Data Science Project Submitted to Edwisor

Cab Fare Prediction

Submitted by
Mr. Divyang Gor

Table of Contents

1. Introduction

1.1 Problem Statement

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

1.2 Data Set

1) Train_cab.zip
2) test.zip

1.3 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.

2. Methodology

In this chapter we will be looking for methodologies to be followed for analysing cab fare from the give data. We will be following sequence of methodologies.

2.1 Data Pre-processing

Before applying a predictive model on our data set we must apply several pre-processing techniques such as exploring the data, data cleaning as well as visualization by graph. All these steps are combined under one shed which is Exploratory Data Analysis that includes following steps:

- Data Exploration & Cleaning
- Missing Values Analysis
- Outlier Analysis
- Feature Selection
- Features Scaling
- Visualization

2.2 Modelling

After completing all steps of data Pre-processing we apply different models on out train data. 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 pre-processed 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 Regression
- Random Forest Regression
- XGB Regression

Hear Hyper parameter tunings to check the parameters on which our model runs best Cross Value Score is used.

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. Data Pre-Processing

Data Pre-processing is first step for all type of projects. If the data is messy, we try to improve it by sorting deleting extra rows and columns. This stage is called as Exploratory Data Analysis. This stage generally involves data cleaning, merging, sorting, looking for outlier analysis, looking for missing values in the data, imputing missing values if found by various methods such as mean, median, mode, KNN imputation, etc.

3.1 Data Types

In this step we convert the given data as per the appropriate data type. In this project following data type related operations done.

- pickup_datatime was given as object and which I have converted to datetime. [For both train & test data sets]
- Fare_amount was given as object and which I have converted to numeric.

3.2 Missing Value Analysis & Imputation

Before going for missing value analysis we need to remove unexpected values.
Remove data values which are not valid, following operations done.

- Removed fare_amount values < = 0.
- Removed passenger_count values <1 and >6.
- Removed pickup_latitude >180
- Round off passenger_count variable as passenger value can't be 1.3 or 1.5.

Check for missing value count for the given data.

For the variables containing missing values we will calculate Mean, Median, Mode and KNN Imputation by identifying best fit replace all NA's.

- For pickup_datetime variable variable only one missing value is there so we will just omit this.
- For passenger_count variable 56 missing values are there I have applied KNN Imputation.
- For fare_amount variable 24 missing values are there I have applied KNN Imputation.

3.3 Outliner Analysis

It may be possible that our data set contain outliners which are harmful for our predictions. For removing outliners Box plot has been used.

For the given data set fare_amount variable has 1396 outliners, I have converted these outliners to NA's and then Imputed by KNN Imputation.

3.4 Feature Engineering

Feature Engineering is used to drive new features from existing features.

For pickup_datetime variable:

I have used 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.

For 'Latitudes' & 'Longitudes' variables:

The variables 'pickup_latitude', 'pickup_longitude', 'dropoff_latitude', 'dropoff_longitude'.
I have calculated geodesic distance between pickup and dropoff.

Furthermore for geodesic distance variable I have checked for outliners. The outliners are replaced by NA's and imputed by KNN imputation method.

3.5 Feature Selection

In this step irrelevant features from the dataset to be removed. This is done by some statistical techniques. Depending upon data type statistical methods are used. In this data set for numerical variables I have used Correlation Analysis and for categorical variables I have used ANOVA test.

Even after deriving distance from 'pickup_latitude', 'pickup_longitude', 'dropoff_latitude', 'dropoff_longitude' we will drop these variables from our train data set.

3.6 Feature Scaling

Data Scaling methods are used when 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, the independent numerical variable ‘geodesic’ is not distributed normally so I have chosen normalization over standardization.
- I 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.

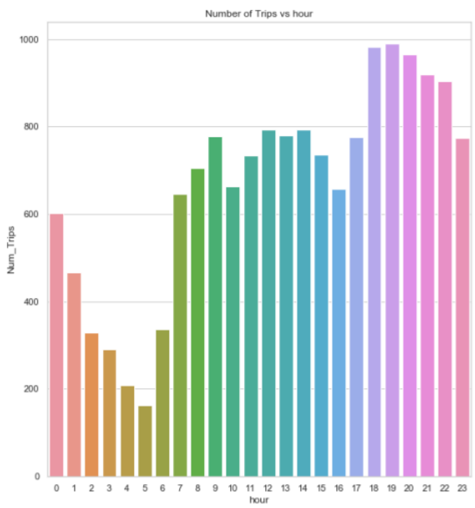
3.7 Visualization

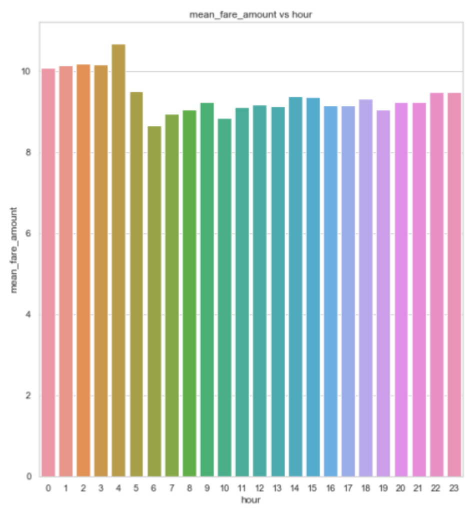
For visualization below mentioned graphs I have plotted.

- Count Vs Year & Average Fare_amount Vs Year
- Count Vs Month & Average Fare_amount Vs Month
- Count Vs Day of week & Average Fare_amount Vs Day of week
- Count Vs Hour & Average Fare_amount Vs Hour
- Count Vs Passenger Count & Average Fare_amount Vs Passenger Count

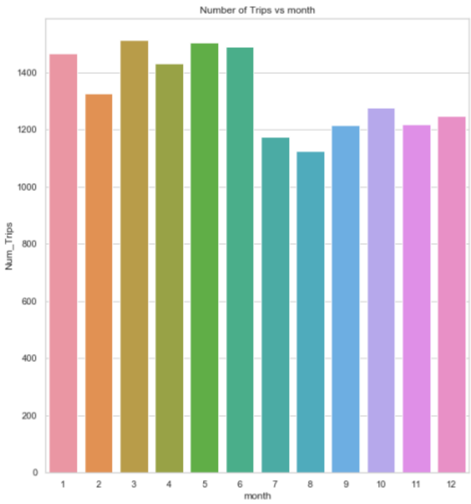
Form these visualizations we can conclude

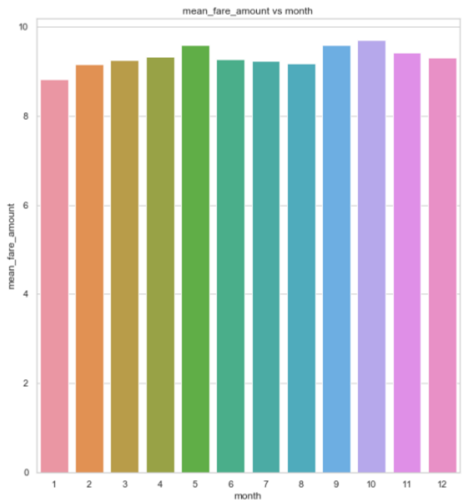
- Average Fare amount has been increasing over the years.
- Fares across months are fairly constant, though number of trips are lower from June to December.
- Fares across day of week are fairly constant.
- Average fare amount is higher at 4 and highest pickup during 18 to 20 hours.
- average fare amount is same for all passenger counts. Single passenger travels maximum.



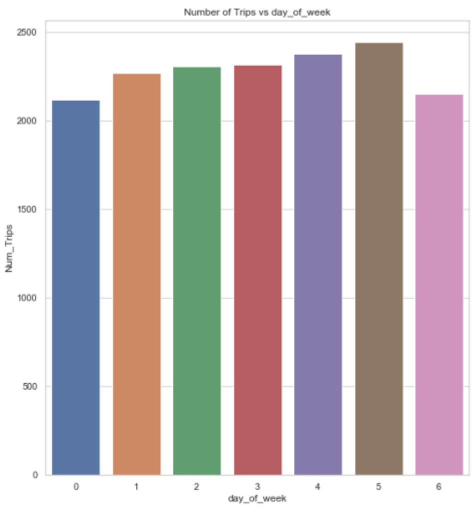


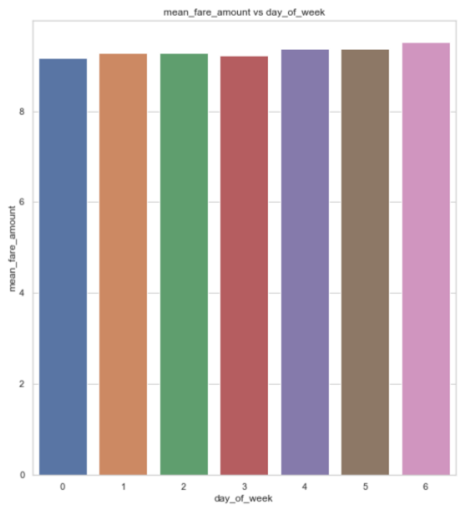
Count Vs Year & Average Fare_amount Vs Year



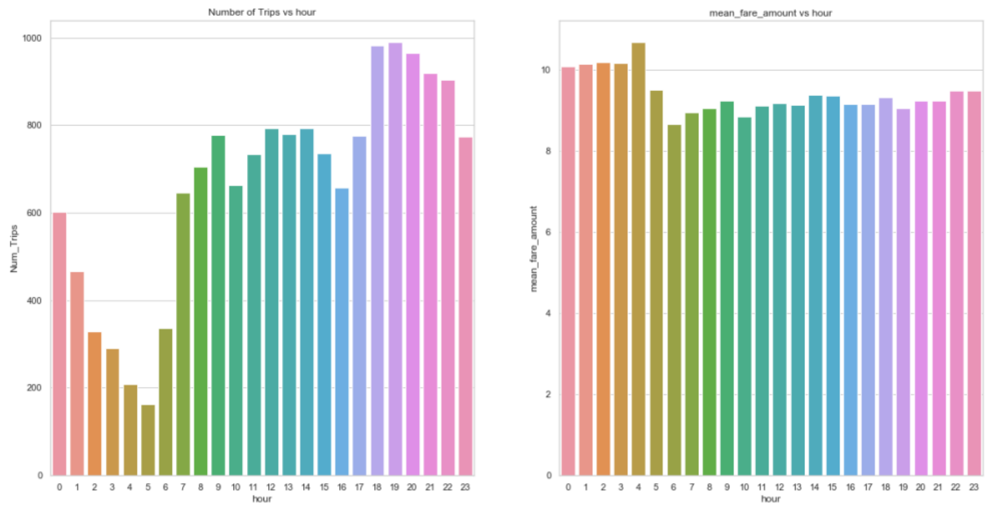


Count Vs Month & Average Fare_amount Vs Month

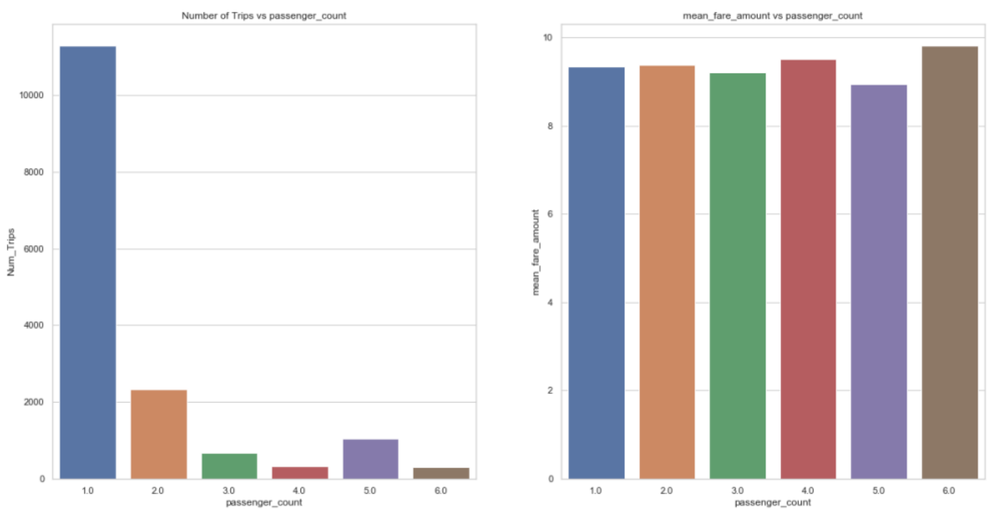




Count Vs Day of week & Average Fare_amount Vs Day of week



Count Vs Hour & Average Fare_amount Vs Hour



Count Vs Passenger Count & Average Fare_amount Vs Passenger Count

4. Modelling

The problem statement asks to predict the fare_amount. This is a Regression problem. So, I have built regression models on training data and predict it on test data. In this project I have built models using below mentioned Regression Algorithms:

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

To evaluate performance I have used:

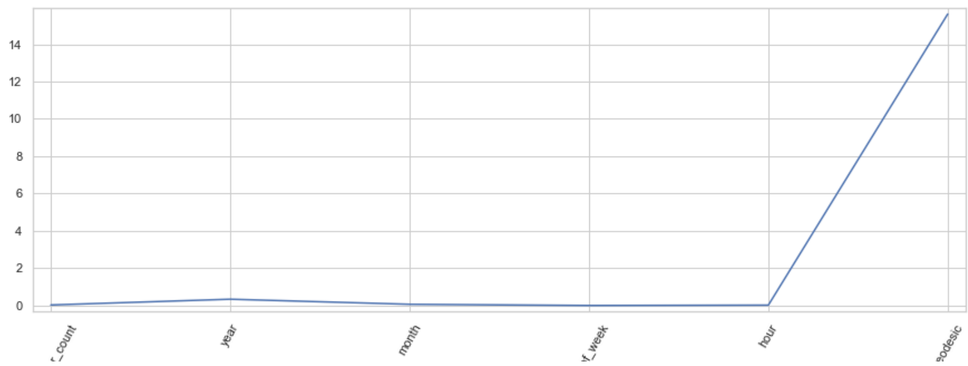
- The coefficient of determination R^2 of the prediction
- RMSE(Root Mean Square Error)
- RMSLE(Root Mean Squared Log Error)

Dividing in to Train & Test Data Sets

Before running any model, split the data into two parts which is train and test data. Here in this case I have taken 80% of the data as our train data.

4.1 Linear Regression Model

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



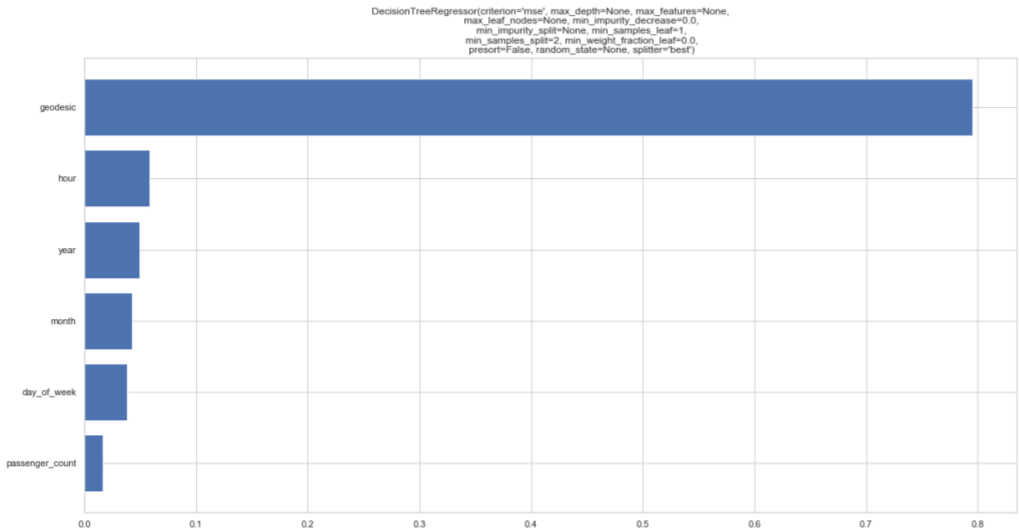
The coefficient of determination R^2 = 0.6840788779220737.
RMSE_test 2.3763761075113545
RMSE_train 2.387846886144272

CV Score for K = 5

LinearRegression_CV [0.69105989 0.67923199 0.70507178 0.70816572 0.66963886]
LinearRegression_CV_mean 0.6906336465836012

4.2 Decision Tree

A tree has many analogies in real life, and turns out that it has influenced a wide area of machine learning, covering both classification and regression. In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. As the name goes, it uses a tree-like model of decisions. Below are the observations:



The coefficient of determination R^2 = 0.4229640932944523
RMSE_test 3.183908928775855
RMSE_train 0.013964479679398043

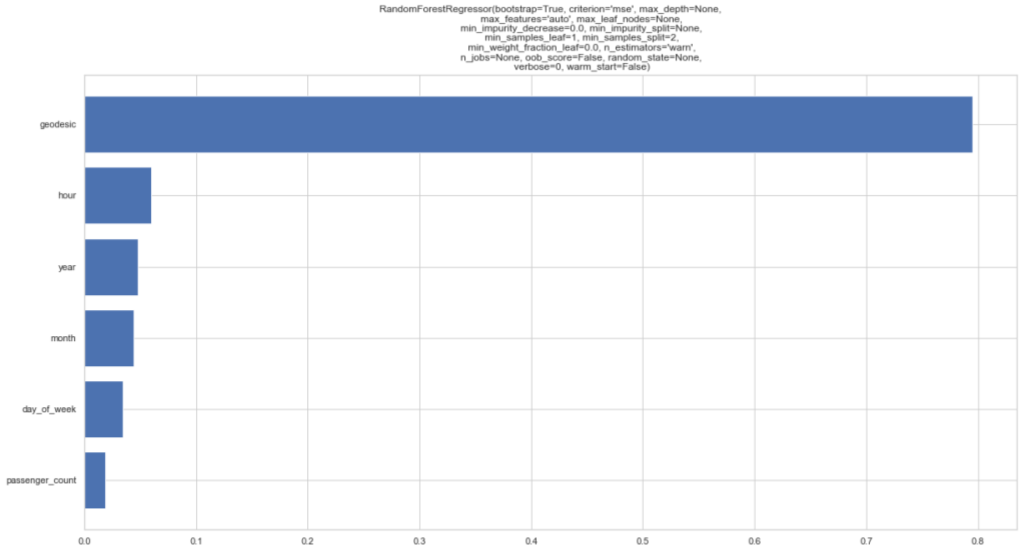
CV Score for K = 5

DecisionTreeRegressor_CV [0.4224513 0.45637032 0.42876695 0.50563121 0.44273142]
DecisionTreeRegressor_CV_mean 0.4436377236916845

4.3 Random Forest

Random forests or random decision forests are an ensemble learning method for classification, regression and other task, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

To say it in simple words: Random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction.
Below are the observations:



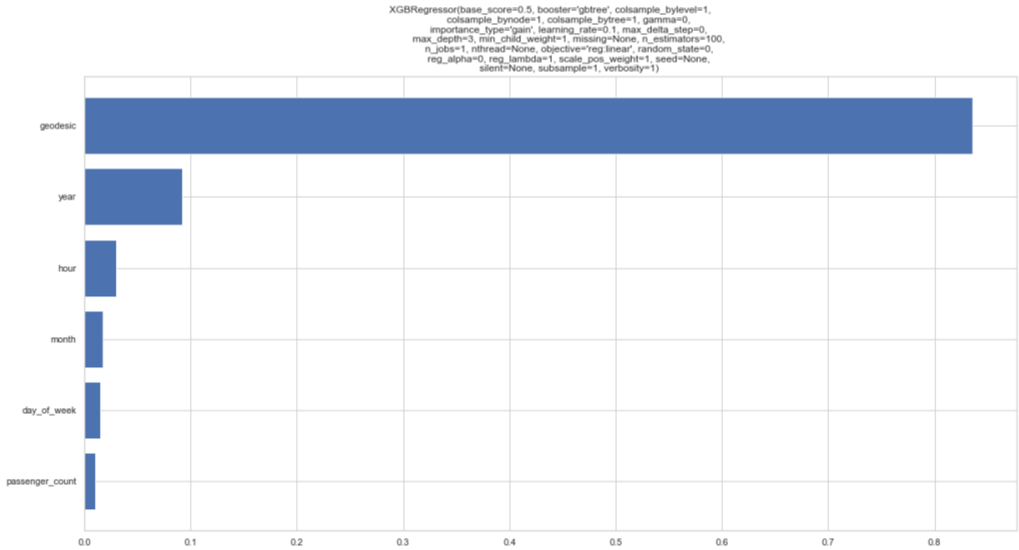
The coefficient of determination R^2 = 0.683747456978568
RMSE_test 2.3847321328759965
RMSE_train 1.0082452916644926

CV Score for K = 5

RandomForestRegressor_CV [0.68820902 0.68492772 0.68923233 0.71079251 0.68440965]
RandomForestRegressor_CV_mean 0.6933771878088129

4.4 XGB Regression

XGBoost (Extreme Gradient Boosting) belongs to a family of boosting algorithms and uses the gradient boosting (GBM) framework at its core. It is an optimized distributed gradient boosting library. Below are the observations:



The coefficient of determination R^2 = 0.7319002257126387
RMSE_test 2.1891426883468053
RMSE_train 2.1234972646489703

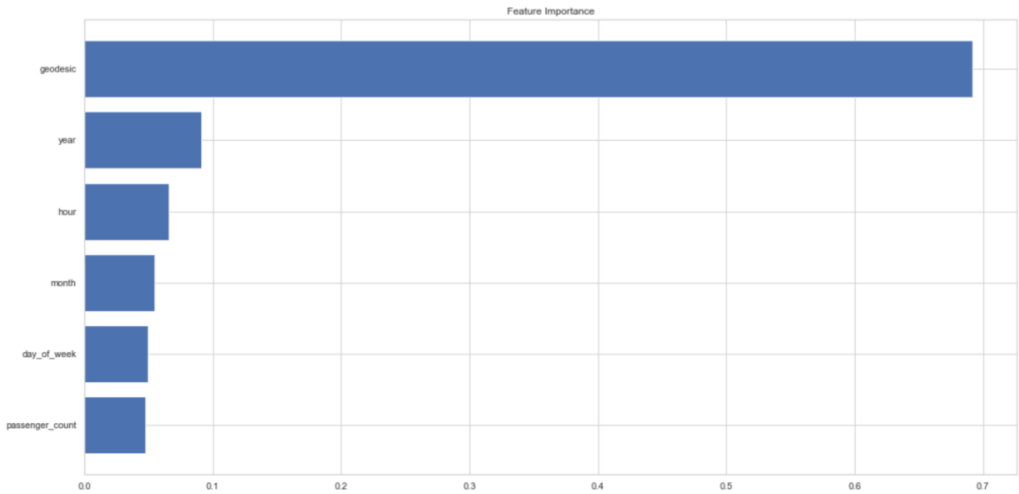
CV Score for K = 5
XGBRegressor_CV [0.74321283 0.73612606 0.74348742 0.75401336 0.73133161]
XGBRegressor_CV_mean 0.7416342551819691

5. Improving Accuracy

For improving accuracy I have used XGBoost as a ensemble technique.

Xgboost hyperparameters tuned parameters:Tuned Xgboost Parameters: {'subsample': 0.1, 'reg_alpha': 0.08685113737513521, 'n_estimators': 500, 'max_depth': 3, 'learning_rate': 0.05, 'colsample_bytree': 0.7000000000000001, 'colsample_bynode': 0.7000000000000001, 'colsample_bylevel': 0.9000000000000001}

After applying XGBoost ensemble technique:



The coefficient of determination R^2 = 0.7303281336291445
RMSE_test 2.1955516894225844
RMSE_train 2.113705445258577

CV Score for K = 5
XGBRegressor_CV [0.74052413 0.73634252 0.74060759 0.75100974 0.73106192]
XGBRegressor_CV_mean 0.7398257636872658

6. Conclusion

6.1 Model Evaluation

The main concept of looking at what is called residuals or difference between our predictions f(x[I,]) and actual outcomes y[i].

In general, most data scientists use two methods to evaluate the performance of the model:

- RMSE (Root Mean Square Error): is a frequently used measure of the difference between values predicted by a model and the values actually observed from the environment that is being modelled.
- R Squared(R^2): is a statistical measure of how close the data are to the fitted regression line. It is also known as the coefficient of determination, or the coefficient of multiple determination for multiple regression. In other words, we can say it explains as to how much of the variance of the target variable is explained.
- Average CV Score I have included for model evaluation.

Below table shows the model results:

Model Name	**R^2**	**RMSE Train**	**RMSE Test**	**CV Score Mean**
Linear Regression	0.684078878	2.387846886	2.376376108	0.690633647
Decision Tree	0.422964093	0.01396448	3.183908929	0.443637724
Random Forest	0.683747457	1.008245292	2.384732133	0.693377188
XGBoost	0.731900226	2.123497265	2.189142688	0.741634255

Below table shows the model result after improving accuracy:

Model Name	**R^2**	**RMSE Train**	**RMSE Test**	**CV Score Mean**
XGBoost	0.730328134	2.113705445	2.195551689	0.739825764

There is small Improvement in RMSE train.

6.2 Model Selection

On the basis RMSE and R Squared results a good model should have least RMSE and max R Squared value. So, from above tables we can see XBGoost is the best method for doing predictions in this project.

Finally, I used this method to predict the target variable for the test data file shared in the problem statement. Results that I found are attached with my submissions.

Note: Usual methods to run code in Python & R to be used.

7. Python Code

```
#!pip install pyforest
from pyforest import *
from geopy.distance import geodesic
from scipy.stats import chi2_contingency
from statsmodels.formula.api import ols
import statsmodels.api as sm
from fancyimpute import KNN
from patsy import dmatrices
from statsmodels.stats.outliers_influence import variance_inflation_factor
import scipy.stats as stats
from sklearn.linear_model import LinearRegression,Ridge,Lasso
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error
from sklearn import metrics
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from xgboost import XGBRegressor
import xgboost as xgb
from sklearn.externals import joblib
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score


# In[2]:


#Working Directory
os.getcwd()


# In[3]:


#Read data as data frame from CSV
train_cab = pd.read_csv("https://s3-ap-southeast-1.amazonaws.com/edwisor-india-bucket/projects/data/DataN0104/train_cab.zip")
test_cab = pd.read_csv("https://s3-ap-southeast-1.amazonaws.com/edwisor-india-bucket/projects/data/DataN0104/test.zip")


# In[4]:


#View top 5 rows of data
```

```
train_cab.head()


# In[5]:


test_cab.head()


# In[6]:


#Data Information
train_cab.info()


# In[7]:


test_cab.info()


# ## Data Type

# In[8]:


train_cab.dtypes


# In[9]:


test_cab.dtypes


# In[10]:


#Change data type from object to datetime /float64
train_cab["pickup_datetime"] = pd.to_datetime(train_cab.pickup_datetime, errors = 
'coerce')
train_cab["fare_amount"] = pd.to_numeric(train_cab.fare_amount, errors = 'coerce')
test_cab["pickup_datetime"] = pd.to_datetime(test_cab.pickup_datetime, errors = 'coerce')


# In[11]:
```

```
train_cab.dtypes


# In[12]:


test_cab.dtypes


# In[13]:


train_cab.describe()


# In[14]:


#Fare amount can't be negative/zero
train_cab = train_cab.drop(train_cab[train_cab['fare_amount']<=0].index, axis = 0)
# Passenger_count must be positive integer 1, 2, 3, 4, 5 or 6
train_cab = train_cab.drop(train_cab[train_cab['passenger_count']>6].index, axis =0)
train_cab = train_cab.drop(train_cab[train_cab['passenger_count']<1].index, axis =0)
train_cab['passenger_count'] = train_cab['passenger_count'].round(0)
#latitude can never be more than 180
train_cab = train_cab.drop(train_cab[train_cab['pickup_latitude']>180].index, axis =0)


# In[15]:


train_cab.describe()


# # Missing Value Analysis

# In[16]:


train_cab.isnull().sum()


# In[17]:
```

```
test_cab.isnull().sum()


# In[18]:


#Missing Value Analysis
missing_value = pd.DataFrame(train_cab.isnull().sum())
missing_value = missing_value.reset_index()
missing_value = missing_value.rename(columns =
{'index':'variables',0:'missing_percentage'})
missing_value['missing_percentage']=(missing_value['missing_percentage']/len(train_cab))*
100
missing_value = missing_value.sort_values('missing_percentage', ascending= False)
missing_value


# In[19]:


#imputation_passenger_count
#Original Value = 1
#Mean = 1.3142578044948425
#Median = 1.0
#Mode = 1.0
#KNN = 1.38
train_cab['passenger_count'].loc[100] = np.nan
train_cab['passenger_count'].loc[100]
#We will impute passenger_count by KNN Imputation


# In[20]:


print('Mean: ',train_cab['passenger_count'].mean())
print('Median:',train_cab['passenger_count'].median())
print('Mode:', train_cab['passenger_count'].mode())


# In[21]:


#Null Values Count
pd.DataFrame(train_cab.isnull().sum())


# In[22]:
```

```
train_cab = train_cab.dropna(subset = ['pickup_datetime'])
pd.DataFrame(train_cab.isnull().sum())


# In[23]:


#Imputation fare_amount
#actual = 10.0
#Mean = 15.409399247348453
#Median = 8.5
#Mode = 6.5
#KNN = 10.068677622209403
print(train_cab['fare_amount'].loc[100] )
train_cab['fare_amount'].loc[100]=np.nan
train_cab['fare_amount'].loc[100]


# In[24]:


print('Mean: ',train_cab['fare_amount'].mean())
print('Median: ', train_cab['fare_amount'].median())
print('Mode', train_cab['fare_amount'].mode())


# In[25]:


train_cab_pickup_datetime = train_cab['pickup_datetime']


# In[26]:


columns=['fare_amount', 'pickup_longitude', 'pickup_latitude','dropoff_longitude',
'dropoff_latitude', 'passenger_count']
train_cab = pd.DataFrame(KNN(k = 3).fit_transform(train_cab.drop('pickup_datetime',axis =
1)),columns = columns, index = train_cab.index)


# In[27]:


train_cab['fare_amount'].loc[100]
```

```
# In[28]:


train_cab['passenger_count'].loc[100]


# In[29]:


train_cab['passenger_count'] = train_cab['passenger_count'].round(0)


# In[30]:


train_cab['pickup_datetime'] = train_cab_pickup_datetime
pd.DataFrame(train_cab.isnull().sum())


# # Outliner Analysis

# In[31]:


sns.set(style="whitegrid")
get_ipython().run_line_magic('matplotlib', 'inline')
plt.figure(figsize = (20,5))
sns.boxplot(data=train_cab['fare_amount'],orient='h')


# In[32]:


plt.figure(figsize = (20,5))
sns.boxplot(x=train_cab['fare_amount'],y=train_cab['passenger_count'],data=train_cab,orie
nt='h')
#sns.boxplot(data=temp,x=train['fare_amount'],y=train['passenger_count'],orient = 'h',
palette="colorblind",width=0.9)


# In[33]:


plt.figure(figsize=(20,5))
plt.xlim(0,100)
```

```
sns.boxplot(x=train_cab['fare_amount'],data=train_cab,orient='h')
plt.title('Boxplot of fare_amount')
# plt.savefig('bp of fare_amount.png')
plt.show()


# In[34]:


#Outliner Analysis
q75, q25 = np.percentile(train_cab['fare_amount'], [75 ,25])

#Calculate IQR
iqr = q75 - q25

#Calculate inner and outer fence
minimum = q25 - (iqr*1.5)
maximum = q75 + (iqr*1.5)


#Replace with NA
train_cab.fare_amount[train_cab.fare_amount < minimum] = np.nan
train_cab.fare_amount[train_cab.fare_amount > maximum] = np.nan
pd.DataFrame(train_cab.isnull().sum())


# In[35]:


train_cab_pickup_datetime = train_cab['pickup_datetime']
#temp_cab = train_cab
#actual 17.3
train_cab['fare_amount'].loc[150]=np.nan


# In[36]:


#KNN for NA values
columns=['fare_amount', 'pickup_longitude', 'pickup_latitude','dropoff_longitude',
'dropoff_latitude', 'passenger_count']
train_cab = pd.DataFrame(KNN(k = 50).fit_transform(train_cab.drop('pickup_datetime',axis
= 1)),columns = columns, index = train_cab.index);


# In[37]:
```

```
train_cab['fare_amount'].loc[150]


# In[38]:


train_cab['pickup_datetime']=train_cab_pickup_datetime


# In[39]:


train_cab.head()


# In[40]:


pd.DataFrame(train_cab.isnull().sum())


# # Feature Engineering

# In[41]:


train_cab['year'] = train_cab['pickup_datetime'].apply(lambda row: row.year)
train_cab["month"] = train_cab["pickup_datetime"].apply(lambda row: row.month)
train_cab["day_of_week"] = train_cab["pickup_datetime"].apply(lambda row:
row.dayofweek)
train_cab["hour"] = train_cab["pickup_datetime"].apply(lambda row: row.hour)


# In[42]:


test_cab['year'] = test_cab['pickup_datetime'].apply(lambda row: row.year)
test_cab["month"] = test_cab["pickup_datetime"].apply(lambda row: row.month)
test_cab["day_of_week"] = test_cab["pickup_datetime"].apply(lambda row:
row.dayofweek)
test_cab["hour"] = test_cab["pickup_datetime"].apply(lambda row: row.hour)


# In[43]:
```

```
test_cab.head()


# In[44]:


train_cab.head()


# In[45]:


train_cab['geodesic']=train_cab.apply(lambda x:
geodesic((x['pickup_latitude'],x['pickup_longitude']), (x['dropoff_latitude'],
x['dropoff_longitude'])).km, axis=1)
test_cab['geodesic']=test_cab.apply(lambda x:
geodesic((x['pickup_latitude'],x['pickup_longitude']), (x['dropoff_latitude'],
x['dropoff_longitude'])).km, axis=1)


# In[46]:


train_cab.head()


# In[47]:


train_cab.head()


# In[48]:


sns.set(style="whitegrid")
get_ipython().run_line_magic('matplotlib', 'inline')
plt.figure(figsize = (20,5))
sns.boxplot(data=train_cab['geodesic'],orient='h')


# In[49]:


train_cab.describe()
```

```
# In[50]:


#Outliner Analysis
q75, q25 = np.percentile(train_cab['geodesic'], [75 ,25])

#Calculate IQR
iqr = q75 - q25

#Calculate inner and outer fence
minimum = q25 - (iqr*1.5)
maximum = q75 + (iqr*1.5)


#Replace with NA
train_cab.geodesic[train_cab.geodesic < minimum] = np.nan
train_cab.geodesic[train_cab.geodesic > maximum] = np.nan
pd.DataFrame(train_cab.isnull().sum())


# In[51]:


#outliner Analysis
#Actual Value =1.1734217770759794
#Mean = 2.3812342631431775
#Median =1.9334082879643537
#Mode = 0.0
#KNN =
train_cab['geodesic'].loc[200] = np.nan


# In[52]:


print("Mean", train_cab['geodesic'].mean())
print("Median", train_cab['geodesic'].median())
print("Mode", train_cab['geodesic'].mode()[0])


# In[53]:


columns=['fare_amount', 'pickup_longitude', 'pickup_latitude','dropoff_longitude',
'dropoff_latitude', 'passenger_count','year','month','day_of_week','hours','geodesic']
train_cab = pd.DataFrame(KNN(k = 5).fit_transform(train_cab.drop('pickup_datetime',axis =
1)),columns = columns, index = train_cab.index);
```

In[54]:

```
train_cab.head()
```

Data Visulization

In[55]:

```
def groupandplot(data,groupby_key,value,aggregate='mean'):
    plt.figure(figsize=(20,10))

agg_data=data.groupby([groupby_key])[value].agg(aggregate).reset_index().rename(colum
ns={value:aggregate+'_'+value})
    plt.subplot(1,2,1)

count_data=data.groupby([groupby_key])['fare_amount'].count().reset_index().rename(col
umns={'fare_amount':'Num_Trips'})
    sns.barplot(x=groupby_key,y='Num_Trips',data=count_data).set_title("Number of Trips vs
"+groupby_key)

    plt.subplot(1,2,2)

sns.barplot(x=groupby_key,y=aggregate+'_'+value,data=agg_data).set_title(aggregate+'_'+v
alue+" vs "+groupby_key)
```

In[56]:

```
print(groupandplot(train_cab,'hours','fare_amount'))
print(groupandplot(train_cab,'month','fare_amount'))
print(groupandplot(train_cab,'year','fare_amount'))
print(groupandplot(train_cab,'day_of_week','fare_amount'))
print(groupandplot(train_cab,'passenger_count','fare_amount'))
```

Avg Fare amount has been increasing over the years.

Fares across months are fairly constant, though number of trips are lower from june to decemeber

Average fare amount is higher at 4 and highest pickup during 18 to 20 hours.

average fare amount is same for all passerger counts.
Single passenger travels maximum.

Feature Selection

In[57]:

```
#Select continuous variable for correlaiton analysis
conti_vari_train_cab = train_cab[['fare_amount','geodesic']]
corr_train_cab = conti_vari_train_cab.corr()
corr_train_cab
```

In[58]:

```
sns.heatmap(corr_train_cab, annot=True)
```

In[59]:

```
sns.jointplot(x='fare_amount',y='geodesic',data=train_cab,kind = 'reg')
plt.show()
```

In[60]:

```
train_cab.describe()
```

In[61]:

```
train_cab = train_cab.drop(['pickup_longitude','pickup_latitude'],axis=1)
```

In[62]:

```
train_cab = train_cab.drop(['dropoff_longitude','dropoff_latitude'],axis=1)
```

In[63]:

```
test_cab =
test_cab.drop(['pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitude'],a
xis = 1)


# In[64]:


model_name = ols('fare_amount ~ passenger_count + hours + day_of_week + month +
year', data=train_cab).fit()


# In[65]:


model_name.summary()


# In[66]:


aov_table = sm.stats.anova_lm(model_name,type = 1)
aov_table


# # Feature Scaling

# In[67]:


#Feature Scaling Check with or without normalization of standarscalar
sns.distplot(train_cab['geodesic'],bins=50)


# In[68]:


plt.figure()
stats.probplot(train_cab['geodesic'], dist='norm', fit=True,plot=plt)


# In[69]:


#Normalization
```

```
train_cab['geodesic'] = (train_cab['geodesic'] -
min(train_cab['geodesic']))/(max(train_cab['geodesic']) - min(train_cab['geodesic']))
#test.csv['geodesic'] = (test['geodesic'] - min(test['geodesic']))/(max(test['geodesic']) -
min(test['geodesic']))


# In[70]:


sns.distplot(train_cab['geodesic'],bins=50)


# In[71]:


stats.probplot(train_cab['geodesic'], dist='norm', fit=True,plot=plt)


#
# # Splitting train and Validation Dataset

# In[72]:


#Splitting Data into train and validation subsets
X = train_cab.drop('fare_amount',axis=1).values
y = train_cab['fare_amount'].values
X_train_cab, X_test_cab, y_train_cab, y_test_cab = train_test_split(X, y, test_size = 0.20,
random_state=42)
print(train_cab.shape, X_train_cab.shape,
X_test_cab.shape,y_train_cab.shape,y_test_cab.shape)


# ## Model Development

# # Linear Regression Model

# In[73]:


def get_score(model, X_train_cab, X_test_cab, y_train_cab, y_test_cab):
   model.fit(X_train_cab, y_train_cab)
   return model.score(X_test_cab,y_test_cab)


# In[74]:
```

```
print('Linear Regression',get_score(LinearRegression(), X_train_cab, X_test_cab,
y_train_cab, y_test_cab))
print('DecisionTreeRegressor',get_score(DecisionTreeRegressor(), X_train_cab, X_test_cab,
y_train_cab, y_test_cab))
print('RandomForestRegressor',get_score(RandomForestRegressor(), X_train_cab,
X_test_cab, y_train_cab, y_test_cab))
print('XGBRegressor',get_score(XGBRegressor(), X_train_cab, X_test_cab, y_train_cab,
y_test_cab))


# In[75]:


def RMSE(model, X_train, X_test, y_train, y_test):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    y_pred_train = model.predict(X_train)
    print(model)
    print('RMSE_test', np.sqrt(mean_squared_error(y_test, y_pred)))
    print('RMSE_train', np.sqrt(mean_squared_error(y_train, y_pred_train)))


# In[76]:


print('Linear Regression',RMSE(LinearRegression(), X_train_cab, X_test_cab, y_train_cab,
y_test_cab))
print('DecisionTreeRegressor',RMSE(DecisionTreeRegressor(), X_train_cab, X_test_cab,
y_train_cab, y_test_cab))
print('RandomForestRegressor',RMSE(RandomForestRegressor(), X_train_cab, X_test_cab,
y_train_cab, y_test_cab))
print('XGBRegressor',RMSE(XGBRegressor(), X_train_cab, X_test_cab, y_train_cab,
y_test_cab))


# # KFold

# In[77]:


folds = StratifiedKFold(n_splits=10)


# In[78]:
```

```
print('LinearRegression_CV', cross_val_score(LinearRegression(), X, y,cv=5))
print('LinearRegression_CV_mean', cross_val_score(LinearRegression(), X, y,cv=5).mean())


# In[79]:


print('DecisionTreeRegressor_CV', cross_val_score(DecisionTreeRegressor(), X, y,cv=5))
print('DecisionTreeRegressor_CV_mean', cross_val_score(DecisionTreeRegressor(), X,
y,cv=5).mean())


# In[80]:


print('RandomForestRegressor_CV', cross_val_score(RandomForestRegressor(), X, y,cv=5))
print('RandomForestRegressor_CV_mean', cross_val_score(RandomForestRegressor(), X,
y,cv=5).mean())


# In[81]:


print('XGBRegressor_CV', cross_val_score(XGBRegressor(), X, y,cv=5))
print('XGBRegressor_CV_mean', cross_val_score(XGBRegressor(), X, y,cv=5).mean())


# In[86]:


test_cab_1 = test_cab.drop(['pickup_datetime'],axis =1)


# In[87]:


def plot_regression(model,X_train, y_train):
    reg_coef_m = model.fit(X_train,y_train).coef_
    print(reg_coef_m)
    # Plot the coefficients
    plt.figure(figsize=(15,5))
    plt.plot(range(len(test_cab_1.columns)), reg_coef_m)
    plt.xticks(range(len(test_cab_1.columns)), test_cab_1.columns.values, rotation=60)
    plt.margins(0.02)
    plt.show()
```

```
# In[88]:


print('Linear Regression Coefficient Plot',
plot_regression(LinearRegression(),X_train_cab,y_train_cab))


# In[89]:


def plot_importance(model, X_train_cab, y_train_cab):
   # Creating plot
   fig = plt.figure(figsize=(20,10))
   plt.title(model)
   tree_features = model.fit(X_train_cab,y_train_cab).feature_importances_
   print(tree_features)
   indices = np.argsort(tree_features)[::1]
   names = [test_cab_1.columns[i] for i in indices]
   # Add horizontal bars
   plt.barh(range(pd.DataFrame(X_train_cab).shape[1]),tree_features[indices],align =
'center')
   plt.yticks(range(pd.DataFrame(X_train_cab).shape[1]), names)
   plt.show()


# In[91]:


print('DecisionTreeRegressor',plot_importance(DecisionTreeRegressor(),X_train_cab,y_train
_cab))
print('RandomForestRegressor',plot_importance(RandomForestRegressor(),X_train_cab,y_t
rain_cab))
print('XGBRegressor',plot_importance(XGBRegressor(),X_train_cab,y_train_cab))


# In[92]:


#Improve Accuraccy Using XGBOOST
data_dmatrix = xgb.DMatrix(data=X,label=y)
dtrain = xgb.DMatrix(X_train_cab, label=y_train_cab)
dtest = xgb.DMatrix(X_test_cab)


# In[93]:
```

```
dtrain,dtest,data_dmatrix


# In[94]:


params = {"objective":"reg:linear",'colsample_bytree': 0.3,'learning_rate': 0.1,
          'max_depth': 5, 'alpha': 10}

cv_results = xgb.cv(dtrain=data_dmatrix, params=params, nfold=3,
             num_boost_round=100,early_stopping_rounds=10,metrics="rmse",
as_pandas=True, seed=123)
cv_results.head()


# In[95]:


# the final boosting round metric
print((cv_results["test-rmse-mean"]).tail(1))


# In[96]:


a=pd.read_csv('test.csv')


# In[97]:


test_pickup_datetime=a['pickup_datetime']


# In[98]:


# Instantiate a xgb regressor: xgb
Xgb = XGBRegressor(subsample= 0.1, reg_alpha= 0.08685113737513521, n_estimators=
500, max_depth= 3, learning_rate=0.05, colsample_bytree= 0.7000000000000001,
colsample_bynode=0.7000000000000001, colsample_bylevel=0.9000000000000001)

# Fit the regressor to the data
Xgb.fit(X,y)

# Compute and print the coefficients
xgb_features = Xgb.feature_importances_
```

```
print(xgb_features)

# Sort feature importances in descending order
indices = np.argsort(xgb_features)[::1]

# Rearrange feature names so they match the sorted feature importances
names = [test_cab_1.columns[i] for i in indices]

# Creating plot
fig = plt.figure(figsize=(20,10))
plt.title("Feature Importance")

# Add horizontal bars
plt.barh(range(pd.DataFrame(X_train_cab).shape[1]),xgb_features[indices],align = 'center')
plt.yticks(range(pd.DataFrame(X_train_cab).shape[1]), names)
plt.savefig(' xgb1 feature importance')
plt.show()
RMSE(Xgb,X_train_cab,X_test_cab,y_train_cab,y_test_cab)
print('Linear Regression',get_score(Xgb, X_train_cab, X_test_cab, y_train_cab, y_test_cab))
print('XGBRegressor_CV', cross_val_score(Xgb, X, y,cv=5))
print('XGBRegressor_CV_mean', cross_val_score(Xgb, X, y,cv=5).mean())

# Predictions
pred = Xgb.predict(test_cab_1.values)
pred_results_wrt_date =
pd.DataFrame({"pickup_datetime":test_pickup_datetime,"fare_amount" : pred})
pred_results_wrt_date.to_csv("predictions_xgboost.csv",index=False)


# In[99]:


pred_results_wrt_date
```

8. R Code

```
rm(list = ls())
setwd("/Users/divyanggor/Documents/Study/Online_Course/Edwisor/Project/")

# #loading Libraries
x = c("ggplot2", "corrgram", "DMwR", "usdm", "caret", "randomForest", "e1071",
    "DataCombine", "doSNOW", "inTrees", "rpart.plot",
"rpart",'MASS','xgboost','stats','gdistance', 'Imap', 'car')
#load Packages
lapply(x, require, character.only = TRUE)
rm(x)

train_cab= read.csv("train_cab.csv")
test_cab= read.csv("test.csv")
test_pickup_datetime = test_cab["pickup_datetime"]
str(train_cab)
str(test_cab)
summary(train_cab)
summary(test_cab)
head(train_cab,5)
head(test_cab,5)

# Changing the data types of variables
train_cab$fare_amount = as.numeric(as.character(train_cab$fare_amount))

###################################### Drop Invalid Data Entries ###############
# 1.Fare amount can't be negative or zero
nrow(train_cab[which(train_cab$fare_amount <=0 ),])
train_cab = train_cab[-which(train_cab$fare_amount <=0 ),]

#2. Passenger_count must be positive integer 1, 2, 3, 4, 5 or 6
nrow(train_cab[which(train_cab$passenger_count <1 ),])
nrow(train_cab[which(train_cab$passenger_count >6 ),])
train_cab = train_cab[-which(train_cab$passenger_count < 1 ),]
train_cab = train_cab[-which(train_cab$passenger_count > 6),]
train_cab$passenger_count=round(train_cab$passenger_count)

#3. latitude can never be more than 180
nrow(train_cab[which(train_cab$pickup_latitude > 180), ])
train_cab = train_cab[-which(train_cab$pickup_latitude > 180), ]

###################### Missing Value Analysis ##########################
apply(train_cab,2,function(x){sum(is.na(x))})
missing_val = data.frame(apply(train_cab,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_cab)) * 100
missing_val = missing_val[order(-missing_val$Missing_percentage),]
row.names(missing_val) = NULL
missing_val = missing_val[,c(2,1)]
missing_val

train_cab[,'passenger_count'] = factor(train_cab[,'passenger_count'], labels=(1:6))
unique(train_cab$passenger_count)
apply(test_cab,2,function(x){sum(is.na(x))})

#1. Passenger_count Variable
unique(train_cab$passenger_count)
unique(test_cab$passenger_count)
#Mean Method
mean(train_cab$passenger_count, na.rm = T)
#Mode Methond
getmode = function(x) {
  uniq = unique(x)
  uniq[which.max(tabulate(match(x, uniq)))]
}
getmode(train_cab$passenger_count)
#For KNN
train_cab$passenger_count[200]
train_cab$passenger_count[200]=NA

#Mean = 1.649633
#Mode = 1
#KNN = 1

#2. Fare amount Variable
# Mean Method
mean(train_cab$fare_amount, na.rm = T)
#Median Method
median(train_cab$fare_amount, na.rm = T)
#KNN Imputation
train_cab$fare_amount[500]
train_cab$fare_amount[500]=NA
train_cab = knnImputation(train_cab, k = 3)
train_cab$fare_amount[500]
train_cab$passenger_count[200]
#Actual Value = 6
#Mean = 15.04909
#Median = 8.5
#KNN = 7

########################   Outlinear Analysis ######################
# Boxplot for fare_amount variable
```

```
pl1 = ggplot(train_cab,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)

# Replace all outliers with NA and impute
values = train_cab[,"fare_amount"] %in% boxplot.stats(train_cab[,"fare_amount"])$out
train_cab[which(values),"fare_amount"] = NA
#check for NA's
sum(is.na(train_cab$fare_amount))
#Imputing with KNN
train_cab = knnImputation(train_cab,k=3)
#check for NA's
sum(is.na(train_cab$fare_amount))


##################          Feature Engineering          ##########################
# new features derived from pickup_datetime are year,month,day_of_week,hour
train_cab$pickup_date = as.Date(as.character(train_cab$pickup_datetime))
sum(is.na(train_cab))
train_cab = na.omit(train_cab)
train_cab$day_of_week = as.factor(format(train_cab$pickup_date,"%u"))# Monday = 1
train_cab$month = as.factor(format(train_cab$pickup_date,"%m"))
train_cab$year = as.factor(format(train_cab$pickup_date,"%Y"))
pickup_time = strptime(train_cab$pickup_datetime,"%Y-%m-%d %H:%M:%S")
train_cab$hour = as.factor(format(pickup_time,"%H"))

test_cab$pickup_date = as.Date(as.character(test_cab$pickup_datetime))
test_cab$day_of_week = as.factor(format(test_cab$pickup_date,"%u"))# Monday = 1
test_cab$month = as.factor(format(test_cab$pickup_date,"%m"))
test_cab$year = as.factor(format(test_cab$pickup_date,"%Y"))
pickup_time = strptime(test_cab$pickup_datetime,"%Y-%m-%d %H:%M:%S")
test_cab$hour = as.factor(format(pickup_time,"%H"))

train_cab = subset(train_cab,select = -c(pickup_datetime,pickup_date))
test_cab = subset(test_cab,select = -c(pickup_datetime,pickup_date))
# Calculate Distance
train_cab$geodesic = gdist(train_cab$pickup_longitude, train_cab$pickup_latitude, train_cab$dropoff_longitude, train_cab$dropoff_latitude, units = "km", a = 6378137.0, b = 6356752.3142, verbose = FALSE)
test_cab$geodesic = gdist(test_cab$pickup_longitude, test_cab$pickup_latitude, test_cab$dropoff_longitude, test_cab$dropoff_latitude, units = "km", a = 6378137.0, b = 6356752.3142, verbose = FALSE)
# Removing Outliners from geodesic
# Boxplot for fare_amount variable
pl2 = ggplot(train_cab,aes(x = factor(passenger_count),y = geodesic))
pl2 + geom_boxplot(outlier.colour="red", fill = "grey" ,outlier.shape=18,outlier.size=1, notch=FALSE)+ylim(0,100)
```

```
# Replace all outliers with NA and impute
values_1 = train_cab[,"geodesic"] %in% boxplot.stats(train_cab[,"geodesic"])$out
train_cab[which(values_1),"geodesic"] = NA
#the NA's
sum(is.na(train_cab$geodesic))
#Imputing with KNN
train_cab = knnImputation(train_cab,k=3)
# lets check the missing values
sum(is.na(train_cab$geodesic))
################  Feature selection              ###############
numeric = sapply(train_cab,is.numeric) #selecting numeric variables
numeric_data = train_cab[,numeric]
cnames = colnames(numeric_data)
#Correlation analysis for numeric variables
cor(numeric_data)
corrgram(train_cab[,numeric],upper.panel=panel.pie, main = "Correlation Plot")

#As both numberic variables Facre_amount and geodesic are highly correlated with each other.

#Removing Categorical variables
train_cab = subset(train_cab,select = -c(pickup_longitude,pickup_latitude,dropoff_latitude, dropoff_longitude))
test_cab = subset(test_cab,select = -c(pickup_longitude,pickup_latitude,dropoff_latitude, dropoff_longitude))

#Anova Test
aov_results = aov(fare_amount ~ passenger_count + hour + day_of_week + month + year,data = train_cab)

summary(aov_results)
# pickup_weekdat has p value greater than 0.05
train_cab = subset(train_cab,select=-day_of_week)
#remove from test set
test_cab = subset(test_cab,select=-day_of_week)


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

par(mfrow=c(1,2))
qqPlot(train_cab$geodesic)                     # qqPlot, it has a x values derived from gaussian distribution, if data is distributed normally then the sorted data points should lie very close to the solid reference line
truehist(train_cab$geodesic)                  # truehist() scales the counts to give an estimate of the probability density.
```

```
lines(density(train_cab$geodesic))  # Right skewed       # lines() and density() functions to overlay a density plot on histogram

train_cab[,'geodesic'] = (train_cab[,'geodesic'] - min(train_cab[,'geodesic']))/
  (max(train_cab[,'geodesic'] - min(train_cab[,'geodesic'])))
#################### Splitting train into train and validation subsets ###################
set.seed(1000)
tr = createDataPartition(train_cab$fare_amount,p=0.80,list = FALSE) # 80% in trainin and 25% in Validation Datasets
train_data = train_cab[tr,]
test_data = train_cab[-tr,]
###################Model Selection###############
#Error metric used to select model is RMSE
#############          Linear regression          #################
lm_model = lm(fare_amount ~.,data=train_data)

summary(lm_model)
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)
#############                    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)
#############                    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)
############              Improving Accuracy by using Ensemble technique ---- XGBOOST ############################
train_data_matrix = as.matrix(sapply(train_data[-1],as.numeric))
test_data_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(xgboost_model)
```

```
xgb_predictions = predict(xgboost_model,test_data_data_matrix)
qplot(x = test_data[,1], y = xgb_predictions, data = test_data, color = I("blue"), geom = "point")
regr.eval(test_data[,1],xgb_predictions)
############# Finalizing and Saving Model for later use ####################
# In this step we will train our model on whole training Dataset and save that model for later use
train_data_matrix2 = as.matrix(sapply(train_cab[-1],as.numeric))
test_data_matrix2 = as.matrix(sapply(test_cab,as.numeric))

xgboost_model2 = xgboost(data = train_data_matrix2,label = train_cab$fare_amount,nrounds = 15,verbose = FALSE)

# Saving the trained model
saveRDS(xgboost_model2, "./final_Xgboost_model_using_R.rds")

# loading the saved model
super_model <- readRDS("./final_Xgboost_model_using_R.rds")
print(super_model)

# Lets now predict on test dataset
xgb = predict(super_model,test_data_matrix2)

xgb_pred = data.frame(test_pickup_datetime,"predictions" = xgb)

write.csv(xgb_pred,"xgb_predictions_R.csv",row.names = FALSE)
```