Cab Fare Prediction

Vishal Tyagi

09-10-2019

Contents

1. INTRODUCTION

1.1 Problem statement

There is a cab rental start-up company which wants to launch a cab service. They have successfully run the pilot project and now want to launch your cab service ac!ross the country. They have collected the historical data from their pilot project and now have a requirement to apply analytics for fare prediction.

The objective of this Project is to predict the fare of the Cab rental in the city. This Fare prediction takes distance, date/time and other factors in to account from historical data which was gathered from the pilot project for the same. We would be building a model that can successfully predict the fare of rentals on relevant factors.

1.2 Data

Understanding of data is the very first and important step in the process of finding solution of any business problem. Let's have a quick preview of the train and test data. Will discuss in detail about the data understanding in the CRISP-DM process section. Let's understand shape of training and test dataset:

Train Dataset

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

Test Dataset

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

Below mentioned is a list of all the variable names with their meanings:

Variables	Description
fare_amount	Fare amount
pickup_datetime	Cab pickup date with time
pickup_longitude	Pickup location longitude
pickup_latitude	Pickup location latitude
dropoff_longitude	Drop location longitude
dropoff_latitude	Drop location latitude
passenger_count	Number of passengers sitting in the cab

Now let’s have a look at the data type of dataset attributes.

```
fare_amount       object
pickup_datetime    object
pickup_longitude  float64
pickup_latitude    float64
dropoff_longitude float64
dropoff_latitude   float64
passenger_count   float64
dtype: object
```

Here, the datatype of fare_amount attribute is an object which is not correct. So we converted this attribute into numeric. But, while converting it to numeric we found a problem that it contains a string value “-430” at location 1123. So we basically replaced this value with 430 and then converted it to a numeric datatype. Also passenger_count variable has datatype float so once again we will convert it to object or factor datatype.

2.Methodology

2.1- Pre-Processing

When we required to build a predictive model, we require to look and manipulate the data before we start modelling which includes multiple preprocessing steps such as exploring the data, cleaning the data as well as visualizing the data through graph and plots, all these steps is combined under one shed which is **Exploratory Data Analysis**, which includes following steps:

- Missing values treament
- Outlier Analysis
- Feature Engineering
- Feature Selection
- Features Scaling
- Visualization

2.2- Modelling

Once all the Pre-Processing steps has been done on our data set, we will now further move to our next step which is modelling. Modelling plays an important role to find out the good inferences from the data. Choice of models depends upon the problem statement and data set. As per our problem statement and dataset, we will try some models on our preprocessed data and post comparing the output results we will select the best suitable model for our problem. As per our data set following models need to be tested:

- Linear regression
- Decision Tree
- Random forest,

2.3- Model Selection

The final step of our methodology will be the selection of the model based on the different output and results shown by different models. We have multiple parameters which we will study further in our report to test whether the model is suitable for our problem statement or not.

3-Pre-Processing

Missing Value Analysis

Missing value analysis is a method or technique to find out if there are missing values in the attributes of the dataset. When we applied missing value analysis on our dataset we found that passenger_count had most numbers of missing values followed by fare_amount. Except for these two variables, there were no missing values in any other variables. Missing values can be found by using these syntaxes

is.null( ).sum() in python
function(x){sum(is.na(x))} in R

Train Data	Test data
fare_amount 24 pickup_datetime 1 pickup_longitude 0 pickup_latitude 0 dropoff_longitude 0 dropoff_latitude 0 passenger_count 55	pickup_datetime 0 pickup_longitude 0 pickup_latitude 0 dropoff_longitude 0 dropoff_latitude 0 passenger_count 0

As we can see missing values above only in train data, during conversion of pickup_datetime variable, one value has been converted in missing value, so I dropped it. And calculated missing value percentages

Depending on the percentage of missing values we can decide if we need to keep a variable or drop it based on the following conditions:

- Missing value percentage < 30% - We can include the variable.
- Missing value percentage > 30 % - We need to remove those variable/s

because even if we impute values, they are not the actual values, the variable will not contain actual values and hence will not contribute effectively to our model.

	Variables	Missing_percentage
0	passenger_count	0.342317
1	fare_amount	0.149374
2	pickup_datetime	0.000000
3	pickup_longitude	0.000000
4	pickup_latitude	0.000000
5	dropoff_longitude	0.000000
6	dropoff_latitude	0.000000

For the given train dataset, we can see that we have only two variables fare_amount and passenger count with missing values and the percentage of missing value is less as per the general practice mentioned above. I have found median method most beneficial for fare_amount variable. And As we know that passenger_count is a categorical variable we used mode method to impute missing values of this variable.

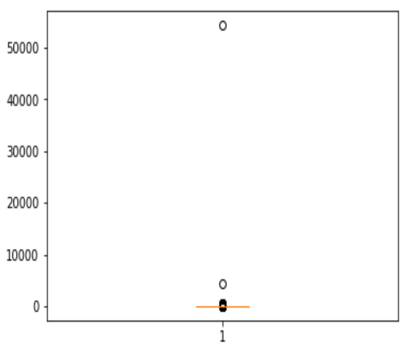
Outlier Analysis

In statistics, an outlier is defined as a data point that differs significantly from other observations. Outlier analysis is a technique to find these points. Outlier analysis can only be done on a numerical variable. Causes of Outliers

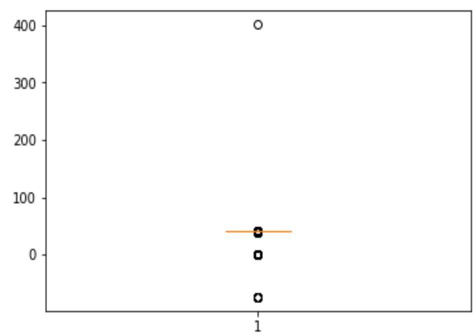
- Poor data quality/contamination
- Low-quality measurements, malfunctioning equipment, manual error
- Correct but exceptional data

In our case, we first analyzed location variables i.e latitude and longitude. As we know the fare of the cab may change from location to location hence we considered all the locations of train dataset that were outside the locations of test dataset as outlier locations. And then we removed these locations from the train dataset, for example, look at the following R code.

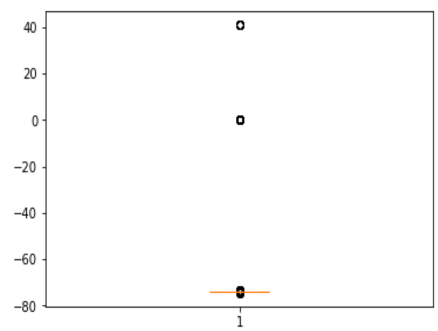
Outliers in Python



1. **Fare_amount**

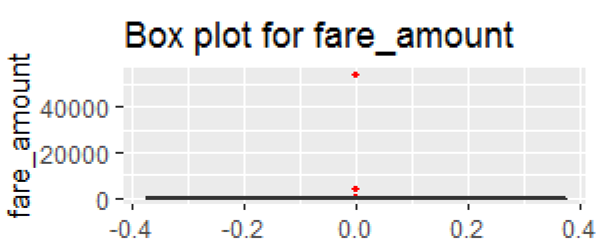


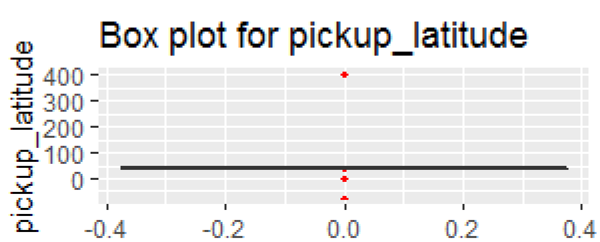
2. Pickup_longitude

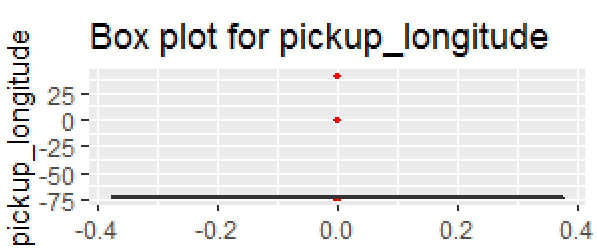


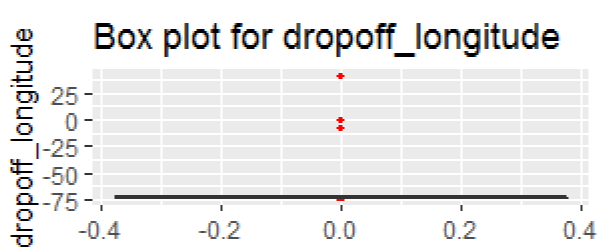
3. Pickup_latitude

Outliers in R

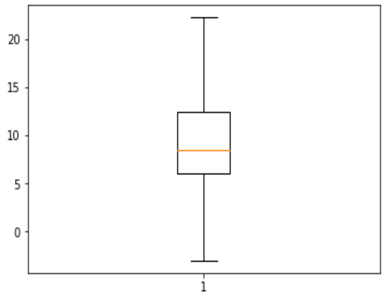


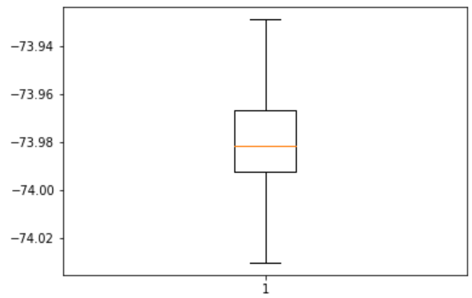


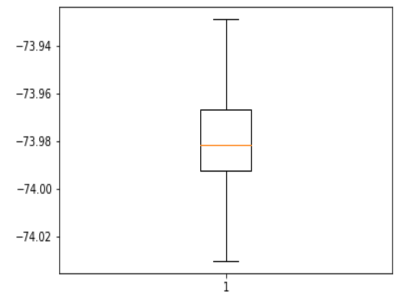




As we can see above there are outliers in our data, using boxplot method I have found outliers in both train and test data. I used cap filling method to replace outliers. After using cap filling method we can see that outliers has been successfully removed as below.







1. **Fare_amount**

2. Pickup_longitude

3. Pickup_latitude

3.2- Feature Engineering

Feature engineering is the science (and art) of extracting more information from existing data, not adding any new data to it, but making the data more meaningful and usable.

1. For ‘Latitude and longitude’ variable:

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.

In our Project we derived a new variable trip distance from given pickup and drop off latitudes and longitudes using haversine formula.

The haversine formula determines the great-circle distance between two points on a sphere given their longitudes and latitudes. Important in navigation, it is a special case of a more general formula in spherical trigonometry, the law of haversines, that relates the sides and angles of spherical triangles.

```
def trip_distance(lon1, lat1, lon2, lat2):
    lon1, lat1, lon2, lat2 = map(np.radians, [lon1, lat1, lon2, lat2])
    dlon = lon2 - lon1
    dlat = lat2 – lat1
    a = np.sin(dlat/2.0)**2 + np.cos(lat1) * np.cos(lat2) *
    np.sin(dlon/2.0)**2
    c = 2 * np.arcsin(np.sqrt(a))
    km = 6371 * c
    return km
```

Here, we have used np.radians to convert latitude and longitudes into radian. Where 6371 is nothing but the radius of the earth in kilometers.

2. For ‘pickup_datetime’ variable:

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.

3. For ‘passenger_count’ variable: 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.

4. For ‘fare_amount’ variable
As we know we have some negative values in fare amount so we have to remove those values.

So our new extracted variables are:

- fare_amount
- pickup_longitude
- pickup_latitude
- dropoff_longitude
- dropoff_latitude
- pickup_datetime
- year
- month
- day_of_month
- day_of_week
- hour
- passenger_count

- trip_distance

Now as we know that all above variables are of now use so we will drop the redundant variables:

- pickup_longitude
- pickup_latitude
- dropoff_longitude
- dropoff_latitude
- pickup_datetime

Now only following variables we will use for further steps:

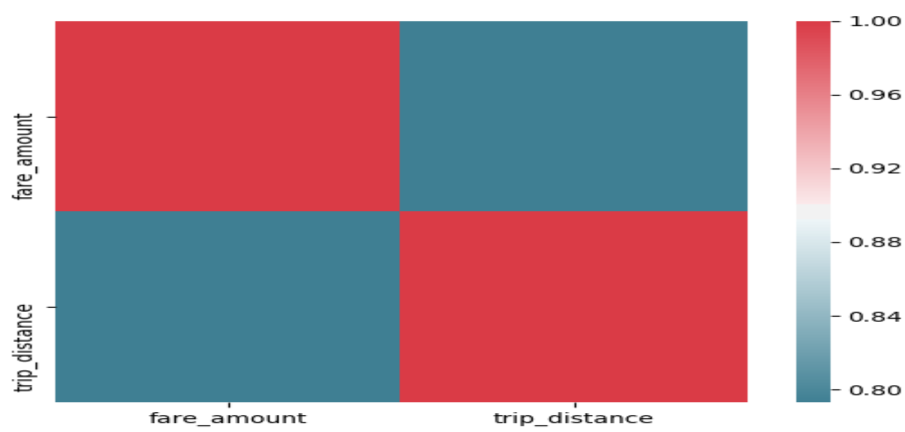
	fare_amount	passenger_count	year	Month	Date	Day of Week	Hour	distance
0	4.5	1.0	2009.0	6.0	15.0	0.0	17.0	1.030764
1	16.9	1.0	2010.0	1.0	5.0	1.0	16.0	8.450134
2	5.7	2.0	2011.0	8.0	18.0	3.0	0.0	1.389525
3	7.7	1.0	2012.0	4.0	21.0	5.0	4.0	2.799270
4	5.3	1.0	2010.0	3.0	9.0	1.0	7.0	1.999157
5	12.1	1.0	2011.0	1.0	6.0	3.0	9.0	3.787239
6	7.5	1.0	2012.0	11.0	20.0	1.0	20.0	1.555807
8	8.9	2.0	2009.0	9.0	2.0	2.0	1.0	2.849627
9	5.3	1.0	2012.0	4.0	8.0	6.0	7.0	1.374577
10	5.5	3.0	2012.0	12.0	24.0	0.0	11.0	0.000000

3.3- Feature Selection

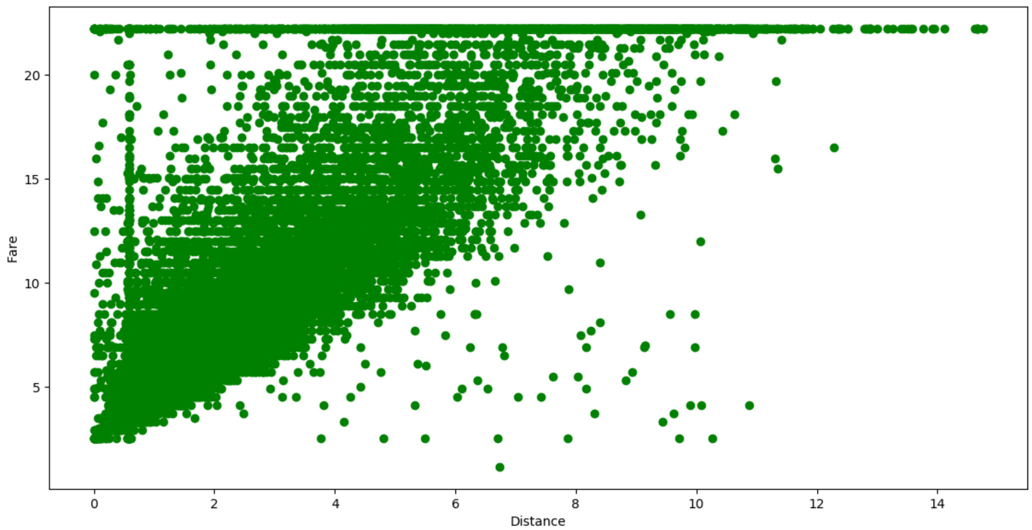
In this step we would allow only to pass relevant features to further steps. We remove irrelevant features from the dataset. We do this by some statistical techniques, like we look for features which will not be helpful in predicting the target variables. In this dataset we have to predict the fare_amount.
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. We can see that in correlation plot faded colour like skin colour indicates that 2 variables are highly correlated with each other. As the colour fades correlation values increases.

From below correlation plot we see that:

- 'fare_amount' and 'trip_distance' are very highly correlated with each other.
- As fare_amount is the target variable and 'trip_distance' is independent variable we will keep 'trip_distance' because it will help to explain variation in fare_amount.



Relationship between fare amount and trip distance



Fare vs trip_distance

Analysis of Variance(Anova) Test –

I. It is carried out to compare between each group in a categorical variable.

II. 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.

Below is the anova analysis table for each categorical variable:

Anova analysis in Python

```
In [95]: aov_table
Out[95]:
                          df        sum_sq  ...          F        PR(>F)
C(day_of_week)           6.0    288.243818  ...   1.656154  1.274137e-01
C(passenger_count)       5.0    588.750670  ...   4.059321  1.104556e-03
C(day_of_month)         30.0    756.047007  ...   0.868799  6.717802e-01
C(year)                  6.0  10467.095293  ...  60.140477  5.279121e-74
C(hour)                 23.0   2898.703208  ...   4.344781  1.597918e-11
Residual             15794.0 458142.092654  ...        NaN           NaN
```

Anova Analysis in R

```
> summary(aov_results)
                   Df Sum Sq Mean Sq F value   Pr(>F)    
passenger_count     5    265    52.9   3.730  0.00224 ** 
hour               23    929    40.4   2.847 6.17e-06 ***
weekday             6     80    13.3   0.938  0.46581    
month              11    652    59.3   4.176 3.40e-06 ***
year                6   5522   920.4  64.851  < 2e-16 ***
day                30    573    19.1   1.345  0.09865 .  
Residuals       13025 184850    14.2                     
---
```

So, from the anova result we have dropped weekday and month day from our data.

<u>**Multicollinearity**</u>– 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.

I. Multicollinearity increases the standard errors of the coefficients.
II. Increased standard errors in turn means that coefficients for some independent variables may be found not to be significantly different from 0.
III. 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.
IV. 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.
V. And if the VIF goes above 10, you can assume that the regression coefficients are poorly estimated due to multicollinearity.

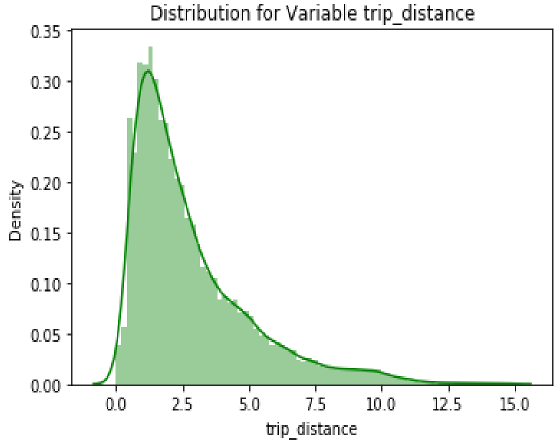
We have checked for multicollinearity in our Dataset and all VIF values are below 5.

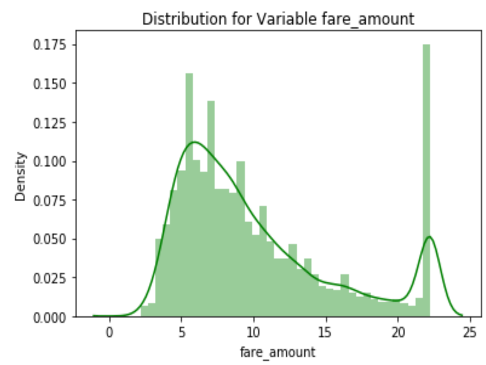
3.4-Feature Scaling

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. Normalisation 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 'trip_distance' is not distributed normally so we had chosen normalization over standardization.
- We have checked variance for each column in dataset before Normalisation
- High variance will affect the accuracy of the model. So, we want to normalise that variance. Graphs based on which standardization was chosen:
Note: It is performed only on Continuous variables.
distplot() for 'trip_distance' feature before normalization:





Splitting train and Validation Dataset

a) We have used sklearn's train_test_split() method to divide whole Dataset into train and validation datset.

b) 20% is in validation dataset and 80% is in training data.

c) 12692 observations in training and 3173 observations in validation dataset.

d) We will test the performance of model on validation datset.

e) The model which performs best will be chosen to perform on test dataset provided along with original train dataset.

f) X_train y_train--are train subset.

g) X_test y_test--are validation subset.

4. Modeling

4.1- Model Selection

Once completing data cleaned next process is model selection it is based on problem statement. In car fare prediction problem statement understood that it comes under supervised machine learning because it has both input and output variables and its regression problem as out target variable is fare amount which is of numeric / continuous type. So, we can consider linear regression, Decision Tree, Random Forest etc.,

In our project used three models viz., linear regression, Decision Tree, Random Forest.

Error matrix chosen for the given problem statement is Root Mean Squared Error (RMSE) and R2(R-Squared). Before building an any model we divided the preprocessed train_cab data set in to train and test set. Data was divided into 80:20 ratio, 80% of data was used as 'train' set and rest of the 20% was used as 'test' set. The training set is used to fit the model and the test set is used to estimate the model prediction accuracy.

4.2 Linear Regression

Linear Regression is a supervised machine learning algorithm where the predicted output is continuous and has a constant slope. It's used to predict values within a continuous range, (e.g. sales, price) rather than trying to classify them into categories (e.g. cat, dog).

Linear Regression, unlike other algorithms, stores information in terms of coefficients. It is a statistical model. We cannot use this for classification. It describes relationship among variables.

our aim is – we always want a model with low RMSE value i.e. minimum calculated errors and high R square value i.e. the independent variables should have maximum potential to explain about the target variable.

Python Result- In our project, we got RMSE as 3.07864 and R square as 0.67534. we are rejecting this model as RMSE is high and R square is low when compared with all other models.

R Result- we got RMSE as 2.0735 and R square as 0.7129. we are accepting this model as RMSE is low and R square is high when compared with all other models.

4.3 Decision Tree

Decision Trees am a type of Supervised Machine Learning (that is you explain what the input is and what the corresponding output is in the training data) where the data is continuously split according to a certain parameter.

The tree can be explained by two entities, namely decision nodes and leaves. The leaves are the decisions or the final outcomes. And the decision nodes are where the data is split.

My aim is – I always want a model with low RMSE value i.e. minimum calculated errors and high R square value i.e. the independent variables should have maximum potential to explain about the target variable.

Python Result- In My project, I get RMSE as 2.8818 and R square as 0.7156. we are accepting this model as RMSE is low and R square is high when compared with all other models.

R Result- we got RMSE as 2.27214 and R square as 0.69534. we are rejecting this model as RMSE is high and R square is low when compared with all other models.

4.4 Random Forest

The Random Forest is a model made up of many decision trees. Rather than just simply averaging the prediction of trees (which I could call a “forest”), this model uses two key concepts that gives it the name *random*:

- Random sampling of training data points when building trees
- Random subsets of features considered when splitting nodes

The random forest combines hundreds or thousands of decision trees, trains each one on a slightly different set of the observations, splitting nodes in each tree considering a limited number of the features. The final predictions of the random forest am made by averaging the predictions of each individual tree.

My aim is – I always want a model with low RMSE value i.e. minimum calculated errors and high R square value i.e. the independent variables should have maximum potential to explain about the target variable.

Python Result- In My project, I get RMSE as 2.9848 and R square as 0.69485. we are rejecting this model as RMSE is high and R square is low when compared with all other models.

R Result- we got RMSE as 2.272141 and R square as 0.68721. we are rejecting this model as RMSE is high and R square is low when compared with all other models.

5. Conclusion

5.1 Model Evaluation

We always need a metric to evaluate the work we did. So, the same way, after we developed my models, we need a metric to validate the model we developed. There am many metrics to evaluate, even, we have different metrics for classification problem and different metrics for regression problems.
For classification problems, we have metrics like:

- Confusion Matrix.
- Accuracy.
- Recall.
- Specificity.

For regression problems, we have metrics like:

- MSE.
- RMSE.
- MAPE.
- R square.

we are choosing RMSE and R square for our project. **Why RMSE over MAPE?**

Because, in RMSE, as the errors are squared before they are averaged, the RMSE gives a relatively high weightage to large errors, another reason is, RMSE penalizes large errors. For the above-mentioned reasons, I choose RMSE over MAPE.

5.2 <u>Root Mean Square Error (RMSE) & R square</u>

We are going to use RMSE and R square as our error metrics to evaluate our models.

RMSE – Simply said, it is the sum of calculated errors.

R square – Simply defined, correlation of original and predicted values.

RMSE And R-2 value that we got in python is as follows:-

Model Name	R-squared	RMSE
Linear Regression	0.67538	3.07864
Deciosion Tree	0.71556	2.88182
Random Forest	0.69485	2.98488

RMSE And R-2 value that we got in R is as follows:-

Model Name	R-squared	RMSE
Linear Regression	0.7129	2.07351
Deciosion Tree	0.69534	2.24598
Random Forest	0.68721	2.27214

5.3 Model Selection

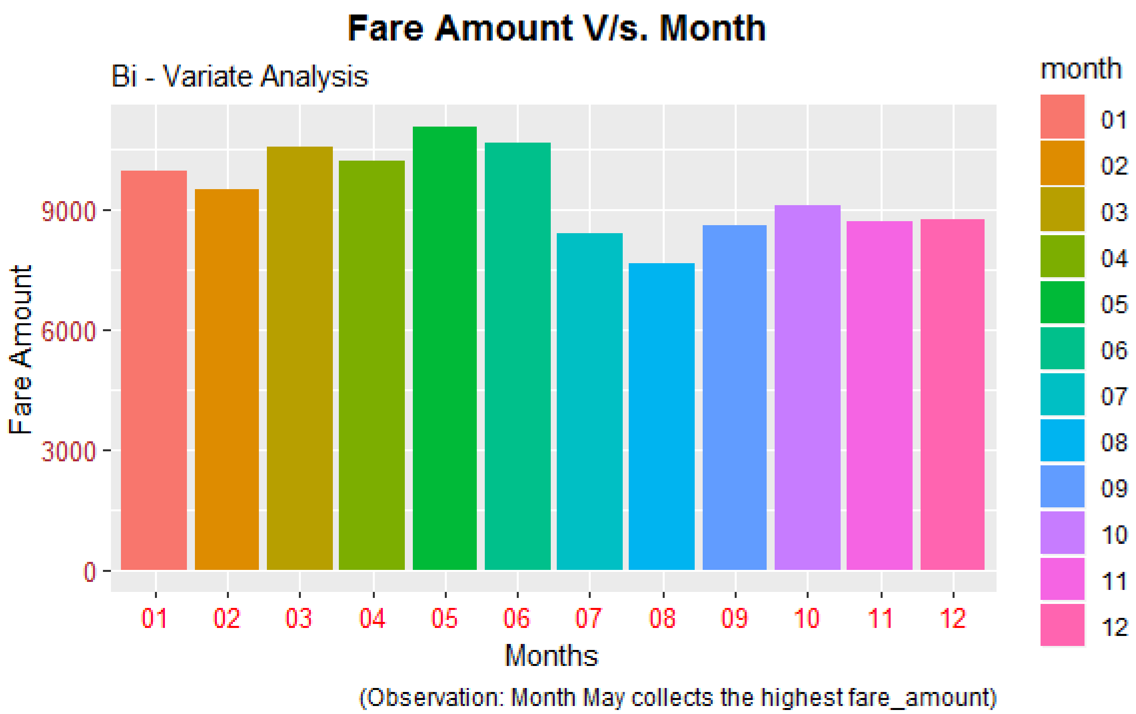
Finally, it's our Model selection time. we developed three models. Linear Regression, Decision Tree and Random Forest

From the results we have chosen decision tree In python and Linear regression model in R, because comparatively these model were perfoming well then others.

After selection of the model, we had to predict fare amount on test data, so we used python's decision tree and R's linear regression model to predict fares.

5.4. Visualizations

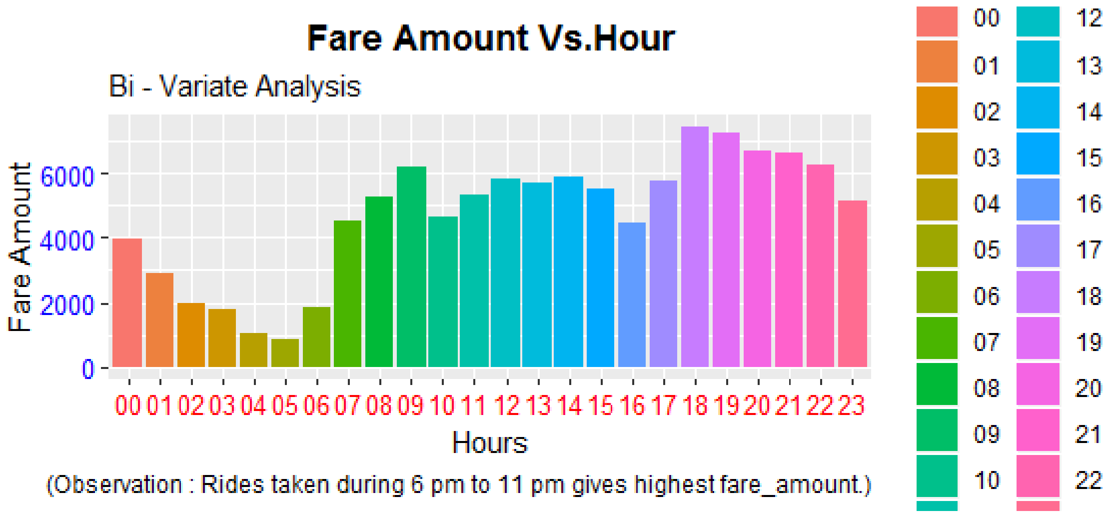
1. Visualization of the distribution of fare_amount vs Month



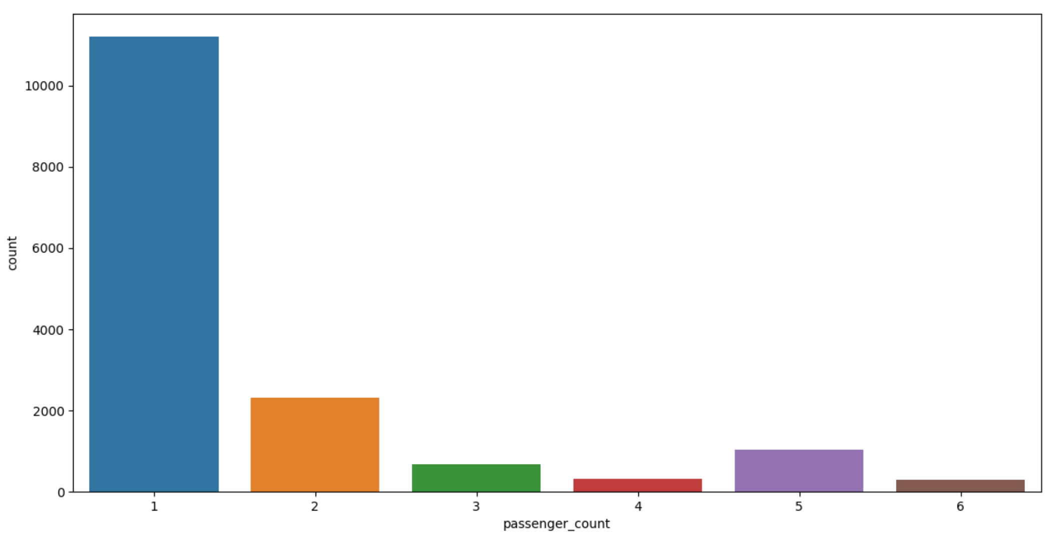
March to June month had very good fare amounts compared to other months. Lowest fare amount found in august month.

2. Visualization on distribution of fare_amount over Hour

- During hours 6 PM to 11PM the frequency of cab boarding is very due to peak hours
- Fare prices during 2PM to 8PM is bit high compared to all other time might be due to high demands.

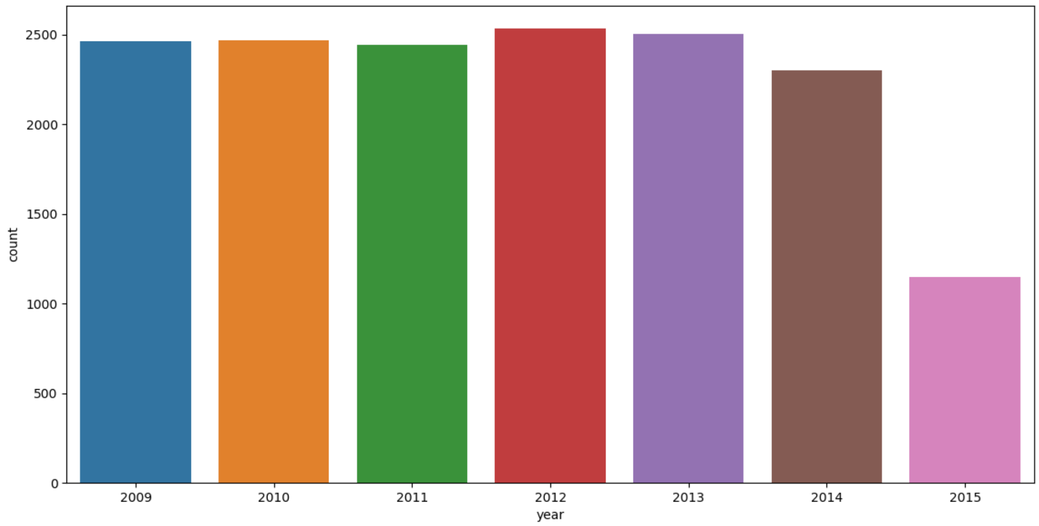


3. Visualization on the count of passengers



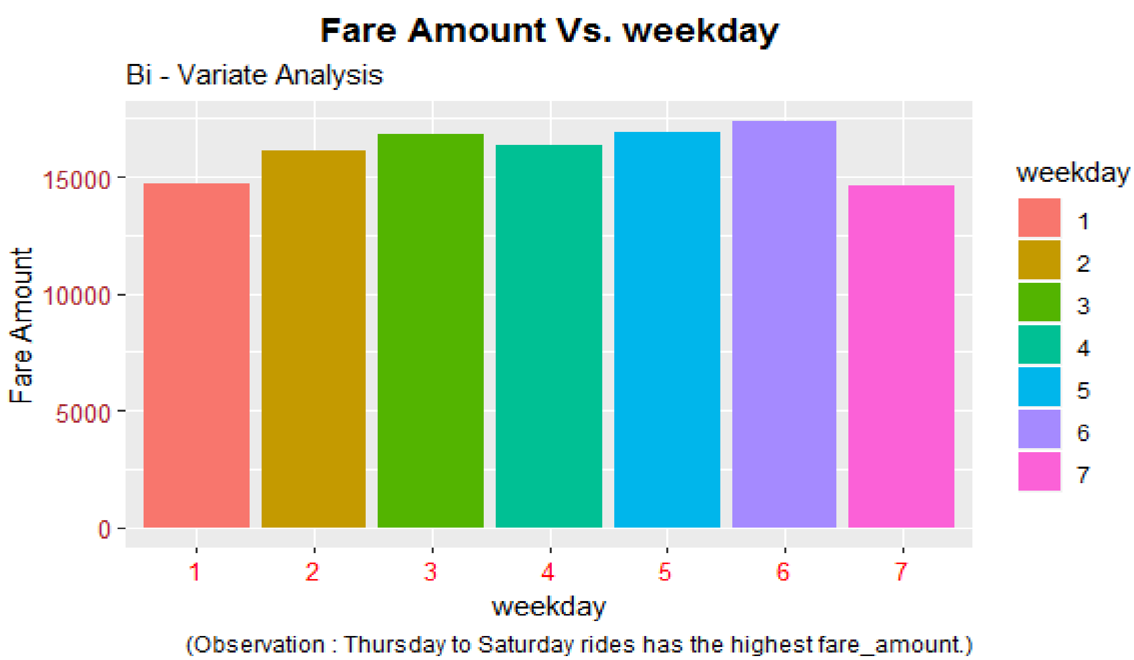
As we can see in the above bar graph that single passengers booked a cab for most numbers of the time whereas family booking was least.

4. Visualization on distribution of year



5. Week Day and fare

Cab fare is high on Friday, Saturday and Monday, may be during weekend and first day of the working day they charge high fares because of high demands of cabs.



Appendix A – Python Script

```
    # Cab Fare Project Python
#importing Libraries
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from collections import Counter
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor


#from fancyimpute import KNN

# Setting up new woorking directory
os.chdir("C:\\Users\\pc\\Desktop\\R\\projects\\Cab care project")
#Checking current directory
os.getcwd()

#importing data
train = pd.read_csv("train_cab.csv")
test = pd.read_csv("test.csv")
# Checking data dimenssions
train.shape
test.shape
# No. of rows in data
train.shape[0]
test.shape[0]
```

```
# No. of columns
train.shape[1]
test.shape[1]
# name of columns
list(train)
list(test)
# data detail
train.info()
test.info()

# Converting data into required format
data = [train, test]
for i in data:
   i['pickup_datetime'] = pd.to_datetime(i['pickup_datetime'], errors='coerce')

#train['fare_amount'].astype(float)
train['fare_amount'] = np.where(train['fare_amount']== "430-", 430,
train['fare_amount'] )
train['fare_amount'] = train['fare_amount'].astype(float)

############################################## Missing Value Analysis
train.isnull().sum()
test.isnull().sum()     # 0 missing value

# we are getting 1 missing value in pickup_datetime, we will drop that
observation.
np.where(train['pickup_datetime'].isnull())
train = train.dropna(subset = ['pickup_datetime'], how = 'all')

# filling its values by mode, Becouse passenger_count is categorical variable
train['passenger_count'] =
train['passenger_count'].fillna(train['passenger_count'].mode()[0])

# We will convert Passenger_count variable into object type, becouse it is
categorical variable
train['passenger_count']=train['passenger_count'].round().astype('object').astyp
e('category')
test['passenger_count']=test['passenger_count'].round().astype('object').astype(
'category')
```

```
missing_val = pd.DataFrame(train.isnull().sum())
#Reset index
missing_val = missing_val.reset_index()
# Reanaming variables
missing_val = missing_val = missing_val.rename(columns = {'index': 'Variables',
0: 'Missing_percentage'})
#Calculate percentage
missing_val['Missing_percentage'] =
(missing_val['Missing_percentage']/len(train))*100


# Best method
#actual value= 10
#mean=  15.041185637700874
#median= 8.5

# we will a value to replace na in fare_amount variable
train['fare_amount'].loc[100] = np.nan
#Mean
train['fare_amount'] = train['fare_amount'].fillna(train['fare_amount'].mean())
#Median
train['fare_amount'] = train['fare_amount'].fillna(train['fare_amount'].median())
# we have find median as best method to fill null values
train.fillna(value = train.median(), inplace= True)

################################ Outliers analysis
# we will use cap filling to replace outliers

plt.boxplot(train["fare_amount"])
plt.boxplot(train["pickup_latitude"])
plt.boxplot(train['pickup_longitude'])
plt.boxplot(train["dropoff_latitude"])
plt.boxplot(train['dropoof_longitude'])

def outlier_detect(df):
   for i in df.describe().columns:
      q1=df.describe().at["25%",i]
      q3=df.describe().at["75%",i]
      IQR=(q3-q1)
      lb=(q1-1.5*IQR)
```

```
        ub=(q3+1.5*IQR)
        x=np.array(df[i])
        p=[]
        for j in x:
            if j<lb:
              p.append(lb)
            elif j>ub:
              p.append(ub)
            else:
              p.append(j)
        df[i]=p
    return(df)

outlier_detect(train)
outlier_detect(test)

#################################### Feature Engineering
#1. Feautre Engineering for fare_amount variable

#Removing values which are not within desired range(outlier) depending upon basic understanding of dataset.
# In fare_amount values which are lesser then 1 dont have any significance in data
Counter(train['fare_amount']<1)
train = train.drop(train[train['fare_amount']<1].index, axis=0)

#2. Feature engineering for pickup_datetime variable

#lets create a function to get important features from pickup_datetime variable in train and test datasets
#train = pd.concat((pickup_datetime, train), axis= 1)

data = [train,test]
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_month"] = i["pickup_datetime"].apply(lambda row: row.day)
    i["day_of_week"] = i["pickup_datetime"].apply(lambda row: row.dayofweek)
    i["hour"] = i["pickup_datetime"].apply(lambda row: row.hour)
```

```
#3. Feature engineering for passenger_count variable
train['passenger_count']=train['passenger_count'].astype('int')

test['passenger_count'].unique()
train['passenger_count'].unique()

train['passenger_count']=train['passenger_count'].round().astype('object')
train.std()

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

#so 20 observations of passenger_count is consistenly above from 6,7,8,9,10
passenger_counts, let's check them.
Counter(train['passenger_count']>6)
#Also we need to see if there are any passenger_count<1
Counter(train['passenger_count']<1)

#passenger_count variable conatins values which are equal to 0.
#we will remove those 0 values.
#Also, We will remove 20 observation which are above 6 value because a cab
cannot hold these number of passengers.

train = train.drop(train[train['passenger_count']>6].index, axis=0)
train = train.drop(train[train['passenger_count']<1].index, axis=0)
train.std()
train['passenger_count']=train['passenger_count'].astype('int')
train['passenger_count'].unique()

#4. Feature engineering for latitude and logitude

#Latitudes range from -90 to 90.Longitudes range from -180 to 180.
#Removing which does not satisfy these ranges
print('pickup longitude above
180={}'.format(sum(train['pickup_longitude']>180)))
print('pickup_longitude below -180={}'.format(sum(train['pickup_longitude']<-
180)))
print('pickup_latitude above 90={}'.format(sum(train['pickup_latitude']>90)))
```

```
print('pickup_latitude below -90={}'.format(sum(train['pickup_latitude']<-90)))
print('dropoff_longitude above
180={}'.format(sum(train['dropoff_longitude']>180)))
print('dropoff_longitude below -180={}'.format(sum(train['dropoff_longitude']<-
180)))
print('dropoff_latitude below -90={}'.format(sum(train['dropoff_latitude']<-90)))
print('dropoff_latitude above 90={}'.format(sum(train['dropoff_latitude']>90)))


#There's only one outlier which is in variable pickup_latitude.So we will remove
it.
#train = train.drop(train[train['pickup_latitude']>90].index, axis=0)

#Also we will see if there are any values equal to 0.
for i in
['pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitude']:
    print(i,'equal to 0={}'.format(sum(train[i]==0)))

#there are values which are equal to 0. we will remove them.
for i in
['pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitude']:
    train = train.drop(train[train[i]==0].index, axis=0)

train.shape
train.info()


### Now let's calculate trip distance from picup and dropoff latitude and
longitude
def trip_distance(lon1, lat1, lon2, lat2):
    lon1, lat1, lon2, lat2 = map(np.radians, [lon1, lat1, lon2, lat2])
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = np.sin(dlat/2.0)**2 + np.cos(lat1) * np.cos(lat2) * np.sin(dlon/2.0)**2
    c = 2 * np.arcsin(np.sqrt(a))
    km = 6371 * c
    return km

train['trip_distance']=trip_distance(train['pickup_longitude'],train['pickup_latitu
de'],
```

```
                        train['dropoff_longitude'],train['dropoff_latitude'])

test['trip_distance']=trip_distance(test['pickup_longitude'],test['pickup_latitude'
],
                        test['dropoff_longitude'],test['dropoff_latitude'])

###we will remove the rows whose distance value is zero
Counter(train['trip_distance']==0)
train = train.drop(train[train['trip_distance']== 0].index, axis=0)
train.shape

Counter(test['trip_distance']==0)
test = test.drop(test[test['trip_distance']== 0].index, axis=0)
train.shape

#Now we will plot a scatter plot
plt.xlabel("trip Distance")
plt.ylabel("Fare Amount")
plt.scatter(x=train['trip_distance'],y=train['fare_amount'])
plt.title("Trip Distance vs Fare Amount")

train.describe()

df=train.copy()
# train=df.copy()

# # Data Visualization :

# Visualization of following:
#
# 1. Number of Passengers effects the the fare
# 2. Pickup date and time effects the fare
# 3. Day of the week does effects the fare
# 4. Distance effects the fare

plt.figure(figsize=(20,10))
sns.countplot(train['year'])
# plt.savefig('year.png')
plt.figure(figsize=(20,10))
sns.countplot(train['month'])
```

```
# Count plot on passenger count
plt.figure(figsize=(15,7))
sns.countplot(x="passenger_count", data=train)

#Relationship beetween number of passengers and Fare

plt.figure(figsize=(15,7))
plt.scatter(x=train['passenger_count'], y=train['fare_amount'], s=10)
plt.xlabel('No. of Passengers')
plt.ylabel('Fare')
plt.show()


#Relationship between date and Fare
plt.figure(figsize=(15,7))
plt.scatter(x=train['day_of_month'], y=train['fare_amount'], s=10)
plt.xlabel('Date')
plt.ylabel('Fare')
plt.show()
# day_of_month is not much significance in dataset.
#
plt.figure(figsize=(20,10))
sns.countplot(train['hour'])
plt.show()


# Lowest cabs at 5 AM and highest at and around 7 PM i.e the office rush hours

#Relationship between Time and Fare
plt.figure(figsize=(15,7))
plt.scatter(x=train['hour'], y=train['fare_amount'], s=10)
plt.xlabel('Hour')
plt.ylabel('Fare')
plt.show()

# From the above plot We can observe that the cabs taken at 7 am and 23 Pm
are the costliest.
```

```
# Hence we can assume that cabs taken early in morning and late at night are
costliest

#impact of Day of week on the number of cab rides
plt.figure(figsize=(15,7))
sns.countplot(x="day_of_week", data=train)


# Observation :
# The day of the week does not seem to have much influence on the number of
cabs ride


#We will remove the variables which were used to feature engineer new
variables
train=train.drop(['pickup_datetime','pickup_longitude', 'pickup_latitude',
'dropoff_longitude', 'dropoff_latitude'],axis=1)

test=test.drop(['pickup_datetime','pickup_longitude', 'pickup_latitude',
'dropoff_longitude', 'dropoff_latitude'],axis=1)


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

# Calculation of correlation between numerical variables
num_var=['fare_amount','trip_distance']
df_num = train.loc[:,num_var]

corr = df_num.corr()
print(corr)

# plotiing the heatmap
f, ax = plt.subplots(figsize=(7, 5))
sns.heatmap(corr, mask=np.zeros_like(corr, dtype=np.bool),
cmap=sns.diverging_palette(220, 10, as_cmap=True),square=True, ax=ax)
plt.show()
## we can say from the plot that both variable are correlated to each other

###################       Anova test    ################################
from statsmodels.formula.api import ols
```

```
import statsmodels.api as sm
model = ols('fare_amount ~
C(day_of_week)+C(passenger_count)+C(day_of_month)+C(year)+C(hour)',data=t
rain).fit()
aov_table = sm.stats.anova_lm(model)
aov_table
# we are getting two values day_of_week and day_of_month values higher then
0.05, so will drop them

######################     Multicollinearity Test     ################
#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.

# Detecting and Removing Multicollinearity
# use statsmodels library to calculate VIF
# Import VIF function from statmodels Library
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Get variables for which to compute VIF and add intercept term:

X = train[['passenger_count', 'year',
           'month', 'day_of_month', 'day_of_week',
'hour','trip_distance']].dropna() #subset the dataframe
X ['Intercept'] = 1

# Compute and view VIF:

vif = pd.DataFrame()         # Create an empty dataframe
vif["Variables"] = X.columns   # Add "Variables" column to empty dataframe
vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

# View results using print
print(vif)

# from the results,we will just drop 2 variables=
```

```
train=train.drop(['day_of_week','day_of_month'], axis=1)
test=test.drop(['day_of_week','day_of_month'], axis=1)


#Normality check of training data is uniformly distributed or not-

for i in ['fare_amount', 'trip_distance']:
    print(i)
    sns.distplot(train[i],bins='auto',color='green')
    plt.title("Distribution for Variable "+i)
    plt.ylabel("Density")
    plt.show()

# data is normally distributed

############################################ Model development

# divided into independent (x) and dependent variables (y)
x= train.iloc[:,1:6]
x.shape
y =train.iloc[:,0]
y

 # Splitting the data into training and test sets
x_train, x_test,y_train, y_test =train_test_split(x,y,test_size=.2, random_state
=100)
print(train.shape, x_train.shape, x_test.shape,y_train.shape,y_test.shape)

### Linear Regression

# linear regression using sklearn

lm =LinearRegression()
lm= lm.fit(x_train,y_train)
# coefficients
lm.coef_
# To store coefficients in a data frame along with their respective independent
variables
coefficients=pd.concat([pd.DataFrame(x_train.columns),pd.DataFrame(np.trans
pose(lm.coef_))], axis = 1)
```

```
print(coefficients)

# intercept
lm.intercept_

#prediction on train data
pred_train = lm.predict(x_train)

#prediction on test data
pred_test = lm.predict(x_test)

##calculating RMSE for test data
RMSE_test = np.sqrt(mean_squared_error(y_test, pred_test))

##calculating RMSE for train data
RMSE_train= np.sqrt(mean_squared_error(y_train, pred_train))

print("Root Mean Squared Error For Training data = "+str(RMSE_train))
#3.2878183375029786
print("Root Mean Squared Error For Test data = "+str(RMSE_test))
#3.078645997656653

#calculate R^2 for train data
r2_score(y_train, pred_train) #0.6393978978272477
# r2 foe test data
r2_score(y_test, pred_test) #0.675381984761757

###### Decision tree

#Decision tree for regression
fit_DT = DecisionTreeRegressor(max_depth=6).fit(x_train, y_train)

#prediction on train data
pred_DT = fit_DT.predict(x_train)
#Apply model on test data
predictions_DT = fit_DT.predict(x_test)

#calculate R^2 for train data
r2_score(y_train, pred_DT) #0.7030262920691919
# r2 for test data
```

```
r2_score(y_test, predictions_DT) #0.7155614076403296

# RMSE for both data
RMSE_train=np.sqrt(mean_squared_error(pred_DT,y_train))
RMSE_test=np.sqrt(mean_squared_error(predictions_DT,y_test))

print("RMSE of train data = ", RMSE_train)  #2.983683049406448
print("RMSE of test data = ",RMSE_test)   # 2.881825667387774


#### Random Forest

fit_RF = RandomForestRegressor(n_estimators = 500).fit(x_train,y_train)

#prediction on train data
pred_RF = fit_RF.predict(x_train)
#Apply model on test data
predictions_RF = fit_RF.predict(x_test)

#calculate R^2 for train data
r2_score(y_train, pred_RF) # 0.9529716271574351
# r2 foe test data
r2_score(y_test, predictions_RF)  # 0.6948525495517299

# RMSE for both data
RMSE_train=np.sqrt(mean_squared_error(pred_RF,y_train))
RMSE_test=np.sqrt(mean_squared_error(predictions_RF,y_test))

print("RMSE of train data = ", RMSE_train)  #1.187336073286684
print("RMSE of test data = ",RMSE_test)      #2.9848899080585043


# ### As we got best Accuracy with Decision Tree Model we will use this Model
to predict Fare

# test data
test.describe()
test.shape
# prediction on test data using decision tree mode;
predicted_fare=fit_DT.predict(test)
```

```
# Saving predicted fare in test data
test['predicted_fare']=predicted_fare

test.head(10)
# saving test data in our memory
test.to_csv("test_predicted.csv",index=False)
```

Appendix B – R Script

```
# Cab Fare prediction

rm(list=ls())
setwd("C:\\Users\\pc\\Desktop\\R\\projects\\Cab care project")

#Load Libraries
x = c("ggplot2", "corrgram", "DMwR", "caret", "randomForest", "unbalanced",
"dummies", "e1071", "Information",
    "MASS", "rpart", "gbm", "ROSE", 'sampling', 'DataCombine', 'inTrees', 'dplyr')

#install.packages(x)
lapply(x, require, character.only = TRUE)
rm(x)


# The details of data attributes in the dataset are as follows:
# 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.

# loading datasets
train = read.csv("train_cab.csv", header = T)
test = read.csv("test.csv")
```

```
# Structure of data
str(train)
str(test)
summary(train)
summary(test)
head(train,5)
head(test,5)
# Check class of the data
class(train)

#Check the dimensions(no of rows and no of columns)
dim(train)

#Check names of dataset(no need of renaming variables)
names(train)


# Let's Check for data types of train data:
sapply(train, class)
str(train)

# Let's Check for data types of train data:
sapply(train, class)
str(train)

#############    Exploratory Data Analysis          #######################

# In train data observed that fare_amount and pickup_datetime variables are of
Factor type.
# and passenger_count variable of numeric type
# So, Need to convert fare_amount datatype to 'numeric' & pickup_datetime
data type to 'datatime' format.
# Passenger_count datatype to integer datatype

train$fare_amount = as.numeric(as.character(train$fare_amount))
class(train$fare_amount)# Data type After conversion


# Convert Passeneger_count data type from numeric to integer type:
```

```
class(train$passenger_count)
train$passenger_count = as.integer(train$passenger_count)
class(train$passenger_count)

# Convert pickup_datetime data type from factor to datetime

train$pickup_datetime <- as.POSIXct(strptime(train$pickup_datetime, "%Y-%m-
%d %H:%M:%S"))
test$pickup_datetime <- as.POSIXct(strptime(test$pickup_datetime, "%Y-%m-
%d %H:%M:%S"))

str(train$pickup_datetime)

head(train)

summary(train) # There is one observation in pickup_datetime which is not in
correct format need to delete it.

# Check observations which are not formatted  correctly in pickup_datetime
variable

train[is.na(strptime(train$pickup_datetime,format="%Y-%m-%d %H:%M:%S")),]

# Remove observation which are not in correct datetime format :In our case 1
observation found.

sum(is.na(train$pickup_datetime))
train <- train[-c(1328),]
dim(train)



#############                    Missing Value Analysis              #############
missing_val = data.frame(apply(train,2,function(x){sum(is.na(x))}))
missing_val$Columns = row.names(missing_val)
names(missing_val)[1] =  "Missing_percentage"
missing_val$Missing_percentage =
(missing_val$Missing_percentage/nrow(train)) * 100
missing_val = missing_val[order(-missing_val$Missing_percentage),]
row.names(missing_val) = NULL
```

```
missing_val = missing_val[,c(2,1)]
# percentage of missing values is lower then 30 %.

##As passenger_count is categorical variable we will impute it using mode
###Mode Method
train$passenger_count[is.na(train$passenger_count)]
=as.data.frame(mode(train$passenger_count))

df=train
#train=df

#actual value=10.9
#mean=15.0526
#median=8.5

train[1000,1] = NA

## we will impute missing value of fare_amount by using mean or median
method
####Mean Method
train$fare_amount[is.na(train$fare_amount)] = mean(train$fare_amount, na.rm
= T)

####Median Method
train$fare_amount[is.na(train$fare_amount)] = median(train$fare_amount,
na.rm = T)


sum(is.na(train))
str(train)
summary(train)

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

# #Plot boxplot to visualize Outliers

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

```
cnames =
colnames(train[,c("fare_amount","pickup_longitude","pickup_latitude","dropof
f_longitude","dropoff_latitude")])

for (i in 1:length(cnames))
{
  assign(paste0("gn",i), ggplot(aes_string(y = cnames[i]), data = train)+
         stat_boxplot(geom = "errorbar", width = 0.5) +
         geom_boxplot(outlier.colour="red", fill = "grey" ,outlier.shape=18,
                      outlier.size=1, notch=FALSE) +
         theme(legend.position="bottom")+
         labs(y=cnames[i])+
         ggtitle(paste("Box plot for",cnames[i])))
}
gridExtra::grid.arrange(gn1,gn3,gn2,gn4,ncol=2)

# dropping the outliers

for (i in cnames)
{
  val = train[,i][train[,i] %in% boxplot.stats(train[,i])$out]
  train = train[which(!train[,i] %in% val),]
}

#lets check the NA's  in test data
sum(is.na(test))

cname =
colnames(test[,c("pickup_longitude","pickup_latitude","dropoff_longitude","dr
opoff_latitude")])

for (i in 1:length(cname))
{
  assign(paste0("gn",i), ggplot(aes_string(y = cname[i]), data = test)+
         stat_boxplot(geom = "errorbar", width = 0.5) +
         geom_boxplot(outlier.colour="red", fill = "grey" ,outlier.shape=18,
                      outlier.size=1, notch=FALSE) +
         theme(legend.position="bottom")+
         labs(y=cname[i])+
         ggtitle(paste("Box plot for",cname[i])))
```

```
}
gridExtra::grid.arrange(gn1,gn3,gn2,gn4,ncol=2)

# dropping the outliers

for (i in cname)
{
  val = test[,i][test[,i] %in% boxplot.stats(test[,i])$out]
  test = test[which(!test[,i] %in% val),]
}


#############                    Feature Engineering              #############

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

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

#2.Passenger_count variable
for (i in seq(4,11,by=1)){
  print(paste('passenger_count above '
,i,nrow(train[which(train$passenger_count > i ),])))
}
# so some observations of passenger_count is consistenly above from 6,7,8,9,10
passenger_counts, let's check them.
train[which(train$passenger_count > 6 ),]
# Also we need to see if there are any passenger_count==0
train[which(train$passenger_count <1 ),]
nrow(train[which(train$passenger_count <1 ),])

# We will remove these observation which are above 6 value because a cab
cannot hold these number of passengers.
train = train[-which(train$passenger_count < 1 ),]
train = train[-which(train$passenger_count > 6),]
```

```
# converting passenger_count as categorical variable
train$passenger_count = as.integer(train$passenger_count)
train$passenger_count = as.factor(train$passenger_count)
test$passenger_count = as.factor(test$passenger_count)

# 3.Feature Engineering for timestamp variable
# we will derive new features from pickup_datetime variable
# new features will be year,month,day_of_week,hour
#Convert pickup_datetime from factor to date time
train$day = as.factor(format(train$pickup_datetime,"%d"))
train$weekday = as.factor(format(train$pickup_date,"%u"))# Monday = 1
train$month = as.factor(format(train$pickup_date,"%m"))
train$year = as.factor(format(train$pickup_date,"%Y"))
pickup_time = strptime(train$pickup_datetime,"%Y-%m-%d %H:%M:%S")

train$hour = as.factor(format(pickup_time,"%H"))

#Add same features to test set
test$day = as.factor(format(test$pickup_datetime,"%d"))
test$weekday = as.factor(format(test$pickup_date,"%u"))# Monday = 1
test$month = as.factor(format(test$pickup_date,"%m"))
test$year = as.factor(format(test$pickup_date,"%Y"))
pickup_time = strptime(test$pickup_datetime,"%Y-%m-%d %H:%M:%S")
test$hour = as.factor(format(pickup_time,"%H"))


train = subset(train,select = -c(pickup_datetime,pickup_datetime))
test = subset(test,select = -c(pickup_datetime,pickup_datetime))

#Longitude range----(-180 to 180)
#Latitude range----(-90 to 90)
# Check observations having pickup longitute and pickup latitute out the range
in train dataset.

train[train$pickup_longitude <  -180,]
train[train$pickup_longitude >  180,]
train[train$pickup_latitude <  -90,]
train[train$pickup_latitude >  90,]
```

```r
# Check observations having dropoff longitute and dropoff latitute out the range
in train dataset.

train[train$dropoff_longitude <  -180,]
train[train$dropoff_longitude >  180,]
train[train$dropoff_latitude <  -90,]
train[train$dropoff_latitude >  90,]


# Dropping the observations which are outof range in train dataset:

train<-  filter (train,pickup_longitude >  -180)
train<-  filter (train,pickup_longitude <  180)
train<-  filter (train,pickup_latitude >  -90)
train<-  filter (train,pickup_latitude <  90)
dim(train)

train<-  filter (train,dropoff_longitude >  -180)
train<-  filter (train,dropoff_longitude <  180)
train<-  filter (train,dropoff_latitude >  -90)
train<-  filter (train,dropoff_latitude <  90)
dim(train)

#Longitude range----(-180 to 180)
#Latitude range----(-90 to 90)
# Check observations having pickup longitute and pickup latitute out the range
in train dataset.

test[test$pickup_longitude <  -180,]
test[test$pickup_longitude >  180,]
test[test$pickup_latitude <  -90,]
test[test$pickup_latitude >  90,]


# Check observations having dropoff longitute and dropoff latitute out the range
in test dataset.

test[test$dropoff_longitude <  -180,]
test[test$dropoff_longitude >  180,]
test[test$dropoff_latitude <  -90,]
```

```
test[test$dropoff_latitude >  90,]


# Dropping the observations which are outof range in test dataset:

test<-  filter (test,pickup_longitude >  -180)
test<-  filter (test,pickup_longitude <  180)
test<-  filter (test,pickup_latitude >  -90)
test<-  filter (test,pickup_latitude <  90)
dim(test)

test<-  filter (test,dropoff_longitude >  -180)
test<-  filter (test,dropoff_longitude <  180)
test<-  filter (test,dropoff_latitude >  -90)
test<-  filter (test,dropoff_latitude <  90)


# Also we will see if there are any values equal to 0.
nrow(train[which(train$pickup_longitude == 0 ),])
nrow(train[which(train$pickup_latitude == 0 ),])
nrow(train[which(train$dropoff_longitude == 0 ),])
nrow(train[which(train$pickup_latitude == 0 ),])


### Now let's calculate trip distance from picup and dropoff latitude and
longitude
## Haversine
trip_distance = function(lon1, lat1, lon2, lat2){
  # convert decimal degrees to radians
  lon1 = lon1 * pi / 180
  lon2 = lon2 * pi / 180
  lat1 = lat1 * pi / 180
  lat2 = lat2 * pi / 180
  # haversine formula
  dlon = lon2 - lon1
  dlat = lat2 - lat1
  a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlon/2)**2
  c = 2 * atan2(sqrt(a), sqrt(1-a))
  km = 6367 * c
  return(km)
```

```
}

## Calculating trip_distance for train data
train$trip_distance=trip_distance(train$pickup_longitude,train$pickup_latitude,
                         train$dropoff_longitude,train$dropoff_latitude)


## Calculating trip_distance for test data
test$trip_distance=trip_distance(test$pickup_longitude,test$pickup_latitude,
                         test$dropoff_longitude,test$dropoff_latitude)

# We will remove the variables which were used to feature engineer new
variables
train = subset(train,select = -
c(pickup_longitude,pickup_latitude,dropoff_longitude,dropoff_latitude))
test = subset(test,select = -
c(pickup_longitude,pickup_latitude,dropoff_longitude,dropoff_latitude))

## now lets look at the summary of data
summary(train)

##Now remove the trip_distance having value less than 0, becouse these values
are not useful
train=subset(train, trip_distance>0)

df1=train
# train=df1

##########################  Visualization
#######################################
# Visualization between fare_amount and Hours.
ggplot(data = train, aes(x = hour, y = fare_amount, fill = hour))+
  geom_bar(stat = "identity")+
  labs(title = "Fare Amount Vs.Hour", x = "Hours", y = "Fare Amount",subtitle =
"Bi - Variate Analysis",
     caption = "(Observation : Rides taken during 6 pm to 11 pm gives highest
fare_amount.)")+
  theme(plot.title = element_text(hjust = 0.5, face = "bold"))+
  theme(axis.text.x = element_text( color="Red", size=10, angle=0))+
  theme(axis.text.y = element_text( color="blue", size=10, angle=0))
```

```
# Visualization between fare_amount and day.
ggplot(data = train, aes(x = day, y = fare_amount, fill = day))+
  geom_bar(stat = "identity")+
  labs(title = "Fare Amount Vs. Day", x = "Day", y = "Fare Amount",subtitle = "Bi -
Variate Analysis",
      caption = "(Observation : Rides taken during midweeks gives highest
fare_amount.)")+
  theme(plot.title = element_text(hjust = 0.5, face = "bold"))+
  theme(axis.text.x = element_text( color="Red", size=10, angle=0))+
  theme(axis.text.y = element_text( color="blue", size=10, angle=0))

# Visualization between fare_amount and weekday.
ggplot(data = train, aes(x = weekday,y = fare_amount, fill = weekday))+
  geom_bar(stat = "identity")+
  labs(title = "Fare Amount Vs. weekday", x = "weekday", y = "Fare
Amount",subtitle = "Bi - Variate Analysis",
      caption = "(Observation : Thursday to Saturday rides has the highest
fare_amount.)")+
  theme(plot.title = element_text(hjust = 0.5, face = "bold"))+
  theme(axis.text.x = element_text( color="Red", size=10, angle=0)) +
  theme(axis.text.y = element_text( color="Brown", size=10, angle=0))

# Visualization between fare_amount and years.
ggplot(data = train, aes(x = year, y = fare_amount, fill= year))+
  geom_bar(stat = "identity")+
  labs(title = "Fare Amount V/s years",  x = "Years", y = "Fare Amount", subtitle =
"Bi - Variate Analysis",
      caption = "(Observation : In year 2013 there were rides which got high
fare_amount)") +
  theme(plot.title = element_text(hjust = 0.5, face = "bold"))+
  theme(axis.text.x = element_text( color="Red", size=10, angle=0))+
  theme(axis.text.y = element_text( color="Brown", size=10, angle=0))

# Visualization between fare_amount and months.
#col <- rainbow(ncol(train))
ggplot(train, aes(x = month, y = fare_amount, fill= month))+
  geom_bar(stat = "identity")+
  labs(title = " Fare Amount V/s. Month", x = "Months", y = "Fare Amount",
subtitle = "Bi - Variate Analysis",
```

```
    caption = "(Observation: Month May collects the highest fare_amount)")+
  theme(plot.title = element_text(hjust = 0.5, face = "bold"))+
  theme(axis.text.x = element_text( color="Red", size=10, angle = 0))+
  theme(axis.text.y = element_text( color="Brown", size=10, angle = 0))


################                          Feature selection
###################
numeric_index = sapply(train,is.numeric) #selecting only numeric

numeric_data = train[,numeric_index]

cnames = colnames(numeric_data)
#Correlation analysis for numeric variables
corrgram(train[,numeric_index],upper.panel=panel.pie, main = "Correlation
Plot")

#ANOVA for categorical variables with target numeric variable
aov_results = aov(fare_amount ~ passenger_count + hour + weekday + month +
year+ day,data = train)
summary(aov_results)

# pickup_weekday has p value greater than 0.05
train = subset(train,select = -c(weekday, day))
test = subset(test,select = -c(weekday, day))

#########################     Feature Scaling
#######################
qqnorm(train$fare_amount)
hist(train$fare_amount)

hist(train$trip_distance)
qqnorm(train$trip_distance)

#data is normally distributed, no need of normalization


#################### Splitting train into train and validation subsets
###################
set.seed(1000)
```

```
tr.idx = createDataPartition(train$fare_amount,p=0.80,list = FALSE) # 80% in
trainin and 20% in Validation Datasets
train_data = train[tr.idx,]
test_data = train[-tr.idx,]

rmExcept(c("test","train","df",'df1','df2','df3','test_data','train_data','test_picku
p_datetime'))
###################Model Selection################
#Error metric used to select model is RMSE


#############          Linear regression           #################
lm_model = lm(fare_amount ~.,data=train_data)

summary(lm_model)  #R-squared:  0.7129
str(train_data)
plot(lm_model$fitted.values,rstandard(lm_model),main = "Residual plot",
    xlab = "Predicted values of fare_amount",
    ylab = "standardized residuals")


lm_predictions = predict(lm_model,test_data[,2:6])

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

regr.eval(test_data[,1],lm_predictions)
#mae      mse      rmse      mape
#1.4530745 4.2994675 2.0735157 0.1791255

#############                  Decision Tree          #####################

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

summary(Dt_model)
#Predict for new test cases
predictions_DT = predict(Dt_model, test_data[,2:6])

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

```
regr.eval(test_data[,1],predictions_DT)
#mae            mse           rmse          mape
#1.6439027 5.0444347 2.2459819 0.2129453


#############                    Random forest          #####################
rf_model = randomForest(fare_amount ~.,data=train_data)

summary(rf_model)

rf_predictions = predict(rf_model,test_data[,2:6])

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

regr.eval(test_data[,1],rf_predictions)
#mae          mse          rmse       mape
#1.668754 5.162626 2.272141 0.217559

### As we got best Accuracy with Linear Regression Model we will use this
Model to predict Fare
summary(test)

predicted_Fare=predict(lm_model,test)

test$Predicted_fare=predicted_Fare

write.csv(test, "predicted_test_R.csv", row.names = F)
```