CAB FARE PREDICTION

BY

Deepak Kulkarni

30/10/2019

TABLE OF CONTENTS

CHAPTER 1

Introduction

1.1 Problem Statement

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

1.2 Dataset

The dataset provided has a combination of categorical and numerical data type. The target variable is fare_amount which is a categorical variable.
The dataset provided has all together 07variables (06 independent and 01 dependent). The training data has 16067 observations, testing data has 9914 observations.

Table1.1: Sample data to predict bike rent

fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
4.5	2009-06-15 17:26:21 UTC	-73.844311	40.721319	-73.84161	40.712278	1
16.9	2010-01-05 16:52:16 UTC	-74.016048	40.711303	-73.979268	40.782004	1
5.7	2011-08-18 00:35:00 UTC	-73.982738	40.76127	-73.991242	40.750562	2
7.7	2012-04-21 04:30:42 UTC	-73.98713	40.733143	-73.991567	40.758092	1
5.3	2010-03-09 07:51:00 UTC	-73.968095	40.768008	-73.956655	40.783762	1
12.1	2011-01-06 09:50:45 UTC	-74.000964	40.73163	-73.972892	40.758233	1

As it can be seen from table1.1 the following predictors are used to predict the fare_amount as shown in table 1.2.Table 1.2 gives the details of all the variables with their data types.

Table 1.2: Predictor variables

Sl no	Predictor	Type
1	fare_amount	Numerical
2	pickup_datetime	categorical
3	pickup_longitude	Numerical
4	pickup_latitude	Numerical
5	dropoff_longitude	Numerical
6	dropoff_latitude	Numerical
7	passenger_count	categorical

CHAPTER 2

Methodology

2.1 DATA PREPROCESSING

The first step of the data analysis is the data pre processing. It involves finding out whether there are any missing values present in the given dataset. If the missing data in the dataset is more than 30% then ignore the predictor which has those missing values. On the other hand if the given data has less than 30% of missing values then impute the missing values by mean, median or any other method.

With reference to the given dataset there are three variables which have missing values in the data as shown in Table 2.1 .

Table 2.1: Missing value analysis

Sl no	Predictor	Missing_percentage
1	fare_amount	0.155598
2	pickup_datetime	0.006224
3	pickup_longitude	0.0
4	pickup_latitude	0.0
5	dropoff_longitude	0.0
6	dropoff_latitude	0.0
7	passenger_count	0.342317

2.2 OUTLIER ANALYSIS

In data analysis there is possibility that there are outliers present in the given dataset. An **outlier** is an observation that lies an abnormal distance from other values in a random sample from a population. Examination of the **data** for unusual observations that are far removed from the mass of **data**. These points are often referred to as **outliers**.

Outliers affect the mean value of the **data** but have little **effect** on the median or mode of a given set of **data**.

- **fare_amount:** It should be always a value which is greater than 0. Hence remove all the values of fare_amount which are less than and equal to 0. There are no values less than or equal to 0.

- **pickup_longitude and dropoff_longitude:** The longitude values range from -180 degrees to +180 degrees. All the values beyond these two extremities are removed. There are no values less than or equal to 0.
- **pickup_latitude and dropoff_latitude:** The latitude values range from -90 degrees to +90 degrees. All the values beyond these two extremities are removed. pickup_latitude. pickup_latitude has one value > 90 degrees. Hence remove this observation.
- **passenger count:** The number of passengers depend upon the type of the cab. In SUV it is 6. Its value is minimum 1 and maximum 6. Passenger count cannot be a value which is equal to and less than 0. There are 21 values of passenger_count >6 and 58 values of passenger_count < 1. Hence remove these from dataset.

There are different ways to visualize the presence of outliers. One of the method used is boxplot as shown in fig 2.1. The red colour dots in the figure2.1 indicate the presence of outliers.

Outliers are imputed either by using mean, mode or median based on the requirement.

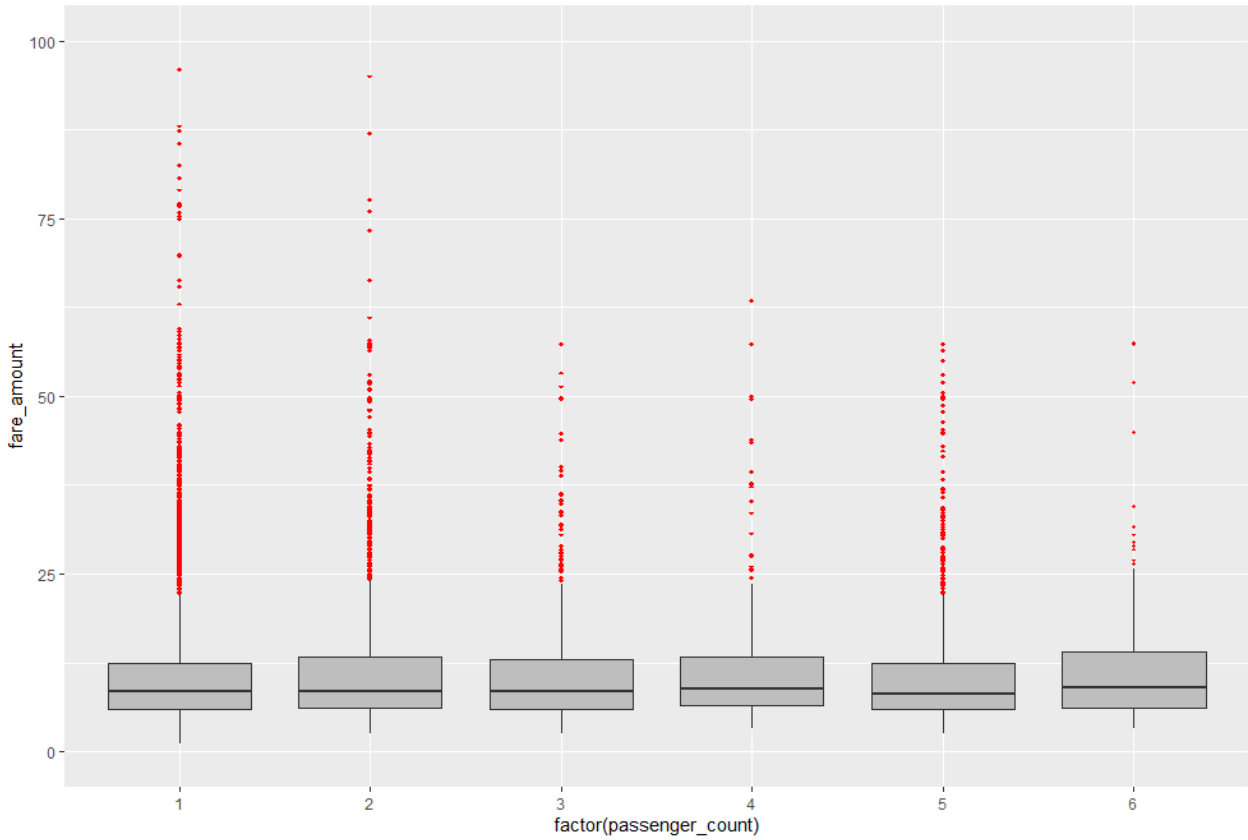

fig 2.1: Outlier analysis

From above Boxplots we see that ‘fare_amount’have outliers .‘fare_amount’ has 1359 outliers.

Impute these outliers with KNN and K value is 3.

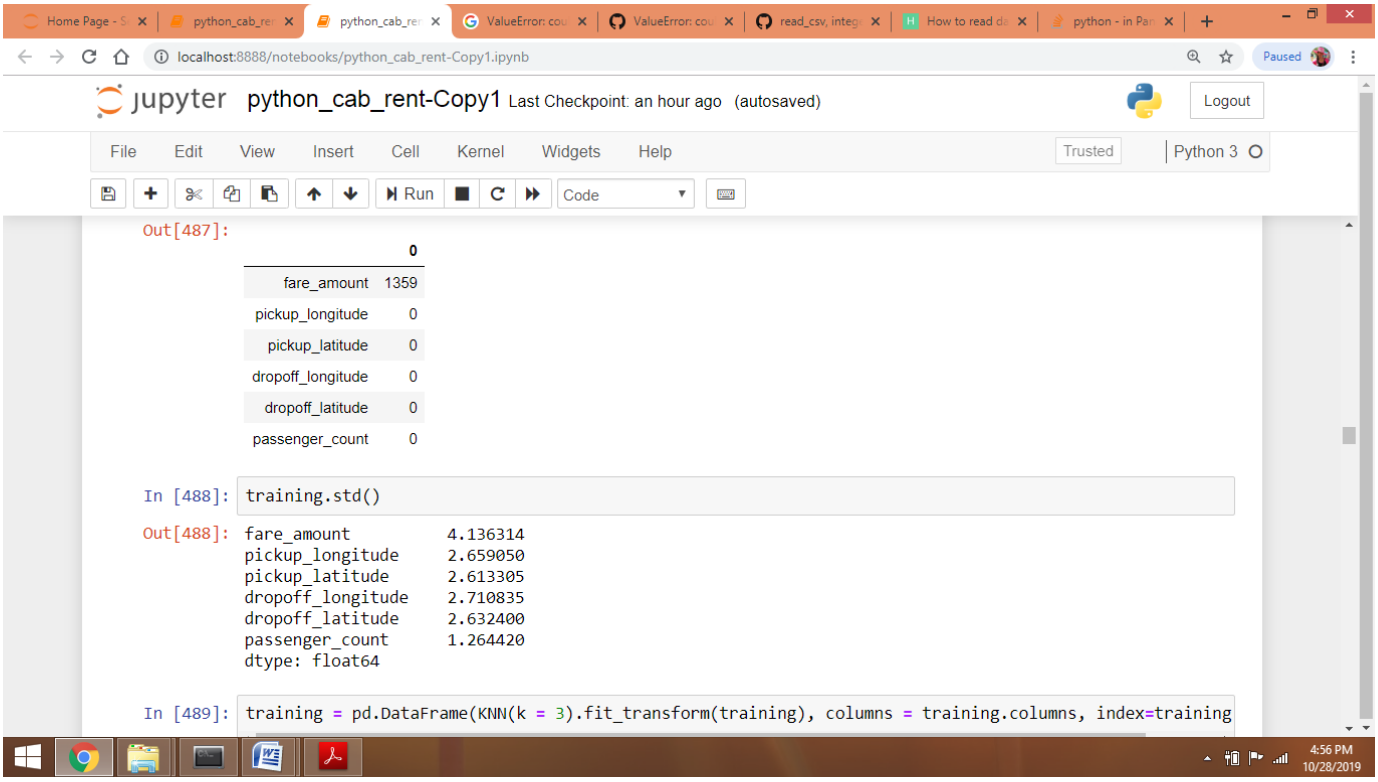

fig 2.2: Outlier imputation

2.3 Feature Engineering

Feature engineering is the process of deriving new features from existing features.

timestamp variable is used to create new variables(features).

The newly derived features are : year, month, day_of_week, hour.

- ‘year’ : contains only years from pickup_datetime.
- ‘month’: contains only months from pickup_datetime.
- ‘day_of_week’: contains only week from pickup_datetime.
- ‘hour’: contain only hours from pickup_datetime.

These variables are categorized as follows

- Week: has categories—weekday, weekend.
- Session: has categories—morning, afternoon, evening, night_PM, night_AM.
- Seasons: has categories—spring, summer, fall, winter.

To calculate distance, great_circle and geodesic packages are used from geopy library

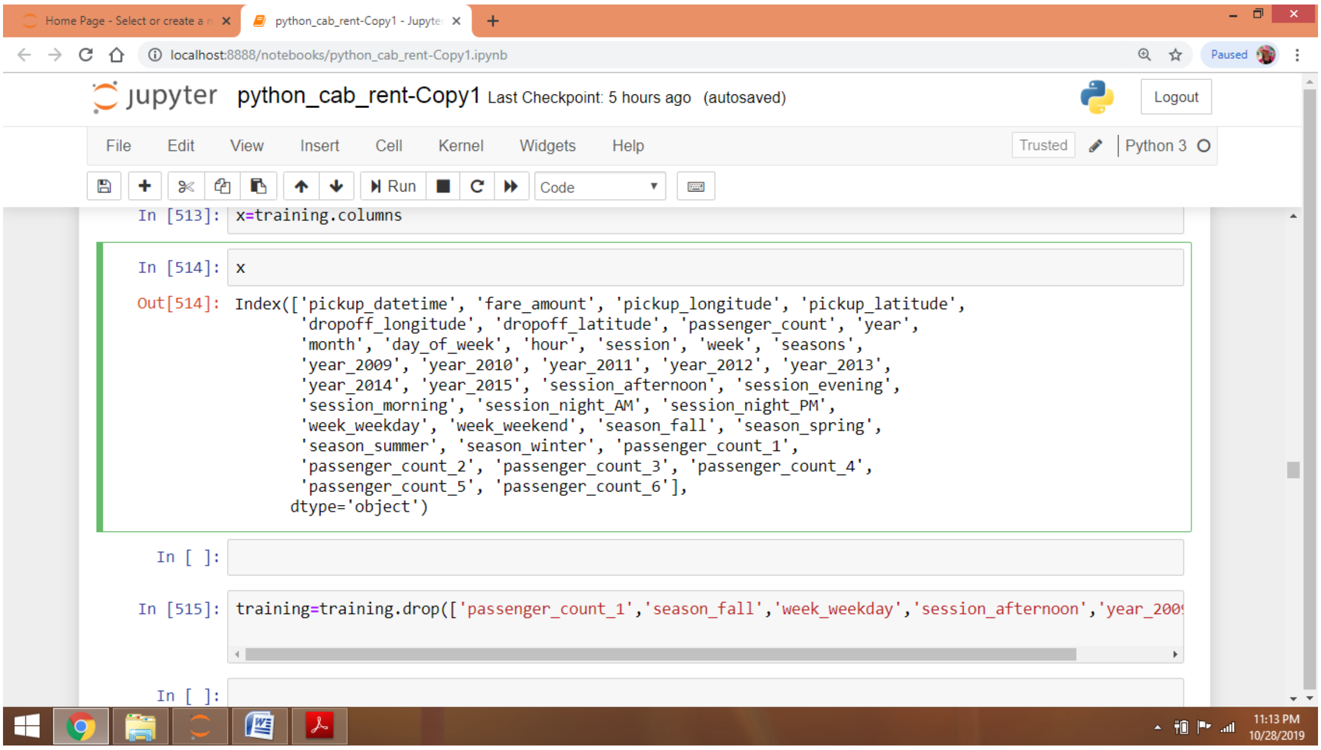

Fig 2.3: columns of training data

The columns of testing data are:

Index(['passenger_count_2', 'passenger_count_3', 'passenger_count_4',
'passenger_count_5', 'passenger_count_6', 'season_spring',
'season_summer', 'season_winter', 'week_weekend', 'session_evening',
'session_morning', 'session_night_AM', 'session_night_PM', 'year_2010',
'year_2011', 'year_2012', 'year_2013', 'year_2014', 'year_2015',
'geodesic'],
dtype='object')

2.4 **Feature Selection**

In the given data it is very important to select the desired predictors. One of the method used is correlation analysis. If the predictors are correlated then such predictors should be ignored. A sample correlation plot is shown in figure 2.4.

In the figure reddish shade of colour colour indicates that the predictors are correlated. i.e. fare_amount' and 'geodesic' are very highly correlated with each other.

Hence those predictors could be ignored while building the model. geodesic' is independent variable hence retain 'geodesic' which helps to explain variation in fare_amount.

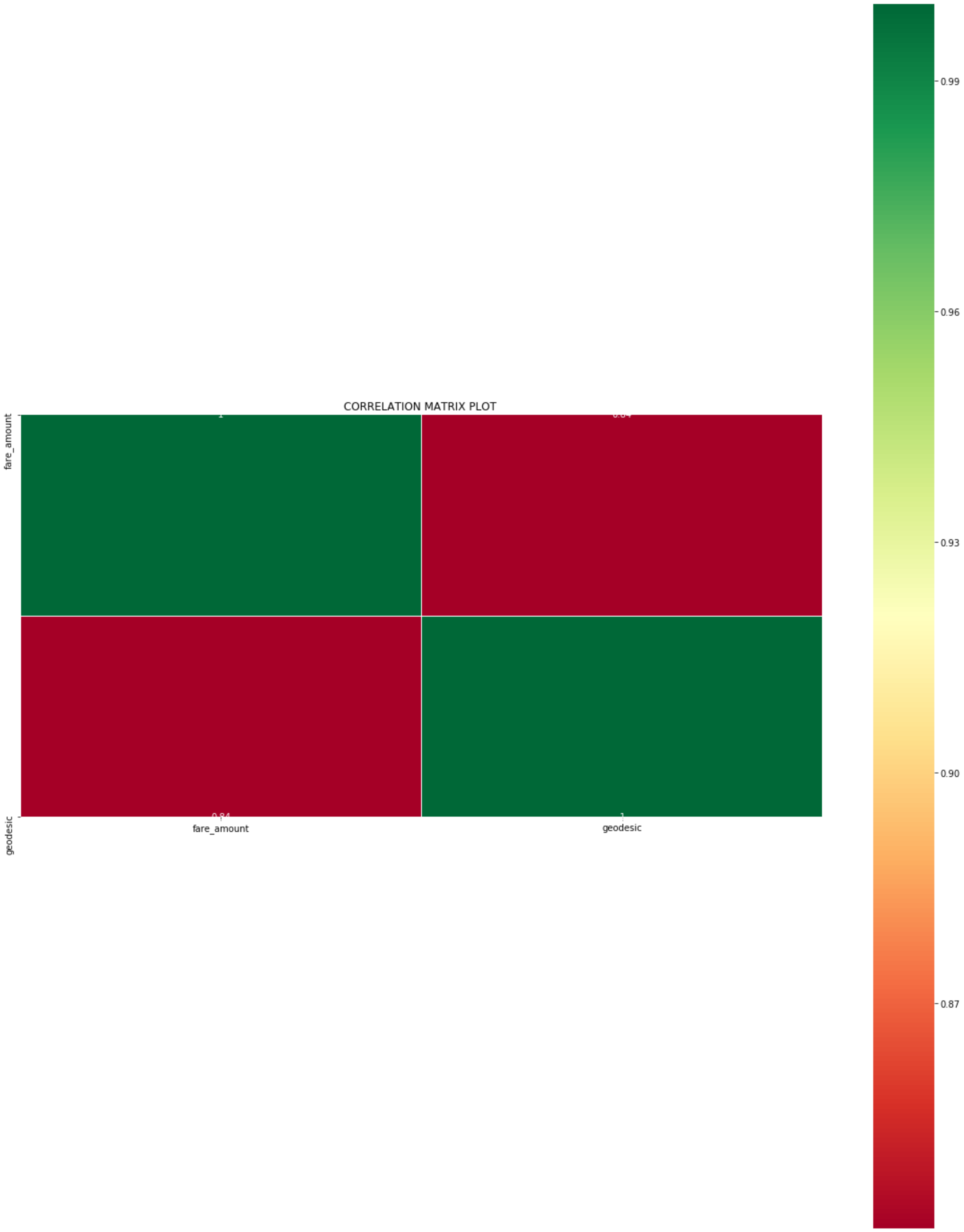

fig2.4: Correlation plot

2.5 Chi-Square test of independence

If p-value < 0.05: reject the null hypothesis i.e. the two variables are dependent.

If p-value > 0.05: we fail to reject null hypothesis i.e.the two variables are independent.

In the proposed project , p-value<0.05 then eliminate the variable, on the other hand p-value>0.05 retain the variable.

2.6 ANOVA (Analysis of Variance)

ANOVA compares means between each group in a categorical variable.

In ANOVA if p-value is less than 0.05 then reject the null hypothesis, if p-value is more than 0.05 then we fail to reject null hypothesis as shown in fig 2.5.

2.7 Multicollinearity

Multicollinearity stands for dependency of independent variable on each other. Presence of multicollinearity increases the standard errors of the coefficients . It also makes some variables statistically insignificant when they should be significant.

Table 2.2 : VIF relation on multicollinearity

VIF = 1	Not correlated to any of the variables
VIF between 1-5	Moderately correlated
VIF > 5	Highly correlated.

In case of multiple variables, if VIF > 5,then eliminate the variable with the highest VIF. Refer table 2.3 for the same.

2.8 Feature Scaling

- Normalization : This operation is performed only on continuous variables. It scales the data in the range of 0 to 1. Also it scales the data to a very small interval
- Standardization: subtract the mean from individual point and then dividing by its standard deviation.

Z +ve : when above mean

Z –ve: The raw score is below the mean

When the data is distributed normally then standardization is preferred.

Geodesic variable is not distributed normally hence normalization is performed as shown in histogram in figure 2.5

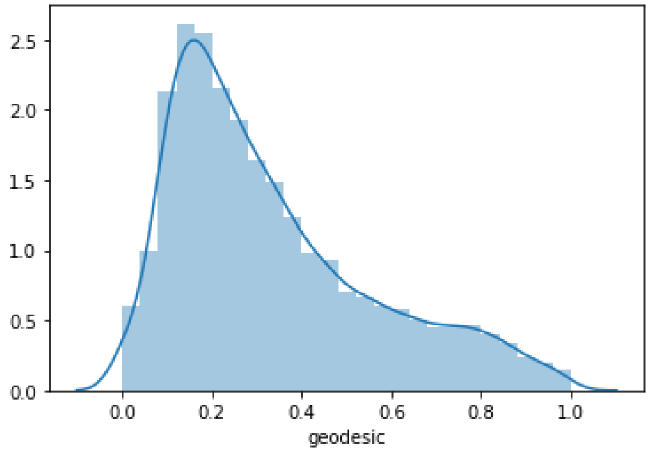

fig2.5: Normalization plot

Table 2.3 : VIF values

	VIF	features
0	13.539661	Intercept
1	1.040608	passenger_count_2[T.1]
2	1.019935	passenger_count_3[T.1]
3	1.011762	passenger_count_4[T.1]
4	1.024929	passenger_count_5[T.1]
5	1.017192	passenger_count_6[T.1]
6	1.642362	season_spring[T.1]
7	1.552476	season_summer[T.1]
8	1.587200	season_winter[T.1]
9	1.050420	week_weekend[T.1]
10	1.352982	session_night_AM[T.1]
11	1.414590	session_night_PM[T.1]
12	1.523718	session_evening[T.1]
13	1.557980	session_morning[T.1]
14	1.691354	year_2010[T.1]
15	1.687563	year_2011[T.1]
16	1.710999	year_2012[T.1]
17	1.709225	year_2013[T.1]
18	1.664884	year_2014[T.1]
19	1.406898	year_2015[T.1]
20	1.002286	geodesic

	df	sum_sq	mean_sq	F	PR(>F)
C(passenger_count_2)	1.0	15.702342	15.702342	0.810496	3.679876e-01
C(passenger_count_3)	1.0	20.242243	20.242243	1.044828	3.067170e-01
C(passenger_count_4)	1.0	65.984016	65.984016	3.405847	6.498466e-02
C(passenger_count_5)	1.0	20.620463	20.620463	1.064351	3.022409e-01
C(passenger_count_6)	1.0	146.904591	146.904591	7.582662	5.900038e-03
C(season_spring)	1.0	29.553582	29.553582	1.525445	2.168160e-01
C(season_summer)	1.0	26.015488	26.015488	1.342822	2.465547e-01
C(season_winter)	1.0	482.031698	482.031698	24.880662	6.164527e-07
C(week_weekend)	1.0	132.338763	132.338763	6.830829	8.968347e-03
C(session_night_AM)	1.0	2124.324542	2124.324542	109.649639	1.420799e-25
C(session_night_PM)	1.0	184.103941	184.103941	9.502753	2.055208e-03
C(session_evening)	1.0	0.787004	0.787004	0.040622	8.402716e-01
C(session_morning)	1.0	49.251318	49.251318	2.542168	1.108627e-01
C(year_2010)	1.0	1511.939011	1511.939011	78.040602	1.115052e-18
C(year_2011)	1.0	1336.438438	1336.438438	68.981923	1.074341e-16
C(year_2012)	1.0	433.374307	433.374307	22.369151	2.269228e-06
C(year_2013)	1.0	338.710175	338.710175	17.482945	2.914751e-05
C(year_2014)	1.0	1495.981444	1495.981444	77.216933	1.688519e-18
C(year_2015)	1.0	2607.742603	2607.742603	134.601860	5.404235e-31
Residual	15640.0	303005.428128	19.373749	NaN	NaN

Fig 2.5: ANOVA analysis

CHAPTER 3

3.1 Model Selection

Model selection plays a crucial step in data analysis. With respect to the problem, target variable is continuous. Hence it is a prediction problem. Therefore we need to make use of modelling techniques like decision trees, linear regression. In this project decision tree and linear regression has been used to model the system.

25% is in testing dataset and 75% is in training data.

The ERROR METRIC is calculated in both the cases and compared the results.

Since this is a regression problem, we are going to build regression models on training data and predict it on test data.

For linear regression:

Testing Data Details

r square 0.7396859522949595
MAPE:18.996541434699864
MSE: 5.339895187894899
RMSE: 2.3108213232300976
RMSLE: 0.21610620151348917

Training Data Details

r square 0.7340215100925873
MAPE:18.749715161100394
MSE: 5.292756664431027
RMSE: 2.3005991968248245
RMSLE: 0.21672325016344096

For decision tree

Training Data Details

r square 0.7457182342174193
MAPE:18.568099750553237
MSE: 5.0580020429887655
RMSE: 2.2490002318783264
RMSLE: 0.20912873163544968

Testing Data Details

r square 0.739695709486023
MAPE:19.156471529266362
MSE: 5.337649766014794
RMSE: 2.3103354228368644
RMSLE: 0.21276123055020005

APPENDIX A

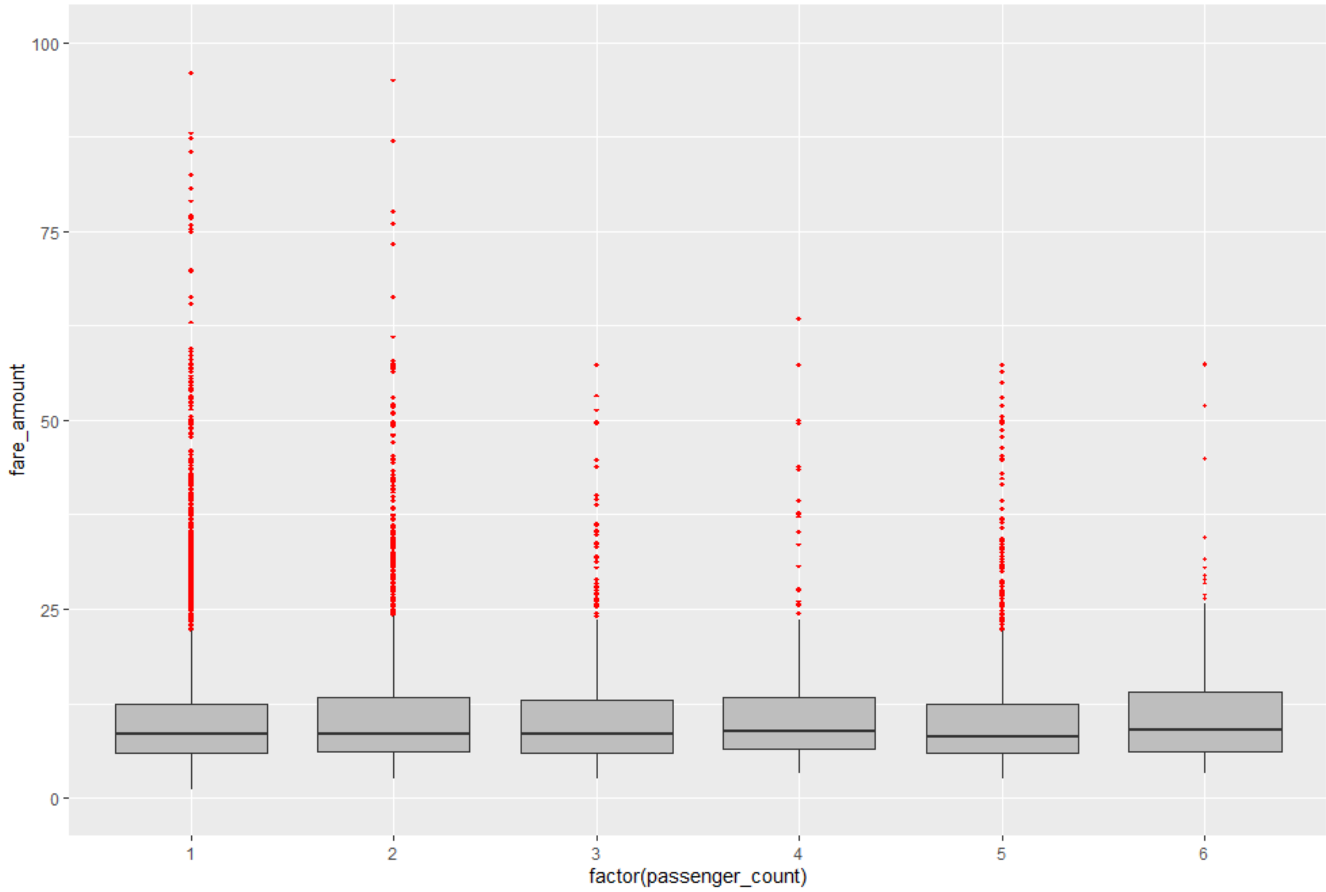

Fig A.1: Outlier Analysis

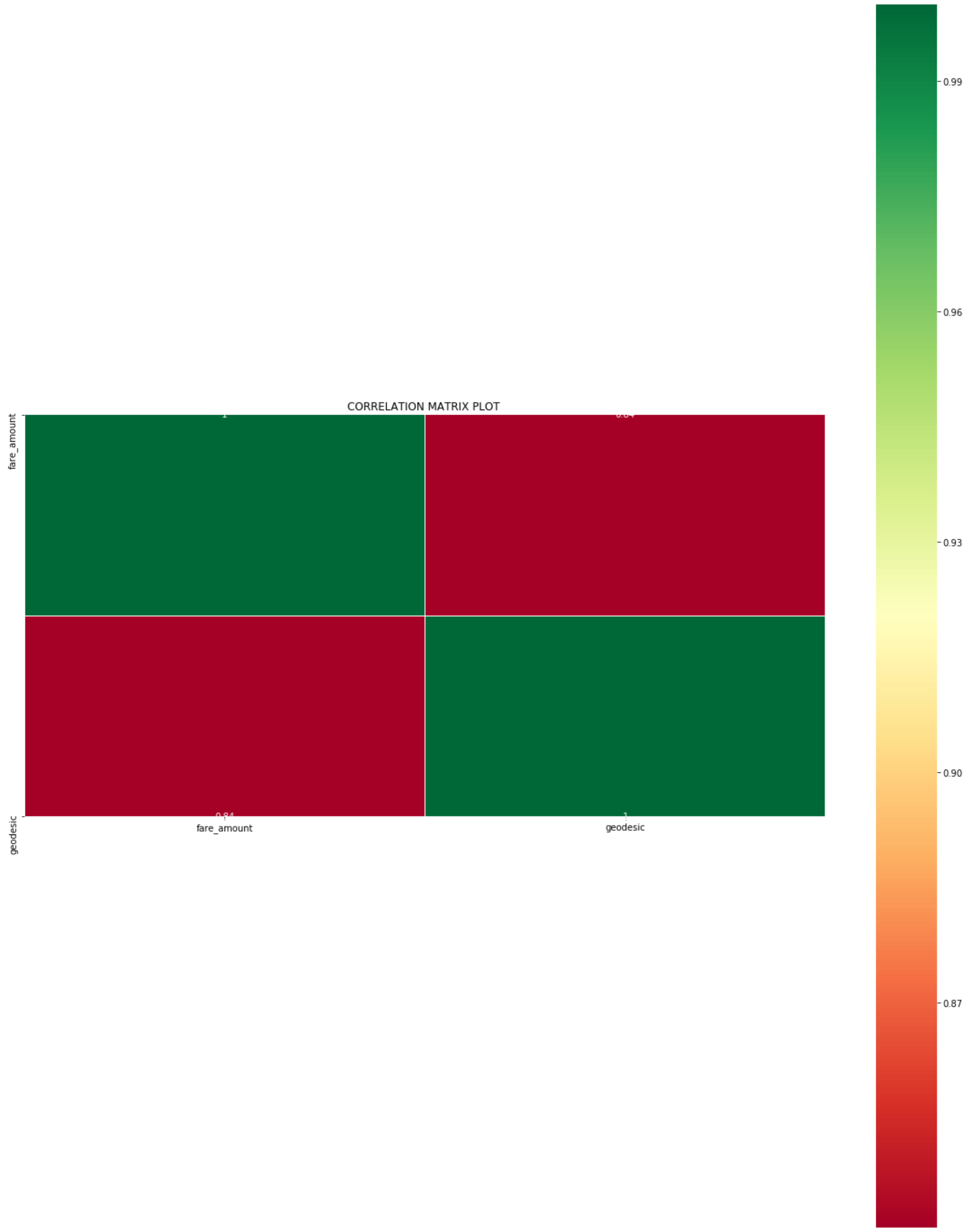

Fig A.2:Heatmap Analysis

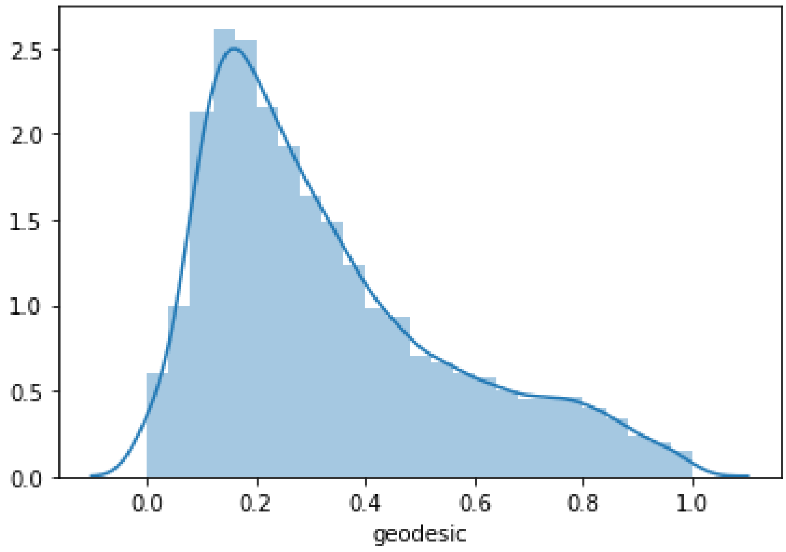

Fig A.3: Histogram of geodesic

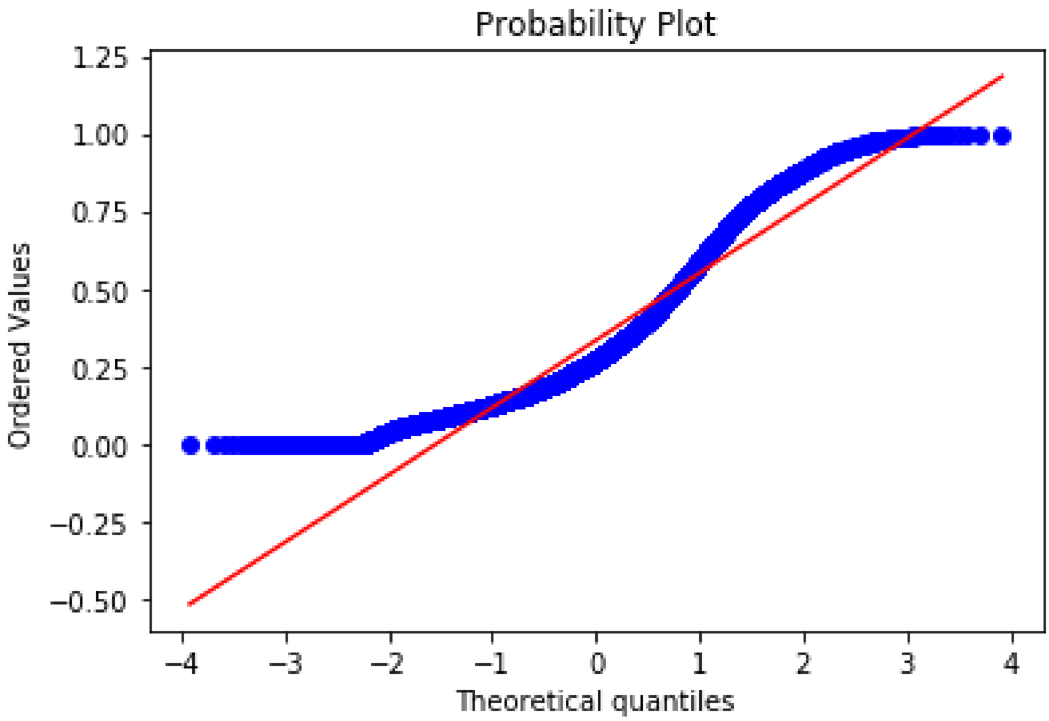

Fig A.4:Probability plot

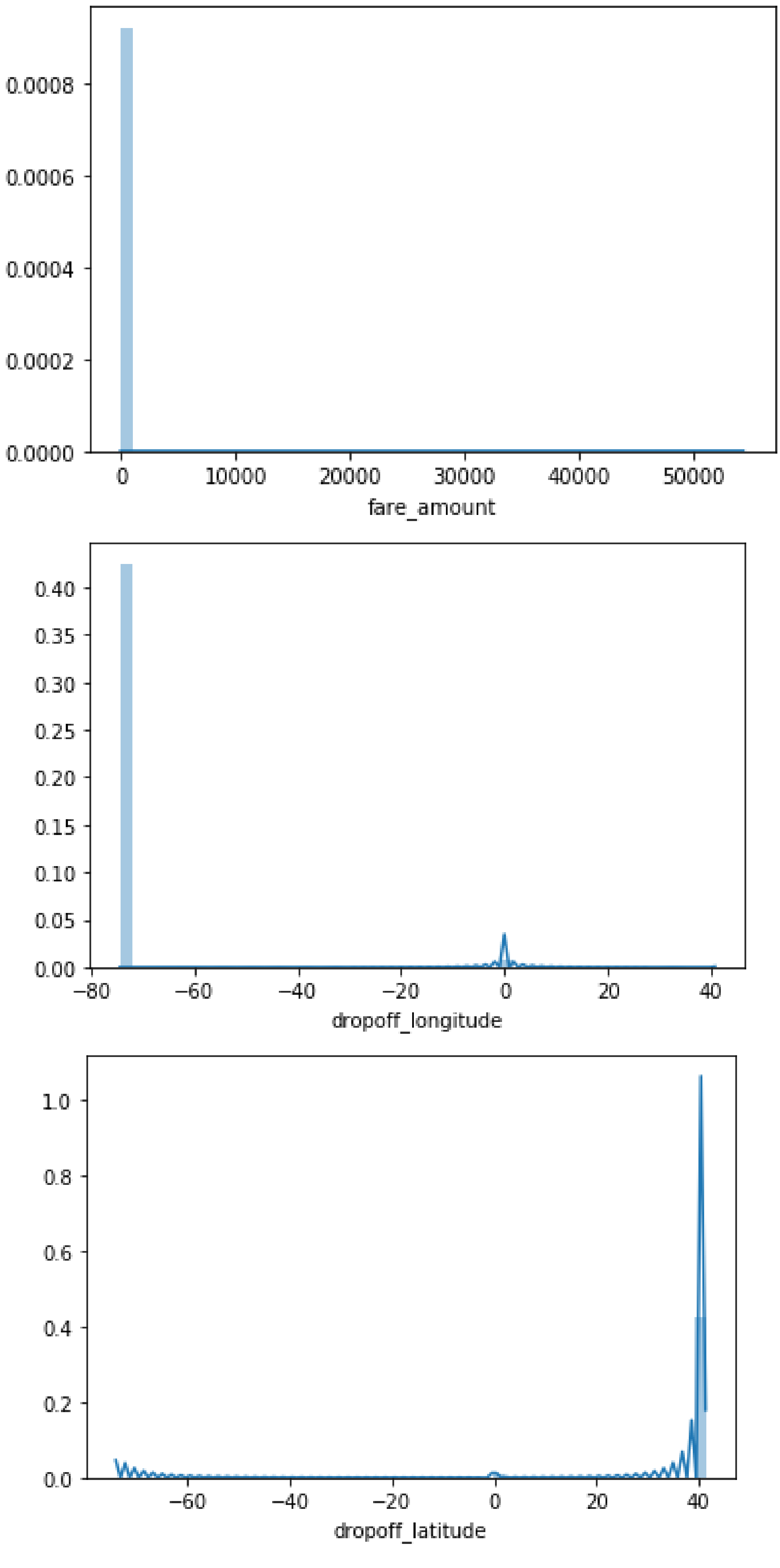

Fig A.5: Histograms of predictors

APPENDIX B

python CODE

```
# LOAD THE REQUIRED LIBRARIES
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.pyplot as plt
import scipy.stats as stats
#import fancyimpute
from fancyimpute import KNN
import warnings
warnings.filterwarnings('ignore')
from scipy.stats import chi2_contingency
import statsmodels.api as sm
from statsmodels.formula.api import ols
from patsy import dmatrices
from statsmodels.stats.outliers_influence import variance_inflation_factor
from xgboost import XGBRegressor
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn import metrics
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.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.externals import joblib
from geopy.distance import geodesic
from geopy.distance import great_circle
##Load the  data into python environment
```

```
#training=pd.read_csv("train_cab.csv")
training =
pd.read_csv('train_cab.csv',dtype={'fare_amount':np.float64},na_values={'fare_amount':'430-
'})
testing=pd.read_csv("test.csv")
car_data=[training,testing]
training.shape
testing.shape
for i in car_data:
    i['pickup_datetime']  = pd.to_datetime(i['pickup_datetime'],errors='coerce')
categorical_pass=['passenger_count']
categorical_pass
numerical_var=['fare_amount','pickup_longitude','pickup_latitude','dropoff_longitude','dropof
f_latitude']
plot_fa = sns.distplot(training['fare_amount'],bins=50)
plot_drop_lat1= sns.distplot(training['dropoff_latitude'],bins=50)
plot_drop_long1= sns.distplot(training['dropoff_longitude'],bins=50)
plot_pick_long1=sns.distplot(training['pickup_longitude'],bins=50)
sns.distplot(training['pickup_latitude'],bins=50)
training[training['fare_amount']<1]
training = training.drop(training[training['fare_amount']<1].index, axis=0)
for i in range(4,11):
    sum(training['passenger_count']>i)
sum(training['passenger_count']>i)
for i in range(4,11):
    print('PASSENGER COUNT MORE THAN '  +str(i)+' ARE:
{}'.format(sum(training['passenger_count']>i)))
training[training['passenger_count']>6]
training = training.drop(training[training['passenger_count']>6].index, axis=0)
training = training.drop(training[training['passenger_count']<1].index, axis=0)
sum(training['pickup_longitude']>180)
sum(training['dropoff_longitude']>180)
sum(training['pickup_longitude']<-180)
sum(training['dropoff_longitude']<-180)
```

```
sum(training['pickup_latitude']>90)
sum(training['dropoff_latitude']>90)
sum(training['pickup_latitude']<-90)
sum(training['pickup_latitude']<-90)
training = training.drop(training[training['pickup_latitude']>90].index, axis=0)
zero_val=['pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitude']
for i in zero_val:
    training = training.drop(training[training[i]==0].index, axis=0)
df=training.copy()
training=df.copy()
mising_value = pd.DataFrame(training.isnull().sum())
mising_value
mising_value = mising_value.reset_index()
mising_value = mising_value.rename(columns = {'index': 'Variable', 0:
'Missing_percentage'})
mising_value['Missing_percentage'] =
(mising_value['Missing_percentage']/len(training))*100
mising_value
a1=training['fare_amount'].loc[1000]
print('fare_amount at loc-1000:{}'.format(a1))
# Replacing 1.0 with NA
training['fare_amount'].loc[1000] = np.nan
print('Value after replacing with nan:{}'.format(training['fare_amount'].loc[1000]))
# Impute with mean
print('Value if imputed with
mean:{}'.format(training['fare_amount'].fillna(training['fare_amount'].mean()).loc[1000]))
# Impute with median
print('Value if imputed with
median:{}'.format(training['fare_amount'].fillna(training['fare_amount'].median()).loc[1000])
)
columns=['fare_amount', 'pickup_longitude', 'pickup_latitude','dropoff_longitude',
'dropoff_latitude', 'passenger_count']
pickup_datetime=pd.DataFrame(training['pickup_datetime'])
```

```
training = pd.DataFrame(KNN(k =
17).fit_transform(training.drop('pickup_datetime',axis=1)),columns=columns,
index=training.index)
training['passenger_count']=training['passenger_count'].astype('int')
training['passenger_count']=training['passenger_count'].round().astype('object').astype('categ
ory',order=True)
training['passenger_count'].unique()
# MISSING value analysis
missing_value = pd.DataFrame(pickup_datetime.isnull().sum())
missing_value = missing_value.reset_index()
missing_value
df_1=training.copy()
training=df_1.copy()
#BOXPLOT for outlier analysis
plt.figure(figsize=(10,10))
plt.xlim(0,100)
_ =
sns.boxplot(x=training['fare_amount'],y=training['passenger_count'],data=training,orient='h')
plt.show()
def outlier_analysis(columns):
   ''' calculating outlier indices and replacing them with NA  '''
   #Extract quartiles
   q75, q25 = np.percentile(training[columns], [75 ,25])
   print(q75,q25)
   #Calculate IQR
   iqr = q75 - q25
   #Calculate inner and outer fence
   minimum = q25 - (iqr*1.5)
   maximum = q75 + (iqr*1.5)
   print(minimum,maximum)
   #Replace with NA
   training.loc[training[columns] < minimum,columns] = np.nan
   training.loc[training[columns] > maximum,columns] = np.nan
outlier_analysis('fare_amount')
```

```
pd.DataFrame(training.isnull().sum())
training = pd.DataFrame(KNN(k = 3).fit_transform(training), columns = training.columns,
index=training.index)
training['passenger_count']=training['passenger_count'].astype('int').round().astype('object').a
stype('category')
df_2 = training.copy()
training=df_2.copy()
#######  FEATURE ENGINEERING########### done
training = pd.merge(pickup_datetime,training,right_index=True,left_index=True)
training=training.reset_index(drop=True)
training.isna().sum()
training=training.dropna()
car_data = [training,testing]
for i in car_data:
    i["year"] = i["pickup_datetime"].apply(lambda row: row.year)
    i["month"] = i["pickup_datetime"].apply(lambda row: row.month)
    i["day_of_week"] = i["pickup_datetime"].apply(lambda row: row.dayofweek)
    i["hour"] = i["pickup_datetime"].apply(lambda row: row.hour)
def function1(x):
    '''  HOUR COLUMN SESSION '''
    if (x >=5) and (x <= 11):
        return 'morning'
    elif (x >=12) and (x <=16 ):
        return 'afternoon'
    elif (x >= 17) and (x <= 20):
        return'evening'
    elif (x >=21) and (x <= 23) :
        return 'night_PM'
    elif (x >=0) and (x <=4):
        return'night_AM'
ef function2(x):
    ''' MONTH COLUMN (YEAR SEASON)'''
    if (x >=3) and (x <= 5):
        return 'spring'
```

```
    elif (x >=6) and (x <=8 ):
        return 'summer'
    elif (x >= 9) and (x <= 11):
        return'fall'
    elif (x >=12)|(x <= 2) :
        return 'winter'
ef function3(x):
    '''  DAY OF WEEK COLUMN '''
    if (x >=0) and (x <= 4):
        return 'weekday'
    elif (x >=5) and (x <=6 ):
        return 'weekend'
training['session'] = training['hour'].apply(function1)
training['week'] = training['day_of_week'].apply(function3)
training['seasons'] = training['month'].apply(function2)
testing['session'] = testing['hour'].apply(function1)
testing['week'] = testing['day_of_week'].apply(function3)
testing['seasons'] = testing['month'].apply(function2)
dummy_var = pd.get_dummies(testing['year'], prefix = 'year')
testing = testing.join(dummy_var)

dummy_var = pd.get_dummies(training['year'], prefix = 'year')
training = training.join(dummy_var)

dummy_var = pd.get_dummies(training['session'], prefix = 'session')
training = training.join(dummy_var)

dummy_var = pd.get_dummies(testing['session'], prefix = 'session')
testing = testing.join(dummy_var)

dummy_var = pd.get_dummies(training['week'], prefix = 'week')
training = training.join(dummy_var)

dummy_var = pd.get_dummies(testing['week'], prefix = 'week')
```

```
testing = testing.join(dummy_var)
dummy_var = pd.get_dummies(training['seasons'], prefix = 'season')
training = training.join(dummy_var)
dummy_var = pd.get_dummies(testing['seasons'], prefix = 'season')
testing = testing.join(dummy_var)
dummy_var = pd.get_dummies(training['passenger_count'], prefix = 'passenger_count')
training = training.join(dummy_var)
dummy_var = pd.get_dummies(testing['passenger_count'], prefix = 'passenger_count')
testing = testing.join(dummy_var)
training=training.drop(['passenger_count_1','season_fall','week_weekday','session_afternoon',
'year_2009'],axis=1)
testing=testing.drop(['passenger_count_1','season_fall','week_weekday','session_afternoon','y
ear_2009'],axis=1)
car_data = [training,testing]
for i in car_data:
    i['great_circle']=i.apply(lambda x: great_circle((x['pickup_latitude'],x['pickup_longitude']),
(x['dropoff_latitude'],   x['dropoff_longitude'])).miles, axis=1)
    i['geodesic']=i.apply(lambda x: geodesic((x['pickup_latitude'],x['pickup_longitude']),
(x['dropoff_latitude'],   x['dropoff_longitude'])).miles, axis=1)
training=training.drop(['pickup_datetime','pickup_longitude', 'pickup_latitude',
       'dropoff_longitude', 'dropoff_latitude', 'passenger_count', 'year',
       'month', 'day_of_week', 'hour', 'session', 'seasons', 'week','great_circle'],axis=1)
testing=testing.drop(['pickup_datetime','pickup_longitude', 'pickup_latitude',
       'dropoff_longitude', 'dropoff_latitude', 'passenger_count', 'year',
       'month', 'day_of_week', 'hour', 'session', 'seasons', 'week','great_circle'],axis=1)
plt.figure(figsize=(20,5))
plt.xlim(0,100)
sns.boxplot(x=training['geodesic'],data=training,orient='h')
plt.title('geodesic plot ')
plt.show()
#Impute  missing values by applying KNN
training = pd.DataFrame(KNN(k = 3).fit_transform(training), columns = training.columns,
index=training.index)
var_categorical=['passenger_count_2',
```

```
        'passenger_count_3', 'passenger_count_4', 'passenger_count_5',
        'passenger_count_6', 'season_spring', 'season_summer',
        'season_winter', 'week_weekend',
        'session_evening', 'session_morning', 'session_night_AM',
        'session_night_PM', 'year_2010', 'year_2011',
        'year_2012', 'year_2013', 'year_2014', 'year_2015']
var_numerical=['fare_amount','geodesic']
training[var_categorical]=training[var_categorical].apply(lambda x: x.astype('category') )
testing[var_categorical]=testing[var_categorical].apply(lambda x: x.astype('category') )
var_categorical=['passenger_count_2',
        'passenger_count_3', 'passenger_count_4', 'passenger_count_5',
        'passenger_count_6', 'season_spring', 'season_summer',
        'season_winter', 'week_weekend',
        'session_evening', 'session_morning', 'session_night_AM',
        'session_night_PM', 'year_2010', 'year_2011',
        'year_2012', 'year_2013', 'year_2014', 'year_2015']
var_numerical=['fare_amount','geodesic']
training[var_categorical]=training[var_categorical].apply(lambda x: x.astype('category') )
testing[var_categorical]=testing[var_categorical].apply(lambda x: x.astype('category') )
# chi square test
for i in var_categorical:
    for j in var_categorical:
        if(i != j):
            chi2, p, dof, ex = chi2_contingency(pd.crosstab(training[i], training[j]))
            if(p < 0.05):
                print(i,",",j,"are dependent on each other with",p,':Eliminate')
            else:
                print(i,",",j,"are independent on each other with",p,':Retain')
model_anova = ols('fare_amount ~ 
C(passenger_count_2)+C(passenger_count_3)+C(passenger_count_4)+C(passenger_count_5
)+C(passenger_count_6)+C(season_spring)+C(season_summer)+C(season_winter)+C(week_
weekend)+C(session_night_AM)+C(session_night_PM)+C(session_evening)+C(session_mo
rning)+C(year_2010)+C(year_2011)+C(year_2012)+C(year_2013)+C(year_2014)+C(year_2
015)',data=training).fit()
```

```
table_anova = sm.stats.anova_lm(model_anova)
outcome, predictors = dmatrices('fare_amount ~
geodesic+passenger_count_2+passenger_count_3+passenger_count_4+passenger_count_5+p
assenger_count_6+season_spring+season_summer+season_winter+week_weekend+session_
night_AM+session_night_PM+session_evening+session_morning+year_2010+year_2011+y
ear_2012+year_2013+year_2014+year_2015',training, return_type='dataframe')
# calculating VIF for each individual Predictors
vi_factor = pd.DataFrame()
vi_factor["VIF"] = [variance_inflation_factor(predictors.values, i) for i in
range(predictors.shape[1])]
vi_factor["features"] = predictors.columns
sns.distplot(training['geodesic'],bins=25)
#Normalization
training['geodesic'] = (training['geodesic'] -
min(training['geodesic']))/(max(training['geodesic']) - min(training['geodesic']))
testing['geodesic'] = (testing['geodesic'] - min(testing['geodesic']))/(max(testing['geodesic']) -
min(testing['geodesic']))
stats.probplot(training['geodesic'], dist='norm', fit=True,plot=plt)
df_4=training.copy()
training=df_4.copy()
df5=testing.copy()
testing=df5.copy()
training=training.drop(['passenger_count_2'],axis=1)
testing=testing.drop(['passenger_count_2'],axis=1)
X = training.drop('fare_amount',axis=1).values
y = training['fare_amount'].values
X_training, X_testing, y_training, y_testing = train_test_split(X, y, test_size = 0.25,
random_state=42)
print(training.shape, X_training.shape, X_testing.shape,y_training.shape,y_testing.shape)

def rmsle(y1,y2):
    log1 = np.nan_to_num(np.array([np.log(v + 1) for v in y1]))
    log2 = np.nan_to_num(np.array([np.log(v + 1) for v in y2]))
```

```
    calc = (log1 - log2) ** 2
    return np.sqrt(np.mean(calc))
def scores(y1, y2):
    print('r square  ', metrics.r2_score(y1, y2))
    #print('Adjusted r square:{}'.format(1 - (1-metrics.r2_score(y1, y2))*(len(y1)-1)/(len(y1)-
X_training.shape[1]-1)))
    print('MAPE:{}'.format(np.mean(np.abs((y1 - y2) / y1))*100))
    print('MSE:', metrics.mean_squared_error(y1, y2))
    print('RMSE:', np.sqrt(metrics.mean_squared_error(y1, y2)))
def test_scores(model):
    print('Training Data Details')
    print()
    #Predicting result on Training data
    y_pred = model.predict(X_training)
    scores(y_training,y_pred)
    print('RMSLE:',rmsle(y_training,y_pred))
    print()
    print(' Testing Data Details')
    print()
    # Evaluating on Test Set
    y_predict = model.predict(X_testing)
    scores(y_testing,y_predict)
    print('RMSLE:',rmsle(y_testing,y_predict))
# Setup the parameters and distributions to sample from: param_dist
param_dist = {'copy_X':[True, False],
          'fit_intercept':[True,False]}
# Instantiate a Decision reg classifier: reg
reg = LinearRegression()

# Instantiate the gridSearchCV object: reg_cv
reg_cv = GridSearchCV(reg, param_dist, cv=5,scoring='r2')

# Fit it to the data
reg_cv.fit(X, y)
```

```
# Print the tuned parameters and score
print("Tuned Decision reg Parameters: {}".format(reg_cv.best_params_))
print("Best score is {}".format(reg_cv.best_score_))
# Create the regressor: reg_all
reg_all = LinearRegression(copy_X= True, fit_intercept=True)

# Fit the regressor to the training data
reg_all.fit(X_training,y_training)

# Predict on the test data: y_pred
y_pred = reg_all.predict(X_testing)

# Compute and print R^2 and RMSE
print("R^2: {}".format(reg_all.score(X_testing, y_testing)))
rmse = np.sqrt(mean_squared_error(y_testing,y_pred))
print("Root Mean Squared Error: {}".format(rmse))
test_scores(reg_all)

# Compute and print the coefficients
reg_coef = reg_all.coef_
print(reg_coef)

# Plot the coefficients
plt.figure(figsize=(15,5))
plt.plot(range(len(testing.columns)), reg_coef)
plt.xticks(range(len(testing.columns)), testing.columns.values, rotation=60)
plt.margins(0.02)
plt.savefig('linear coefficients')
plt.show()
# Setup the parameters and distributions to sample from: param_dist
param_dist = {'max_depth': range(2,16,2),
          'min_samples_split': range(2,16,2)}
```

```
# Instantiate a Decision Tree classifier: tree
dec_tree = DecisionTreeRegressor()

# Instantiate the gridSearchCV object: tree_cv
dec_tree_cv = GridSearchCV(dec_tree, param_dist, cv=5)

# Fit it to the data
dec_tree_cv.fit(X, y)

# Print the tuned parameters and score
print("Tuned Decision Tree Parameters: {}".format(dec_tree_cv.best_params_))
print("Best score is {}".format(dec_tree_cv.best_score_))

# Instantiate a tree regressor: tree
dec_tree = DecisionTreeRegressor(max_depth= 6, min_samples_split=2)

# Fit the regressor to the data
dec_tree.fit(X_training,y_training)

# Compute and print the coefficients
dec_tree_features = dec_tree.feature_importances_
print(dec_tree_features)

# Sort test importances in descending order
tree_indices = np.argsort(dec_tree_features)[::1]

# Rearrange test names so they match the sorted test importances
names_col = [testing.columns[i] for i in tree_indices]
test_scores(dec_tree)
```

R CODE

```
rm(list = ls())
# #loading Libraries
x = c("ggplot2", "corrgram", "DMwR", "usdm", "caret", "randomForest", "e1071",
   "DataCombine", "doSNOW", "inTrees", "rpart.plot", "rpart",'MASS','xgboost','stats')
#load Packages
lapply(x, require, character.only = TRUE)
rm(x)
# loading datasets
train = read.csv("train_cab.csv", header = T, na.strings = c(" ", "", "NA"))
test = read.csv("test.csv")
test_pickup_datetime = test["pickup_datetime"]
##                  Exploratory Data Analysis
#  data types of variables convertion
train$fare_amount = as.numeric(as.character(train$fare_amount))
train$passenger_count=round(train$passenger_count)
### Removing values which are not within desired range(outlier) depending upon basic
understanding of dataset.
# Fare amount cann't be -ve  and also cannot be 0.  remove these fields.
train[which(train$fare_amount < 1 ),]
nrow(train[which(train$fare_amount < 1 ),])
train = train[-which(train$fare_amount < 1 ),]
#2.Passenger_count
for (i in seq(4,11,by=1)){
  print(paste('passenger_count above ' ,i,nrow(train[which(train$passenger_count > i ),])))
}
#  20 observations of passenger_count are above from 6,7,8,9,10 passenger_counts
train[which(train$passenger_count > 6 ),]
# Check if there are any passenger_count =0
train[which(train$passenger_count <1 ),]
nrow(train[which(train$passenger_count <1 ),])
# Eliminate these 58 observations and 20 observation which are above 6 value .
train = train[-which(train$passenger_count < 1 ),]
train = train[-which(train$passenger_count > 6),]
```

```
# 3.Latitudes range from -90 to 90.Longitudes range from -180 to 180.
print(paste('pickup_longitude above 180=',nrow(train[which(train$pickup_longitude >180 ),])))
print(paste('pickup_longitude above -180=',nrow(train[which(train$pickup_longitude < -180 ),])))
print(paste('pickup_latitude above 90=',nrow(train[which(train$pickup_latitude > 90 ),])))
print(paste('pickup_latitude above -90=',nrow(train[which(train$pickup_latitude < -90 ),])))
print(paste('dropoff_longitude above 180=',nrow(train[which(train$dropoff_longitude > 180 ),])))
print(paste('dropoff_longitude above -180=',nrow(train[which(train$dropoff_longitude < -180 ),])))
print(paste('dropoff_latitude above -90=',nrow(train[which(train$dropoff_latitude < -90 ),])))
print(paste('dropoff_latitude above 90=',nrow(train[which(train$dropoff_latitude > 90 ),])))
# There's only one outlier which is in variable pickup_latitude.remove it with nan.
#  check 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 ),])
# values which are equal to 0.
train = train[-which(train$pickup_latitude > 90),]
train = train[-which(train$pickup_longitude == 0),]
train = train[-which(train$dropoff_longitude == 0),]
# Make a copy
df=train
# train=df

#############                    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)]
missing_val
unique(train$passenger_count)
unique(test$passenger_count)
train[,'passenger_count'] = factor(train[,'passenger_count'], labels=(1:6))
test[,'passenger_count'] = factor(test[,'passenger_count'], labels=(1:6))
# kNN Imputation
train = knnImputation(train, k = 181)
#sapply(train, sd, na.rm = TRUE)
df1=train
# train=df1
#####################                    Outlier Analysis              ##################
#
pl1 = ggplot(train,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
vals = train[,"fare_amount"] %in% boxplot.stats(train[,"fare_amount"])$out
train[which(vals),"fare_amount"] = NA
#lets check the NA's
sum(is.na(train$fare_amount))
#Imputing with KNN
train = knnImputation(train,k=3)
# lets check the missing values
sum(is.na(train$fare_amount))
str(train)
df2=train
# train=df2
##################                 Feature Engineering
#########################
# 1.Feature Engineering for timestamp variable
#
#Convert pickup_datetime from factor to date time
train$pickup_date = as.Date(as.character(train$pickup_datetime))
```

```
train$pickup_weekday = as.factor(format(train$pickup_date,"%u"))# Monday = 1
train$pickup_mnth = as.factor(format(train$pickup_date,"%m"))
train$pickup_yr = as.factor(format(train$pickup_date,"%Y"))
pickup_time = strptime(train$pickup_datetime,"%Y-%m-%d %H:%M:%S")
train$pickup_hour = as.factor(format(pickup_time,"%H"))

#Add same features to test set
test$pickup_date = as.Date(as.character(test$pickup_datetime))
test$pickup_weekday = as.factor(format(test$pickup_date,"%u"))# Monday = 1
test$pickup_mnth = as.factor(format(test$pickup_date,"%m"))
test$pickup_yr = as.factor(format(test$pickup_date,"%Y"))
pickup_time = strptime(test$pickup_datetime,"%Y-%m-%d %H:%M:%S")
test$pickup_hour = as.factor(format(pickup_time,"%H"))

sum(is.na(train))# there was 1 'na' in pickup_datetime which created na's in above feature
engineered variables.
train = na.omit(train) # we will remove that 1 row of na's

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

# 2.Calculate the distance travelled using longitude and latitude
deg_to_rad = function(deg){
  (deg * pi) / 180
}
haversine = function(long1,lat1,long2,lat2){
  #long1rad = deg_to_rad(long1)
  phi1 = deg_to_rad(lat1)
  #long2rad = deg_to_rad(long2)
  phi2 = deg_to_rad(lat2)
  delphi = deg_to_rad(lat2 - lat1)
  dellamda = deg_to_rad(long2 - long1)

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

```
   sin(dellamda/2) * sin(dellamda/2)

  c = 2 * atan2(sqrt(a),sqrt(1-a))
  R = 6371e3
  R * c / 1000 #1000 is used to convert to meters
}
# Using haversine formula to calculate distance fr both train and test
train$dist =
haversine(train$pickup_longitude,train$pickup_latitude,train$dropoff_longitude,train$dropof
f_latitude)
test$dist =
haversine(test$pickup_longitude,test$pickup_latitude,test$dropoff_longitude,test$dropoff_lat
itude)
# 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))
#################                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 * pickup_hour * pickup_weekday,data =
train)
aov_results = aov(fare_amount ~ passenger_count + pickup_hour + pickup_weekday +
pickup_mnth + pickup_yr,data = train)

summary(aov_results)
```

```
# pickup_weekdat has p value greater than 0.05
train = subset(train,select=-pickup_weekday)
#remove from test set
test = subset(test,select=-pickup_weekday)

######## Feature Scaling        ########
#Normality check
# qqnorm(train$fare_amount)
# histogram(train$fare_amount)
library(car)
# dev.off()
par(mfrow=c(1,2))
qqPlot(train$fare_amount)                             # qqPlot, it has a x values derived from gaussian
distribution, if data is distributed normally then the sorted data points should lie very close to
the solid reference line
truehist(train$fare_amount)                           # truehist() scales the counts to give an estimate
of the probability density.
lines(density(train$fare_amount))  # Right skewed      # lines() and density() functions to
overlay a density plot on histogram
#Normalisation
print('dist')
train[,'dist'] = (train[,'dist'] - min(train[,'dist']))/
  (max(train[,'dist'] - min(train[,'dist'])))
# #check multicollearity
# library(usdm)
# vif(train[,-1])
#
# vifcor(train[,-1], th = 0.9)

#################### Splitting train into train and validation subsets
set.seed(1000)
tr.idx = createDataPartition(train$fare_amount,p=0.75,list = FALSE) # 75% in trainin and
25% 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_pickup_datetime'))
####################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)
*******************************************************************************
```