=6 ):
        return 'weekend'
```

#Using session function for deriving session variable from hour

```
Cab_Train_Data['session'] = Cab_Train_Data['hour'].apply(session)
Cab_Test_Data['session'] = Cab_Test_Data['hour'].apply(session)
```

#Using seasons function for deriving season variable from month

```
Cab_Train_Data['seasons'] =Cab_Train_Data['month'].apply(season)
Cab_Test_Data['seasons'] = Cab_Test_Data['month'].apply(season)
```

#Using day function for deriving day variable from day_of_week

```
Cab_Train_Data['week'] = Cab_Train_Data['day_of_week'].apply(day)
Cab_Test_Data['week'] = Cab_Test_Data['day_of_week'].apply(day)
```

#2. Feature Engineering for passenger_count variable
#Models in scikit learn require numerical input,if dataset contains categorical variables then we have to encode them,using one hot encoding technique for passenger_count variable.

```
temp = pd.get_dummies(Cab_Train_Data['passenger_count'], prefix = 'passenger_count')
Cab_Train_Data =Cab_Train_Data.join(temp)
temp = pd.get_dummies(Cab_Test_Data['passenger_count'], prefix = 'passenger_count')
Cab_Test_Data = Cab_Test_Data.join(temp)

temp = pd.get_dummies(Cab_Train_Data['seasons'], prefix = 'season')
Cab_Train_Data = Cab_Train_Data.join(temp)
temp = pd.get_dummies(Cab_Test_Data['seasons'], prefix = 'season')
```

```
Cab_Test_Data = Cab_Test_Data.join(temp)

temp = pd.get_dummies(Cab_Train_Data['week'], prefix = 'week')
Cab_Train_Data = Cab_Train_Data.join(temp)
temp = pd.get_dummies(Cab_Test_Data['week'], prefix = 'week')
Cab_Test_Data = Cab_Test_Data.join(temp)

temp = pd.get_dummies(Cab_Train_Data['session'], prefix = 'sessions')
Cab_Train_Data = Cab_Train_Data.join(temp)
temp = pd.get_dummies(Cab_Test_Data['session'], prefix = 'sessions')
Cab_Test_Data = Cab_Test_Data.join(temp)

temp = pd.get_dummies(Cab_Train_Data['year'], prefix = 'year')
Cab_Train_Data = Cab_Train_Data.join(temp)
temp = pd.get_dummies(Cab_Test_Data['year'], prefix = 'year')
Cab_Test_Data = Cab_Test_Data.join(temp)

#Getting column names of train data
Cab_Train_Data.columns
```

Output:

```
#Getting column names of train data
Cab_Train_Data.columns
```

```
Index(['pickup_datetime', 'fare_amount', 'pickup_longitude', 'pickup_latitude',
       'dropoff_longitude', 'dropoff_latitude', 'passenger_count', 'year',
       'month', 'day_of_week', 'hour', 'session', 'seasons', 'week',
       'passenger_count_1', 'passenger_count_2', 'passenger_count_3',
       'passenger_count_4', 'passenger_count_5', 'passenger_count_6',
       'season_fall', 'season_spring', 'season_summer', 'season_winter',
       'week_weekday', 'week_weekend', 'sessions_afternoon',
       'sessions_evening', 'sessions_morning', 'sessions_night_AM',
       'sessions_night_PM', 'year_2009', 'year_2010', 'year_2011', 'year_2012',
       'year_2013', 'year_2014', 'year_2015'],
      dtype='object')
```

```
#Getting column names of test data
Cab_Test_Data.columns
```

Output:

```
#Getting column names of test data
Cab_Test_Data.columns
```

```
Index(['pickup_datetime', 'pickup_longitude', 'pickup_latitude',
       'dropoff_longitude', 'dropoff_latitude', 'passenger_count', 'year',
       'month', 'day_of_week', 'hour', 'session', 'seasons', 'week',
       'passenger_count_1', 'passenger_count_2', 'passenger_count_3',
       'passenger_count_4', 'passenger_count_5', 'passenger_count_6',
       'season_fall', 'season_spring', 'season_summer', 'season_winter',
       'week_weekday', 'week_weekend', 'sessions_afternoon',
       'sessions_evening', 'sessions_morning', 'sessions_night_AM',
       'sessions_night_PM', 'year_2009', 'year_2010', 'year_2011', 'year_2012',
       'year_2013', 'year_2014', 'year_2015'],
      dtype='object')
```

#drop one column from each one-hot-encoded variables

```
Cab_Train_Data=Cab_Train_Data.drop(['passenger_count_1','season_fall','week_weekday','sessions_afternoon','year_2009'],axis=1)
Cab_Test_Data=Cab_Test_Data.drop(['passenger_count_1','season_fall','week_weekday','sessions_afternoon','year_2009'],axis=1)
```

#3. Feature Engineering for latitude and longitude variable

#finding the distance, the cab travelled from pickup and dropoff longitudes and latitudes.

Calculate distance the cab travelled from pickup and dropoff location using geodesic from geopy library

```
data=[Cab_Train_Data,Cab_Test_Data]
for i in data:
   i['distance']=i.apply(lambda x: geodesic((x['pickup_latitude'],x['pickup_longitude']), (x['dropoff_latitude'], x['dropoff_longitude'])).miles, axis=1)
```

#Removing variables which were used to feature engineer new variables

```
Cab_Train_Data=Cab_Train_Data.drop(['pickup_datetime','pickup_longitude', 'pickup_latitude',
     'dropoff_longitude', 'dropoff_latitude', 'passenger_count', 'year',
     'month', 'day_of_week', 'hour', 'session', 'seasons', 'week'],axis=1)
```

```
Cab_Test_Data=Cab_Test_Data.drop(['pickup_datetime','pickup_longitude', 'pickup_latitude',
       'dropoff_longitude', 'dropoff_latitude', 'passenger_count', 'year',
       'month', 'day_of_week', 'hour', 'session', 'seasons', 'week'],axis=1)

#Check for outliers in distance
plt.figure(figsize=(20,5))
plt.xlim(0,100)
sns.boxplot(x=Cab_Train_Data['distance'],data=Cab_Train_Data,orient='h')
plt.title('Boxplot of distance ')
plt.savefig('bp_distance_python.png')
plt.show()
```

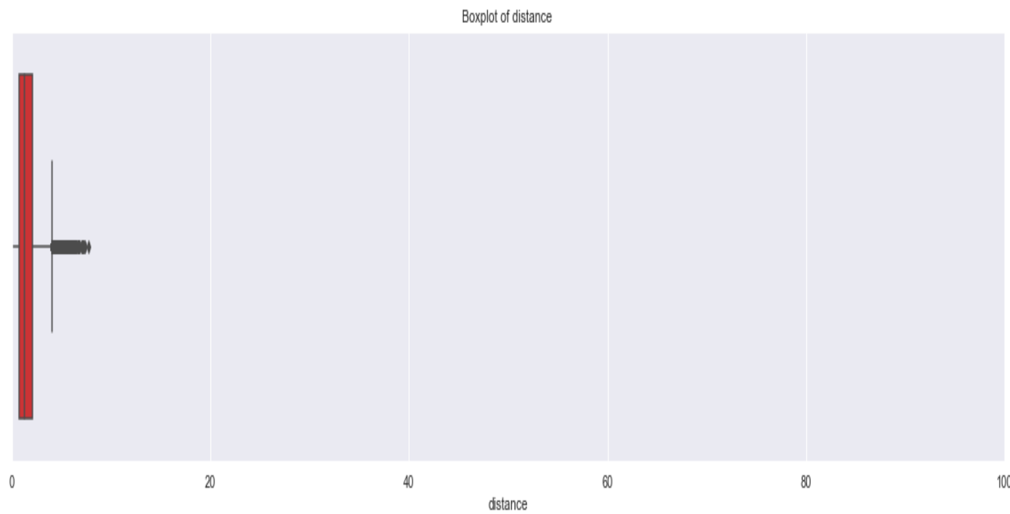

Fig 10: BoxPlot for distance variable

```
#Calling outlier function and replacing outliers with NA
outlier_treatment('distance')

#Check for NA values
pd.DataFrame(Cab_Train_Data.isnull().sum())
```

Output:

	0
fare_amount	0
passenger_count_2	0
passenger_count_3	0
passenger_count_4	0
passenger_count_5	0
passenger_count_6	0
season_spring	0
season_summer	0
season_winter	0
week_weekend	0
sessions_evening	0
sessions_morning	0
sessions_night_AM	0
sessions_night_PM	0

passenger_count_5	0
passenger_count_6	0
season_spring	0
season_summer	0
season_winter	0
week_weekend	0
sessions_evening	0
sessions_morning	0
sessions_night_AM	0
sessions_night_PM	0
year_2010	0
year_2011	0
year_2012	0
year_2013	0
year_2014	0
year_2015	0
distance	592

#Computing missing values generated by outlier analysis.
#Since value computed by median method is closer to the actual value when compared to mean method so will be using median method to compute missing values present in the dataset.

```
Cab_Train_Data["distance"]=Cab_Train_Data["distance"].fillna(Cab_Train_Data[
"distance"].median())
```

#Checking the number of missing values in the variables after computing missing value.

```
Miss_value=pd.DataFrame(Cab_Train_Data.isnull().sum())
Miss_value
```

Output:

season_summer	0
season_winter	0
week_weekend	0
sessions_evening	0
sessions_morning	0
sessions_night_AM	0
sessions_night_PM	0
year_2010	0
year_2011	0
year_2012	0
year_2013	0
year_2014	0
year_2015	0
distance	0

Step-6: Feature selection

This stage involves the process of reducing variables on the basis of correlation present in the variables of the dataset. Correlation plot is being used to analyze the correlation among the variables and reduce the dimensionality on the basis of correlation between variables. In this step we would allow only to pass relevant features to further steps. We remove irrelevant features from the dataset. We do this by some statistical techniques, like we look for features which will not be helpful in predicting the target variables. In this dataset we have to predict the fare_amount.
Further below are some types of test involved for feature selection:

1. **Correlation analysis** – This requires only numerical variables. Therefore, we will filter out only numerical variables and feed it to correlation analysis. We do this by plotting correlation plot for all numerical variables. There should be no correlation between independent variables but there should be high correlation between independent variable and dependent variable. So, we plot the correlation plot.

Following codes used to perform above task:

```
################## FEATURE SELECTION ###################

################## CORRELATION PLOT ####################

#Plotting correlation graph for the variables to look the correlation among variables.
# heatmap using correlation matrix
plt.figure(figsize=(15,15))
_ = sns.heatmap(Cab_Train_Data[num_var].corr(), square=True, cmap='RdYlGn',linewidths=0.5,linecolor='w',annot=True)
plt.title('Correlation matrix ')
plt.savefig('correlation.png')
plt.show()
```

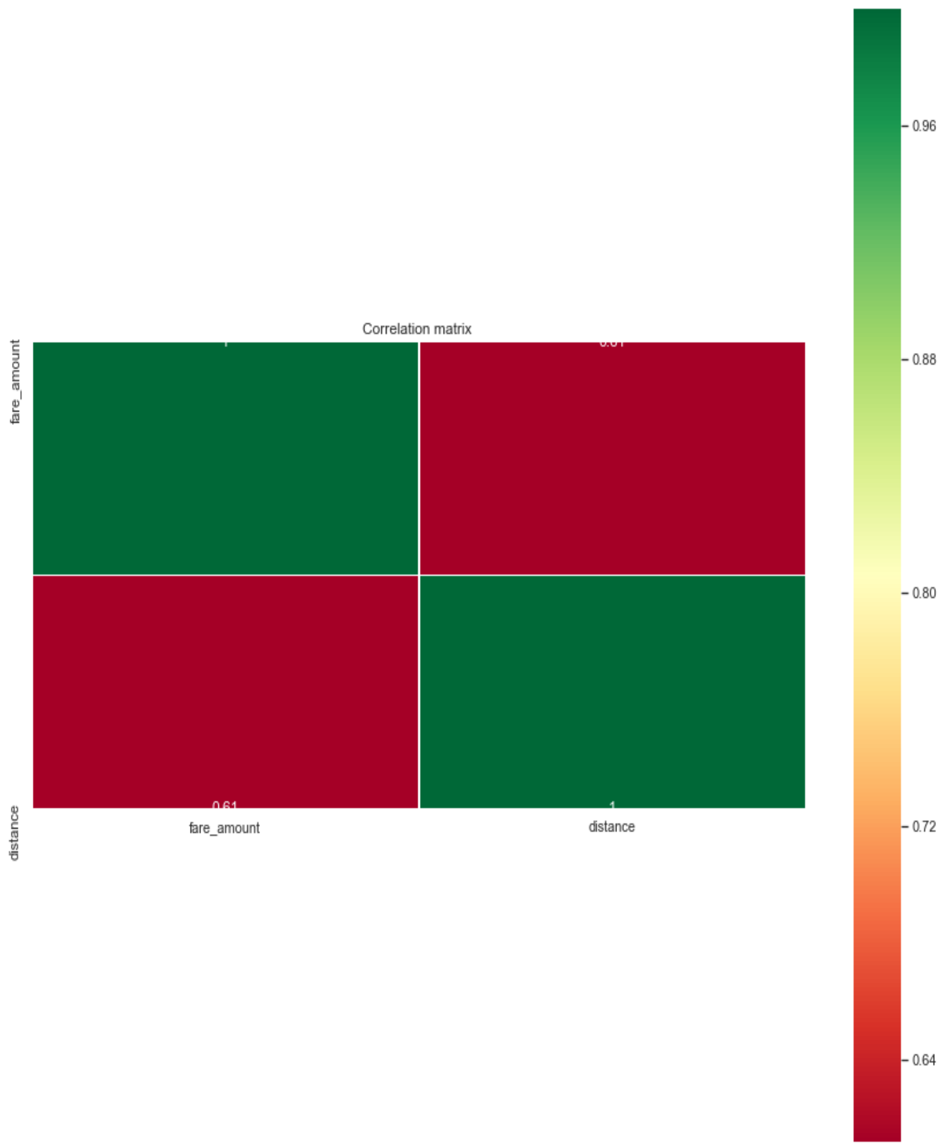

Fig 11: Correlation plot

From above correlation plot we see that:
• 'fare_amount' and 'distance' are very highly correlated with each other.
• As fare_amount is the target variable and 'distance' is independent variable we will keep 'distance' because it will help to explain variation in fare_amount.

2. Analysis of Variance (Anova) Test – It is carried out to compare between each group in a categorical variable. ANOVA only lets us know the means for different groups are same or not. It doesn't help us identify which mean is different.
Hypothesis testing:
- **Null Hypothesis**: mean of all categories in a variable are same.
- **Alternate Hypothesis**: mean of at least one category in a variable is different.

If p-value is less than 0.05 then we reject the null hypothesis and if p-value is greater than 0.05 then we accept the null hypothesis.

Following codes used to perform above task:

```
################## ANOVA TEST ###################

#ANOVA Test for variance analysis

model = ols('fare_amount ~ C(passenger_count_2)+C(passenger_count_3)+C(passenger_count_4)+C(passenger_count_5)+C(passenger_count_6)+C(season_spring)+C(season_summer)+C(season_winter)+C(week_weekend)+C(sessions_night_AM)+C(sessions_night_PM)+C(sessions_evening)+C(sessions_morning)+C(year_2010)+C(year_2011)+C(year_2012)+C(year_2013)+C(year_2014)+C(year_2015)',data=Cab_Train_Data).fit()

aov_table = sm.stats.anova_lm(model)
aov_table
```

Below is the Anova analysis table for each categorical variable:

	df	sum_sq	mean_sq	F	PR(>F)
C(passenger_count_2)	1.0	10.143360	10.143360	0.664134	4.151166e-01
C(passenger_count_3)	1.0	11.631197	11.631197	0.761550	3.828573e-01
C(passenger_count_4)	1.0	83.429285	83.429285	5.462513	1.944124e-02
C(passenger_count_5)	1.0	26.346008	26.346008	1.724999	1.890701e-01
C(passenger_count_6)	1.0	181.104035	181.104035	11.857744	5.757248e-04
C(season_spring)	1.0	46.306709	46.306709	3.031921	8.166098e-02
C(season_summer)	1.0	24.691308	24.691308	1.616658	2.035774e-01
C(season_winter)	1.0	218.544561	218.544561	14.309154	1.556866e-04
C(week_weekend)	1.0	27.376687	27.376687	1.792482	1.806435e-01
C(sessions_night_AM)	1.0	902.107990	902.107990	59.065308	1.615514e-14
C(sessions_night_PM)	1.0	190.779256	190.779256	12.491227	4.100478e-04
C(sessions_evening)	1.0	80.580843	80.580843	5.276012	2.163437e-02
C(sessions_morning)	1.0	55.006069	55.006069	3.601509	5.774553e-02
C(year_2010)	1.0	873.686368	873.686368	57.204409	4.145799e-14
C(year_2011)	1.0	787.380962	787.380962	51.553583	7.280493e-13
C(year_2012)	1.0	295.353060	295.353060	19.338172	1.102182e-05
C(year_2013)	1.0	286.906686	286.906686	18.785148	1.472156e-05
C(year_2014)	1.0	901.557172	901.557172	59.029243	1.645283e-14
C(year_2015)	1.0	1398.996484	1398.996484	91.598965	1.216734e-21
Residual	15640.0	238870.657695	15.273060	NaN	NaN

Fig 12: ANOVA test on Variables

Since every variable has p-value less than 0.05 therefore we reject the null hypothesis.

Step 7: Multicollinearity

In regression, "multicollinearity" refers to predictors that are correlated with other predictors. Multicollinearity occurs when your model includes multiple factors that are correlated not just to your response variable, but also to each other. Multicollinearity increases the standard errors of the coefficients. It Increases standard errors in turn means that coefficients for some independent variables may be found not to be significantly different from 0.

In other words, by overinflating the standard errors, multicollinearity makes some variables statistically insignificant when they should be significant. Without multicollinearity (and thus, with lower standard errors), those coefficients might be significant.

VIF is always greater or equal to 1.

if VIF is 1: Not correlated to any of the variables.

if VIF is between 1-5: Moderately correlated.

if VIF is above 5: Highly correlated.

If there are multiple variables with VIF greater than 5, only remove the variable with the highest VIF and if the VIF goes above 10, you can assume that the regression coefficients are poorly estimated due to multicollinearity.

Following codes used to perform above task:

```
#Multicollinearity Test
outcome, predictors = dmatrices('fare_amount ~ distance+passenger_count_2+passenger_count_3+passenger_count_4+passenger_count_5+passenger_count_6+season_spring+season_summer+season_winter+week_weekend+sessions_night_AM+sessions_night_PM+sessions_evening+sessions_morning+year_2010+year_2011+year_2012+year_2013+year_2014+year_2015',Cab_Train_Data, return_type='dataframe')
# calculating VIF for each individual Predictors
vif = pd.DataFrame()
vif["VIF"] = [variance_inflation_factor(predictors.values, i) for i in range(predictors.shape[1])]
vif["features"] = predictors.columns
vif
```

	VIF	features
0	15.504918	Intercept
1	1.040512	passenger_count_2[T.1]
2	1.019458	passenger_count_3[T.1]
3	1.011721	passenger_count_4[T.1]
4	1.024861	passenger_count_5[T.1]
5	1.017256	passenger_count_6[T.1]
6	1.642376	season_spring[T.1]
7	1.552574	season_summer[T.1]
8	1.587430	season_winter[T.1]
9	1.051441	week_weekend[T.1]
10	1.360788	sessions_night_AM[T.1]
11	1.422485	sessions_night_PM[T.1]
12	1.526961	sessions_evening[T.1]
13	1.558828	sessions_morning[T.1]
14	1.691593	year_2010[T.1]
15	1.687880	year_2011[T.1]
16	1.711905	year_2012[T.1]
17	1.710163	year_2013[T.1]
18	1.665408	year_2014[T.1]
19	1.406998	year_2015[T.1]
20	1.013592	distance

Fig 13: VIF analysis

#VIF is always greater or equal to 1. If there are multiple variables with VIF greater than 5, we remove the variable with the highest VIF.
#In our case VIF is very low that means our dataset has low multicollinearity

Step-8 : **Feature Scaling**

Data Scaling methods are used when we want our variables in data to scaled on common ground. It is performed only on continuous variables.

• **Normalization**: Normalization refer to the dividing of a vector by its length. normalization normalizes the data in the range of 0 to 1. It is generally used when we are planning to use distance method for our model development purpose such as KNN. Normalizing the data improves convergence of such algorithms. Normalization of data scales the data to a very small interval, where outliers can be loosed.

• **Standardization**: Standardization refers to the subtraction of mean from individual point and then dividing by its SD. Z is negative when the raw score is below the mean and Z is positive when above mean. When the data is distributed normally you should go for standardization.

Linear Models assume that the data you are feeding are related in a linear fashion, or can be measured with a linear distance metric. Also, our independent numerical variable 'distance' is not distributed normally so we had chosen normalization over standardization. High variance will affect the accuracy of the model. So, we want to normalise that variance. It is performed only on Continuous variables.

Following codes used to perform above task:

```
################## FEATURE SCALING ###################

#Normality check
plt.hist(Cab_Train_Data["distance"],bins="auto")
plt.savefig('Normality_check_Python.png')
```

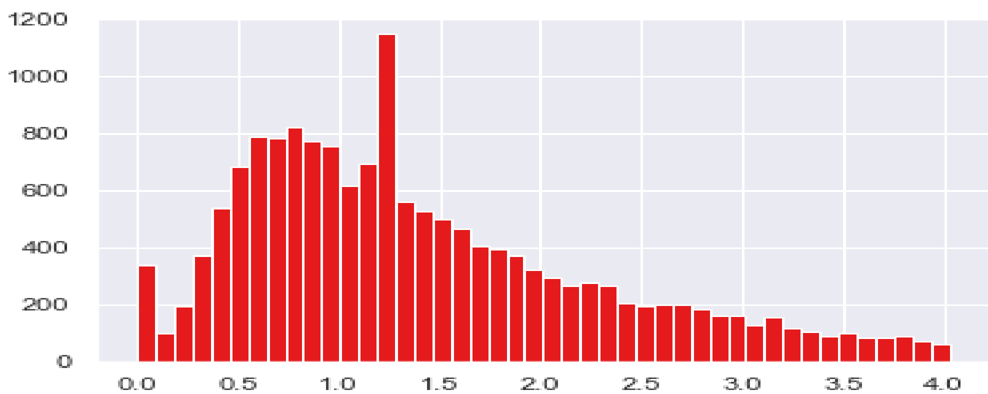

Fig 14: Normality check on distance variable

On observing plots, it is clear that the data of the distance variable is not uniformly distributed. Hence in this applying normalization for proper scaling of the distance variable.

```
#Since data is not normally distributed so applying normalization method to normalize datasets
Cab_Train_Data['distance'] = (Cab_Train_Data['distance'] - min(Cab_Train_Data['distance']))/(max(Cab_Train_Data['distance']) - min(Cab_Train_Data['distance']))
Cab_Test_Data['distance'] = (Cab_Test_Data['distance'] - min(Cab_Test_Data['distance']))/(max(Cab_Test_Data['distance']) - min(Cab_Test_Data['distance']))
```

Step-9: Splitting train and Validation Dataset

We have used **train_test_split()** method to divide whole Dataset into train and validation dataset. 20% is in validation dataset and 80% is in training data. We will test the performance of model on validation dataset. The model which performs best will be chosen to perform on test dataset provided along with original train dataset.

- X_train Y_train--are train subset.
- X_test Y_test--are validation subset.

OR

train,test = train_test_split(Cab_Train_Data, test_size = 0.2)

Following codes used to perform above task:

```
################## SPLITTING DATA ###################

#Splitting train into train and validation subsets
train,test = train_test_split(Cab_Train_Data, test_size = 0.2)
```

OR

#Splitting train into train and validation subsets

```
X = Cab_Train_Data.drop('fare_amount',axis=1).values
Y = Cab_Train_Data['fare_amount'].values
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.25, random_state=42)
```

Step- 10: Model Development

Our problem statement wants us to predict the fare_amount. This is a Regression problem. So, we are going to build regression models on training data and predict it on test data. In this project I have built models using four Algorithms:

- Linear Regression
- Decision Tree
- Random Forest
- Xgboost Regression

We will evaluate performance on validation dataset which was generated using Sampling. We will deal with specific error metrics like –

Regression metrics for our Models:

- MAPE(Mean Absolute Percentage Error)

################### LINEAR REGRESSION ####################

#Running regression model

```
model_LR = LinearRegression().fit(X_train,Y_train)
```

#Making predictions

```
prediction_LR=model_LR.predict(X_test)
```

#Displaying predicted values

```
prediction_LR
```

Output:

```
#Displaying predicted values
prediction_LR
```

```
array([5.54133695, 6.52949837, 7.64841659, ..., 7.24991755, 7.37326752,
       7.67713501])
```

#Function for calculating error rate
def MAPE(actual , predicted):
    mape=np.mean(np.abs((actual-predicted)/predicted))*100
    return mape

#Calculate MAPE
MAPE(Y_test,prediction_LR)

Output:
```
25.119894528549107
```

#Linear Regression model
#Error rate = 25.1
#Accuracy = 74.9%

################## DECISION TREE ###################

#Running regression DT model
model_DT=DecisionTreeRegressor(max_depth=10).fit(X_train,Y_train)

#Making predictions
predictions_DT=model_DT.predict(X_test)

#Displaying predicted values
prediction_DT

Output:

```
predictions_DT

array([ 5.20948905,  6.68766141, 15.7125    , ...,  8.09828866,
        6.75694444, 10.25      ])
```

#Function for calculating error rate
def MAPE(actual , predicted):
    mape=np.mean(np.abs((actual-predicted)/predicted))*100
    return mape

#Calculate MAPE
MAPE(Y_test,predictions_DT)

Output:
```
23.66692092200502
```

#Decision Tree Algorithm
#Error rate = 23.6
#Accuracy = 76.4%

################### RANDOM FOREST ####################

#Running Random Forest model
model_RF =
RandomForestRegressor(n_estimators=600,max_features=5).fit(X_train,Y_train)

#Making predictions
predictions_RF=model_RF.predict(X_test)

#viewing predicted values
predictions_RF

Output:

```
#viewing predicted values
predictions_RF
```

```
array([ 4.65566667,  5.663     , 12.51302778, ...,  8.525     ,
        7.09233333, 10.93033333])
```

#Function for calculating error rate
```
def MAPE(actual , predicted):
   mape=np.mean(np.abs((actual-predicted)/predicted))*100
   return mape
```

#Calculate MAPE
MAPE(Y_test,predictions_RF)

Output:
```
24.06036226
```

#Random Forest model
#Error rate= 24.8
#Accuracy= 75.2%

################## XGBOOST MODEL ###################

```
#Running XGboost algorithm
Xgb = XGBRegressor()
Xgb.fit(X_train,Y_train)


#Making Predictions
prediction_xgb = Xgb.predict(X_test)

#Displaying predicted values obtained from predict() method of xgboost model
prediction_xgb
```

Output:

```
#Displaying predicted values obtained from predict() method of xgboost model
prediction_xgb
```

```
array([ 5.185576 ,  6.199656 , 13.329259 , ...,  8.52641  ,  6.2573977,
       10.501133 ], dtype=float32)
```

```
#Function for calculating error rate
def MAPE(actual , predicted):
   mape=np.mean(np.abs((actual-predicted)/predicted))*100
   return mape

#Calculate MAPE
MAPE(Y_test,prediction_xgb)
```

Output:
```
22.64223541018341
```

#XGBOOST model
#Error rate = 22.6
#Accuracy = 77.4%

Step 11: Model Selection

After developing model, we will evaluate the performance of the model by considering generated Regression metrics for our Models in development stage:

- MAPE (Mean Absolute Percentage Error)

The lesser the error rate better will be the performance and the accuracy of the model. On comparing these regression metrics of different models applied on the dataset we will select that model who has low error rate and better accuracy compared to other models for cab fare prediction.

Models	MAPE	Error Rate	Accuracy
Linear Regression	25.1	25.1	74.9%
Decision Tree	23.6	23.6	76.4%
Random Forest	24.7	24.7	75.3%
Xgboost model	22.6	22.6	77.4%

On Comparing different regression metrics Xgboost model found to be better model than other models and have good performance and accuracy as well as low error rate. So, selecting XGBOOST model for cab fare prediction.

Step 11: Conclusion/Result:

#################### CAB FARE PREDICTION ######################

#Selecting XGBOOST algorithm than other algorithms as it has comparatively low error rate and better accuracy when compared with other models
#Using XGBOOST model for cab fare prediction

#loading test dataset
Test_data=pd.read_csv('test.csv')
test_pickup_datetime=Test_data['pickup_datetime']

#predicting fare_amount from test dataset
```
Fare_prediction =Xgb.predict(Cab_Test_Data.values)
```

#Viewing predicted fare_amount
```
Fare_prediction
```

#Prediction of fare_amount with respect to the test data
```
Cab_Fare_Prediction =
pd.DataFrame({"pickup_datetime":test_pickup_datetime,"fare_amount" :
Fare_prediction})
```

#Displaying predicted fare_amount for test dataset
```
Cab_Fare_Prediction
```

Output:

	pickup_datetime	fare_amount
0	2015-01-27 13:08:24 UTC	6.954877
1	2015-01-27 13:08:24 UTC	5.969587
2	2011-10-08 11:53:44 UTC	5.510411
3	2012-12-01 21:12:12 UTC	7.531249
4	2012-12-01 21:12:12 UTC	5.207213
...	...	...
9909	2015-05-10 12:37:51 UTC	8.910440
9910	2015-01-12 17:05:51 UTC	5.645076
9911	2015-04-19 20:44:15 UTC	7.557964
9912	2015-01-31 01:05:19 UTC	6.266886
9913	2015-01-18 14:06:23 UTC	7.797482

9914 rows × 2 columns

#writing csv file

```
Cab_Fare_Prediction.to_csv("Cab_Fare_Prediction_By_Python.csv",index=False)
```

Note: The predicted fare_amount for the test data is present in the file "Cab_Fare_Prediction_By_Python.csv" which is attached with the report.