Cab Fare Prediction

R. B. Borate

22/07/2020

INDEX

CONTENTS PAGE NO.

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.

So the Objective of this project is to build the Machine learning model on the given data to predict the cab fare.

1.2 Data:

Let's see the pilot project past data that can be used to build the model. We have got train_cab and test data with the problem statement. So let's look first the top five rows of train_cab data given to us is shown below. It contains total 16067 observations & 7 variables from which 6 are dependant and first on is target variable.

Table.1. Given train_cab data with top 5 observations.

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	-73.841610	40.712278	1.0
16.9	2010-01-05 16:52:16 UTC	-74.016048	40.711303	-73.979268	40.782004	1.0
5.7	2011-08-18 00:35:00 UTC	-73.982738	40.761270	-73.991242	40.750562	2.0
7.7	2012-04-21 04:30:42 UTC	-73.987130	40.733143	-73.991567	40.758092	1.0
5.3	2010-03-09 07:51:00 UTC	-73.968095	40.768008	-73.956655	40.783762	1.0

In the above data we can see there are total 7 variable out of which fare_amount is our target variable. Other 6 variables are predictors. But variables are not in proper format we need to do feature engineering of these variables because if we see pickup_datetime variable is containing information about date and time and the next 4 variables contains information about locations. Last variable is giving persons travelled in each ride of cab. So this is the brief idea of train_cab data. Now let's look the given test data.

Table.2. Given test data with top 5 observations.

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

In the above test data we can see the variables are same just we don't have the fare_amount variable here in this data. So we need to build the model on train_cab data and predict the cab fare for this test data that is our objective.

CHAPTER 2

METHODOLOGY

2.1 Exploratory data analysis:

At first we have to set the working directory and load the libraries which are required to perform different operations. So we have uploaded the libraries and loaded the data. Now we will do the exploratory analysis of data that is nothing but visualizing the data and converting the data according to our requirement. It contains exploring the data, cleaning and visualizing with help of some plots and graphs.

We have already looked in to data shortly. Now we will look into data in detail in this step, so we will visualize the data in better way by using some plots and exploring the data by some statistical measures, sorting and basic commands to see the details of data.

2.1.1 Understanding the independent variables in our data-

Now we will see the data types of independent variable first. We can see the data types of each variable below,

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

1. fare_amount- sample value- (4.5 **)**

we can see the data type of fare_amount is object but the values in the variable are of float data type so we need to convert this variable to numeric data type. Also this is our target variable and which is in 1st column of our data set.

2. **Pickup_datetime- sample value- (**2015-01-27 13:08:24 UTC)

We can see the sample value of pickup_datetime variable from that we can easily say that this variable is a time stamp variable. So we will convert it to date time format. Also it contains date and time information so we can extract new features from it like date, month, year etc.

3. **pickup_longitude- sample value- (**-73.973320)

Longitudes are nothing but the imaginary vertical lines on the earth that meet at north and South Pole. These are spread over the earth 180 lines in east and 180 lines in west so total of 360. So these are taken as **-180 to +180.** The center of longitude line is a 0 degree and which pass from Greenwich, England. So 180 degrees to east and 180 degrees to west forms a great circle around the earth.

Now our variable pickup_longitude is nothing but the location of cab from where the passenger is pickup up. It is a pickup location on longitude line.

4. **dropoff_longitude - sample value- (**-73.981430**)**

This is the same as pickup_longitude variable, this is the location of cab where the passenger is dropped off. It is the drop off location on longitude line.

5. **pickup_latitude- - sample value- (**40.763805**)**

Latitudes are nothing but the angle ranges from 0 degree (at equator i.e. horizontal centre line on earth) to 90 degree (north). So these are the imagnary horizontal lineson the great circle formed by longitudes. Longitude and Latitude gives us the exact distance of any object on the earth.

In our case **pickup_latitude** is the pickup location of passenger on the latitude line. It ranges from **-90 to +90**

6. dropoff_latitude- sample value- (40.743835**)**

This is the same as **pickup_latitude** variable, this is the location of cab where the passenger is dropped off. It is the drop off location on latitude line.

7. **passenger_count- sample value** –(1)

This variable gives us the information about how many passengers have travelled in the cab in each ride. The data type of variable is float but it should be a integer because person count can never be a float variable.

So this is about the basic understanding of our variables from train data. The test data has same variables except the fare_amount variable. So let's move further to understand the data and clean the data to make it in the format to be used for ML models.

2.1.2 Understanding and cleaning the data:

So in this section we will understand the variables of our data by using different commands of data exploration. We will understand and clean train_cab data first and then apply the same operations on test data set.

Let's see the statistical measures of all the variables, table 4 below shows the description of train data,

Table.3. Statistical measures of train_cab data

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

We can see the train_cab data statistical measures in the above table. We know that latitude ranges from -90 to +90 and longitude ranges from -180 to +180.

-But in above train_cab data table pickup_latitude variable maximum value is above 90 so we need to remove those values. All other values of latitide and longitude variables are within range.

-Also passenger_count variable contains minimum value as '0.0' and maximum value as '5345.0' and its data type is float. So we also need to solve this issue.

Now let's do the data cleaning of variables from train data.

1) fare_amount-

Now we will sort the fare_amount variable in descending order to see its maximum value, that we can see below.

```
1015    54343.0
1072     4343.0
607       453.0
980       434.0
1335      180.0
          ...
1712        NaN
2412        NaN
2458        NaN
8178        NaN
8226        NaN
```

We can see the maximum value of fare above 453 is above the range and not practical. Also it contains missing values that we deal in missing value analysis. So we will remove the rows in the fare_amount which are above 453 and will check for 0 value rows and remove that also. We have got the 7 observations so we will remove it as shown in below code.

```
train_cab = train_cab.drop(train_cab[train_cab['fare_amount']<1].index, axis=0)
train_cab = train_cab.drop(train_cab[train_cab['fare_amount']>453].index, axis=0)
```

2) **passenger_count-**

We gave seen in the table 3 passenger_count has the minimum value 0 and maximum value as 5345, but it is not practical maximum we can assume 6 passengers in one cab. We can see below in box plot passenger count values are passing above 20.

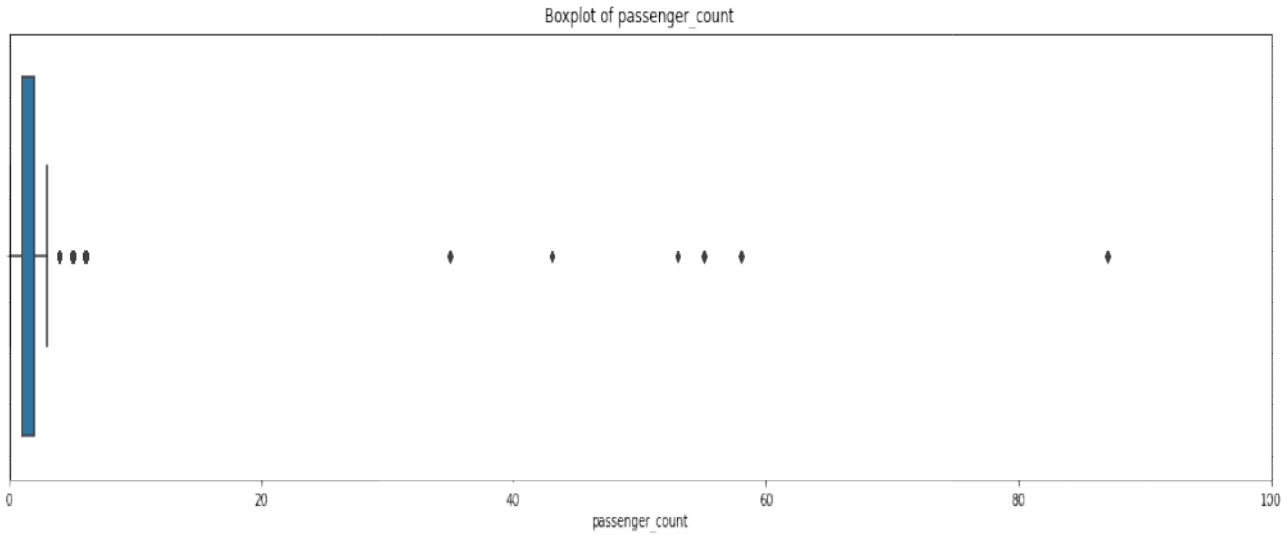


Fig. 1 Box plot of passenger count from train_cab data.

So we assume that there should be at least one passenger and maximum six. So we will remove the observations who have passenger_count more than '6' and less than '1'. Also there is one observation with value 1.3 so we will remove that also. Below is the code for this operation.

```
# Remove passenger_count above '6' and below '1'
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)

# Removing the observations in passenger_count with 1.3 value.
train_cab = train_cab.drop(train_cab[train_cab['passenger_count']==1.3].index, axis=0)
```

3) **Location variables** (pickup_longitude,pickup_latitude, dropoff_longitude, dropoff_latitude)-

Now we will do the data cleaning of location variables. So we will look all the variables for value equal to '0' in each variable. There are total 311, 311, 312, 310 rows that we got in each variable respectively. So we will remove all these rows from each variable. The code for this is as shown below.

```
# Remove observations with '0' value
for i in location_var:
    train_cab = train_cab.drop((train_cab[train_cab[i]==0]).index, axis=0)
```

So we have completed the data cleaning of train_cab data now we will proceed further for data preprocessing methods on this data. The test data is within the range so no need of this step.

2.2 Data Preprocessing:

2.2.1 Missing Value Analysis:

We need to do this analysis to check is there any missing values present in the data. We will do this on whole data and then find the variables who is having missing value percentage is less than 30 % if any variables is having more than 30% missing values then we will skip that variable directly. Now after getting the variables of missing values we will impute that values by using one of the method from *'Mean', 'Median' and 'KNN*, imputation method.

Now we will check our train_cab data for missing values. For that we have created a data frame of missing values and variable names. Then sorted that data frame in descending order and calculated the missing percentage for each variable which is as shown below

Variables	Missing_Percentage
passenger_count	0.351258
fare_amount	0.140503
pickup_datetime	0.006387
pickup_longitude	0.000000
pickup_latitude	0.000000
dropoff_longitude	0.000000
dropoff_latitude	0.00000

We can see there are missing values in top three variables and the count of missing values is 78 and percentage is as shown above. Here the amount of missing values as compare to whole data is very less. We can remove these rows also but we will impute them. Below is the procedure to impute missing values.

Procedure to Impute missing values is as follows

1) Select any random observation from data having its value and equal it with "NA"

2) Now impute that value by using mean, mode, median or KNN Method

3) Compare the value imputed by above methods with actual value.

4) Select the method which will give more accurate result

5) Now choose that method and find all missing values in that variable.

6) Repeat above steps t0 impute all missing values.

So I have applied above steps on our train_cab data variables fare_amount and passenger count variables. The row with missing value in pickup_datetime variable is only one so I have removed that row instead of imputing it.

Now for the fare_amount variable I have imputed the value with different methods listed above and form that I found Median is giving good results in python so I have imputed all the missing values by using median in python code. In R code I have used KNN imputation method to impute missing values.

Now for the passenger count variable as it is categorical variable we know that mode is best method but we can not use this method in our data, because we have more observations for passenger one count. So if we have used mode method the imputed values will be '1'. So instead of mode I have imputed all the missing values by using mean in python code and then rounded off the values and In R code I have used KNN imputation method to impute missing values.

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

After imputing the missing values I have again checked the data for missing values and I have got 0 missing values in all the variables as shown in above.

2.2.2 Outlier Analysis:

This step is used for finding the skewed data entries present in the particular variables. We will find it and then replace that values with 'na' and then impute that 'na' values with one of the method from *'Mean', 'Median' and 'KNN, imputation method.* The outlier analysis can be done on numeric variables only. We will use very powerful tool to do outlier analysis that is Tukey's method. In this method the box plots are plotted for each variable. That may be by taking only values of particular variable or w.r.t. to target variable. Box plot contains lower fence and upper fence at the lower and top portion of box plot. Middle of plot is Median. The values crossing the lower or upper fence will be termed as outliers.

For our data we will not do outlier analysis because it is having more data points biased towards lower values. So instead of outlier analysis we will do log transform of our numeric variables in feature scaling step as we move further in this project.

2.2.3 Feature Engineering:

This is the advanced step in data preprocessing. This is not mandatory step but sometimes depending upon the data we need to feature engineering of variables. In this step actually we will be creating new features from our existing features to convert the variables that we have in more meaningful way and apply those variables to Machine Learning model to get good results.

- **pickupdate_time-**

So if we see our independant variables we have **pickupdate_time** variable which contains information of day, month, year and time in which we can get hour, minute and seconds at that point from where the ride started. But these all are enclosed in one format although wehave converted them to date time format but still we can't use that variables as it is for ML models because our ML model didn't recognize these variables. So we need to create new variables from this variable which we can fit to Ml models.

So we have split pickupdate_time variable and created new variables like **"date", "month", "year", "weekday", "hour" and "minute".** The code for that is shown below.

```
# Now we will seperate pickup_datetime variable
# create new variables like date, month, year, weekday, hour, minute
train_cab['date']= train_cab['pickup_datetime'].dt.day
train_cab['month']= train_cab['pickup_datetime'].dt.month
train_cab['year']= train_cab['pickup_datetime'].dt.year
train_cab['weekday']= train_cab['pickup_datetime'].dt.dayofweek
train_cab['hour']= train_cab['pickup_datetime'].dt.hour
train_cab['minute']= train_cab['pickup_datetime'].dt.minute
```

The train_cab data after creating these variables is having 13 variables, from that pickup_datetime original variable along with new variables are shown below in table 4,

Table 4 Variables extracted from pickup_datetime variable.

pickup_datetime	date	month	year	weekday	hour	minute
2009-06-15 17:26:21+00:00	15	06	2009	0	17	26
2010-01-05 16:52:16+00:00	5	01	2010	1	16	52
2011-08-18 00:35:00+00:00	18	08	2011	3	00	35
2012-04-21 04:30:42+00:00	21	04	2012	5	04	30
2010-03-09 07:51:00+00:00	9	03	2010	1	07	51

We can see the new variables created which will give us better information about our target variable. From above variables minute variable is just the showing the minutes after the particular hour at which the cab ride is started as we have already the time in hour of cab ride start we don't require minute variable for our ML model as it will only increase time of computation and does not add any value to the data. So we will remove this variable afterwards.

So we have completed feature engineering of pickup_datetime variable from train _cab data set. We will repeat the steps above on test data set to create the same variables.

- **Location variables** (pickup_longitude,pickup_latitude, dropoff_longitude, dropoff_latitude)-

Similarly if we see other four variables from our train_cab data set we have pickup_longitude, pickup_latitude, dropoff_longitude, dropoff_latitude. We already discussed about longitude and latitude in our EDA. These are nothing but the imaginary lines on the surface of earth vertically and horizontally forming the grid like structure and circle around the earth.

The combination of longitude and latitude describes the exact location of any object on the earth surface. So in our data set we have both pickup and drop off longitudes and latitudes of passenger from where it is picked up and dropped by cab ride. But if we pass these variables as it is to our Ml model it will not recognize it. So we need to convert these variables to new meaningful variables, so that we can relate that variable with our target variable and extract some important information from it.

So by using these four location variables we will create the new variable 'distance' which will give us the distance travelled during each cab ride. The **"distance"** can be the important variable for deciding the fare_amount.

There are two formulas or methods we can say for calculating the distance from the pair of longitude and latitude.

i) Haversine-

The haversine formula calculates the great circle distance between two points on the sphere. Generally the two points are described by the pair of longitude and latitude. So in our data set we have longitude and latitude of pickup and drop location of passenger.

We have used the haversine formula calculate the distance in R code which we can see the below,

```
### lets write Haversine formulla to calculate distance
# lets create function for Havershine formula and store in Haversine object
Haversine= function(pi_lon,pi_lat, dr_lon, dr_lat){

  ### where pi_lon=pickup_longitude,    pi_lat= pickup_latitude,
  ###       dr_lon= dropoff_longitude,  dr_lat=dropoff_latitude

  # now lets convert pickup_latitude and dropoff_latitude to radians and save in pi_lon_rad and dr_lon_rad to objectpi_lon_rad=Radians(pi_lon)
  dr_lon_rad=Radians(dr_lon)

  # lets substract them adn save in sub_lon
  sub_lon= Radians(dr_lon-pi_lon)
```

```
 # now lets convert pickup_latitude and dropoff_latitude to radians and save in pi_lat_rad and dr_lat_rad to object
 pi_lat_rad=Radians(pi_lat)
 dr_lat_rad=Radians(dr_lat)

 # lets substract them and save in sub_lat
 sub_lat= Radians(dr_lat-pi_lat)

 # lets write the formula
 H=sin(sub_lat/2)*sin(sub_lat/2)+ cos(pi_lat_rad)*cos(dr_lat_rad)* sin(sub_lon/2)* sin(sub_lon/2)
 A=2*atan2(sqrt(H), sqrt(1-H))
 R=6371e3
 R*A/1000
}
```

We can see above the code with formula of Haversine to calculate the distance the steps in calculations are as follows,

a) First the function with Haversine is created, in that function the arguments that should be given are pickup_longitude, pickup_latitude, dropoff_longitude and droopoff_latitude.
b) Now inside the function in the body of function we have converted all the latitude and longitude variables to radians from degrees
c) In next step converted latitudes and longitudes are subtracted from each other and stored in another object as we can see above in code. i.e.(dropoff_longitude- pickup_longitude) and(droopoff_latitude- pickup_latitude)/
d) Next step is to use haversine formula which will calculate haversine of central angle (ɵ =d/r where d= distance between two points and r is radius of sphere) from latitude and longitude. So we have filled the converted variables to this formula and stored in the variable "H" then to get the distance inverse haversine is applied to H and stored in 'A'
e) The multiplication of 'R' and 'A' will give us the distance in meters. Where R is radius of earth.
f) Finally it is divided by 1000 to convert the distance in kilometers.

So by using above steps we have created the distance variable for our data. Haversine formula assumes earth as a complete sphere, but in actual practice we know that earth is not a complete sphere in shape its radius varies from North Pole to South Pole. The radius of earth at pole is less than at equator. So we can say that the distance calculated by haversine formula is good but not accurate.

Table 5 New created distance variable in train_cab (top 3 rows)

pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	distance
-73.84431	40.72132	-73.84161	40.71228	1.030764
-74.01605	40.71130	-73.97927	40.78200	8.450134
-73.98274	40.76127	-73.99124	40.75056	1.389525

There is another method available which is used to calculate the distance from pair of longitude and latitude named as vincenty formula. That we will see now in detail.

ii) Vincenty-

Vincenty formula calculate geodesic distance between latitude/longitude points on ellipsoidal model of earth. As it calculates the distance by considering the earth as elliptical shape not sphere, hence this method is more accurate than haversine. Because in actual earth is not exactly like sphere its radius at poles is less than at equator. The accuracy of this formula is within 0.5mm which is very good if consider in terms of earth dimensions. This formula was developed by person Vincenty in 1975

We have used this formula to calculate distance variable for our train_cab data in python code and created new variable called distance from the longitude and latitude location variables that we have in our data. We have the library called "*geopy.disatance"* in python to calculate the distance from latitude and longitudes directly. In that library there is function called "geodesic" which calculates the distance by vincenty formula just we need to give the input arguments as latitude and longitude values.

So for our train_cab data we have passed all our location variables to "geodesic" function and we have calculated distance variable. The python code for this is shown below,

```
# Load the library
from geopy.distance import geodesic
# lets create distance variable
train_cab['distance']=train_cab.apply(lambda y:
geodesic((y['pickup_latitude'],y['pickup_longitude']), (y['dropoff_latitude'],
y['dropoff_longitude'])).km, axis=1)
```

Table 6 New created distance variable in train_cab (top 3 rows)

pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	distance
-73.84431	40.72132	-73.84161	40.71228	1.029601
-74.01605	40.71130	-73.97927	40.78200	8.443441
-73.98274	40.76127	-73.99124	40.75056	1.389132

We can compare the distance variable values by haversine formula and vincenty formula. There is difference in values but not very big difference so both are good but vincenty is more accurate comparatively.

Now we will check the values of new variable distance for outliers. For that we will sort the values in descending order. The sorted values we can see below,

```
5864     5434.774894
7014     4429.159836
10710     129.767395
14536     129.378017
11619     127.329567
             ...
4365        0.000000
13000       0.000000
4367        0.000000
12973       0.000000
4559        0.000000
Name: distance, Length: 15657, dtype: float64
```

We can see the maximum value is 5434.77 which is not possible to travel cab also there are values equal to 0 that is also not practical. The values above 123.76 a=have increased drastically. So we will remove the rows which are above ‘130’ and equal to “0”. The python code for this is shown below,

```
# lets remove observations with '0' and more than '130' values from train data
train_cab=train_cab.drop(train_cab[train_cab['distance']==0].index,axis=0)
train_cab=train_cab.drop(train_cab[train_cab['distance']>130].index,axis=0)
```

- Removing the variables used for feature engineering-

In this step we will remove the variables that we have used and extracted new features from it. This step is necessary because unwontedly keeping the variables will increase the computation

time unnecessarily. We have used pickup_datetime, pickup_longitude, pickup_latitude, dropoff_longitude, dropoff_latitude variables to create new variables. So we will remove all these from both train and test data. Also we will remove minute variable as it is not looking so important w.r.t. to target variable. For this operation we have stored all the variables in "drop_variables" object and passed it by drop function in pandas library to remove these variables form both the data sets. The python code for this is as shown below,

```
# lets store variables to drop in one variable

drop_variables= ['pickup_datetime', 'pickup_longitude', 'pickup_latitude',
'dropoff_longitude', 'dropoff_latitude', 'minute']
# lets drop above variables from train_cab data
train_cab= train_cab.drop(drop_variables,axis=1)
test_cab= test_cab.drop(drop_variables,axis=1)
```

The train_cab data top 5 rows after dropping variables is shown below,

	fare_amount	passenger_count	date	month	year	weekday	hour	distance
0	4.5	1.0	15	6	2009	0	17	1.029601
1	16.9	1.0	5	1	2010	1	16	8.443441
2	5.7	1.0	18	8	2011	3	0	1.389132
3	7.7	1.0	21	4	2012	5	4	2.795790
4	5.3	1.0	9	3	2010	1	7	1.998338

After data cleaning and feature engineering we have following data left.

```
Data before preprocessing
train_cab        test
(16067, 7),  (9914, 6)

Data after preprocessing
train_cab        test
((15500, 8), (9829, 7))
```

So we can see in train_cab data we lost "500" observations and one variable is added in data. In test data we lost "85" observations and one variable is added in the data. Now lets see the data type of all the variables in train_cab data. Which is shown below

```
fare_amount        float64
passenger_count    float64
date               float64
```

```
month                float64
year                 float64
weekday              float64
hour                 float64
distance             float64
dtype: object
```

We can see the data type is float for all the variables and we know that new variables like month, date, year etc cannot be of float data type so we will convert all new variables to “int” data type from train_cab and test both data sets. Even if we convert them as a ‘int’ we know that those are the categorical variables for visualizations and for ML models simplicity we have converted them to int data type. The python code for this operation is shown below.

```
# we can see in data the new variables that we created are of 'float' data type
# so we will convert all to 'int'
train_cab['passenger_count']= train_cab['passenger_count'].astype('int64')
train_cab['date']= train_cab['date'].astype('int64')
train_cab['month']= train_cab['month'].astype('int64')
train_cab['year']= train_cab['year'].astype('int64')
train_cab['weekday']= train_cab['weekday'].astype('int64')
train_cab['hour']= train_cab['hour'].astype('int64')


#we can see in data the new variables that we created are of 'float' data type
#so we will convert all variables to 'int' data type
test_cab['passenger_count']= test_cab['passenger_count'].astype('int64')
test_cab['date']= test_cab['date'].astype('int64')
test_cab['month']= test_cab['month'].astype('int64')
test_cab['year']= test_cab['year'].astype('int64')
test_cab['weekday']= test_cab['weekday'].astype('int64')
test_cab['hour']= test_cab['hour'].astype('int64')
```

2.2.4 Data Visualizations:

This is actually the part of exploratory data analysis. But our given data is not in proper format. Also we need to do feature engineering on the data. That’s why after doing that and cleaning the data now we will visualize the data to understand it in better way.

We will plot different graphs, charts and plots w.r.t. our target variable and independent variables. We will take our independent variables one by one to see its effect on fare_amount.

1. passenger count-

We have plotted the histogram or we can say density plot of passenger count to see for each ride how many passengers are traveling. We can see frequency of single travelling passengers is very high and also below that is of double traveling passengers.

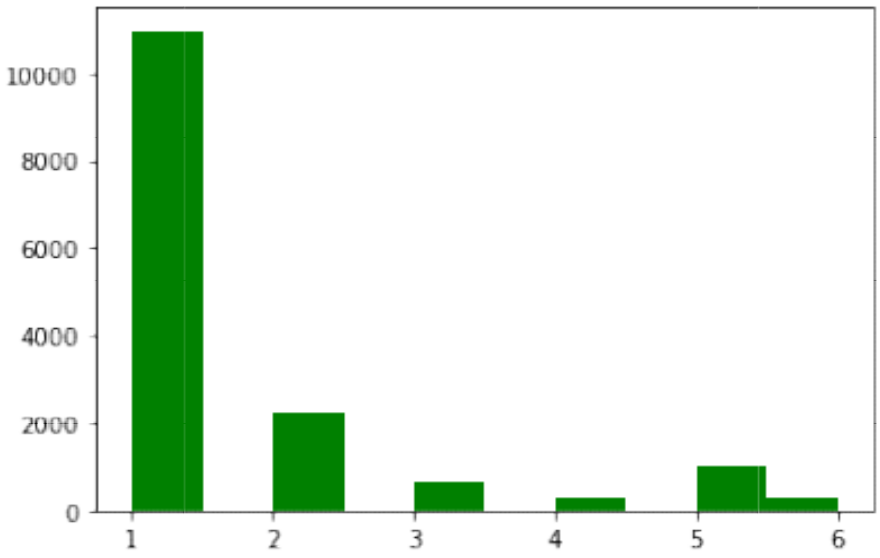


Fig 2 Histogram of passenger count

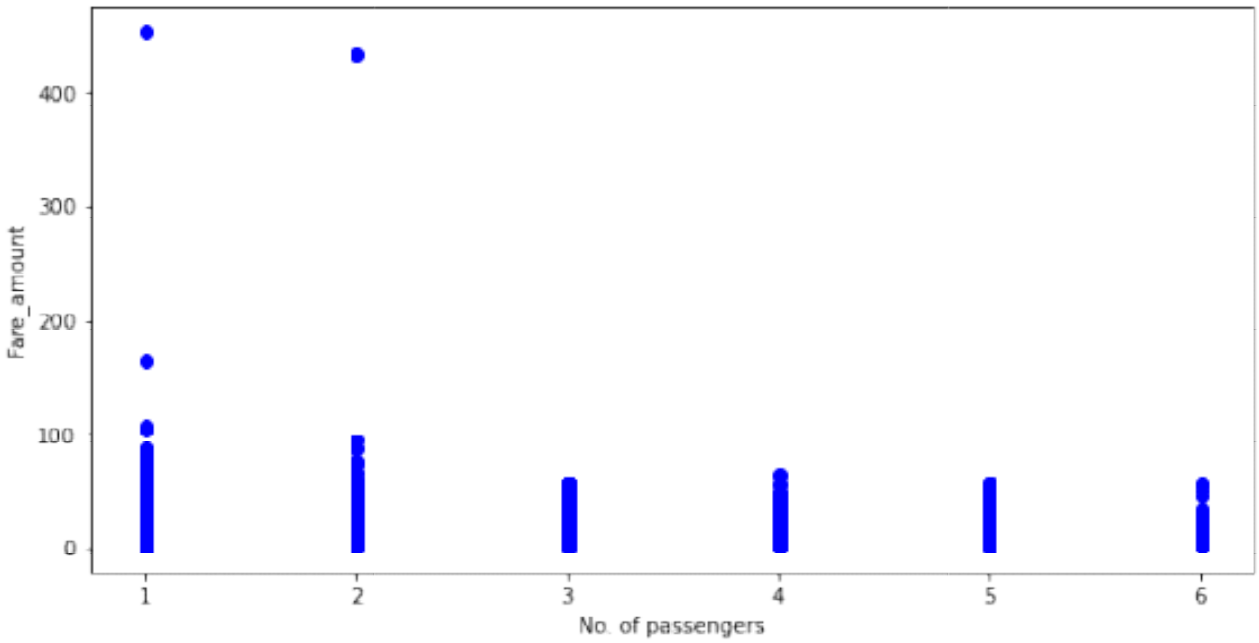


Fig 3 Relationship between passenger count and fare amount.

The scatter plot is plotted above to see the relationship between passenger count and fare amount. We can see that highest fare is coming form single and double travelling passengers.

2. Relationship between date and fare amount-

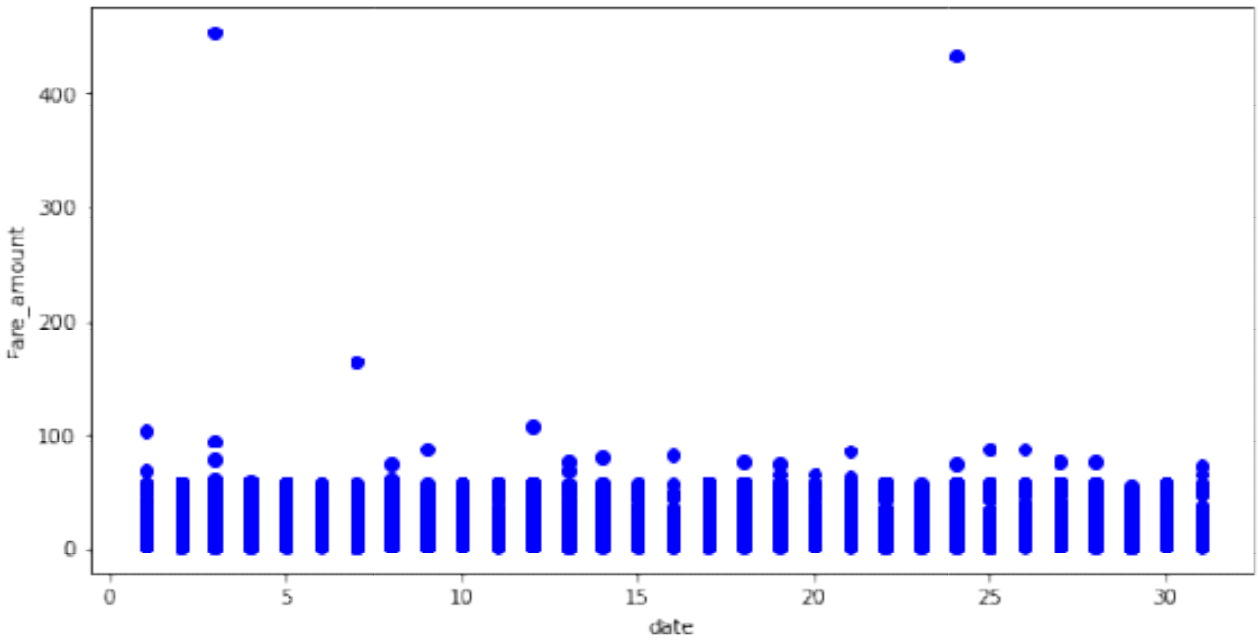


Fig 4 Relationship between date and fare amount.

We can see in the above scatter plot there is no much effect of 'date' on 'fare_amount' and we are also considering 'weekday' variable which is related to 'date' ,So we can drop date variable by observing its dependency with other variables in chi square test.

3. Relationship between hour and fare_amount:

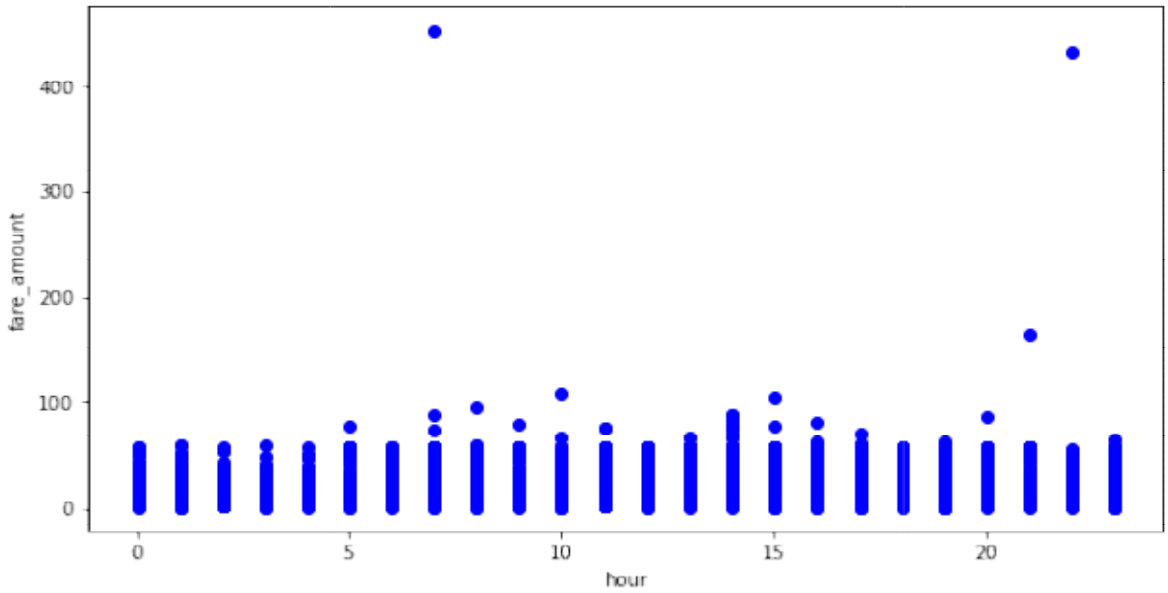


Fig 5 Relationship between hour and fare amount

From the above plot we can say that generally cab fare is higher after 8 pm. Also in the afternoon session cab fare is looking higher.

4. Number of cabs w.r.t. hour in day:

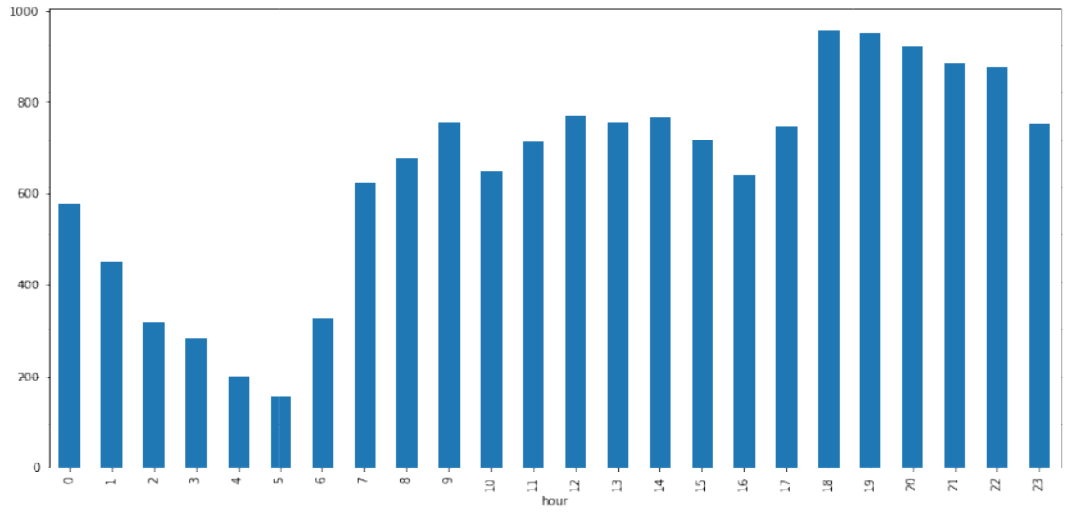


Fig 6 Number of cabs w.r.t. hour in day

We can see in above count plot of cabs w.r.t. hour in day. Highest no of cabs are from 7 pm to 12 pm. Also no of cabs are very less upto 5 am.

5. Number of cab rides w.r.t. weekday:

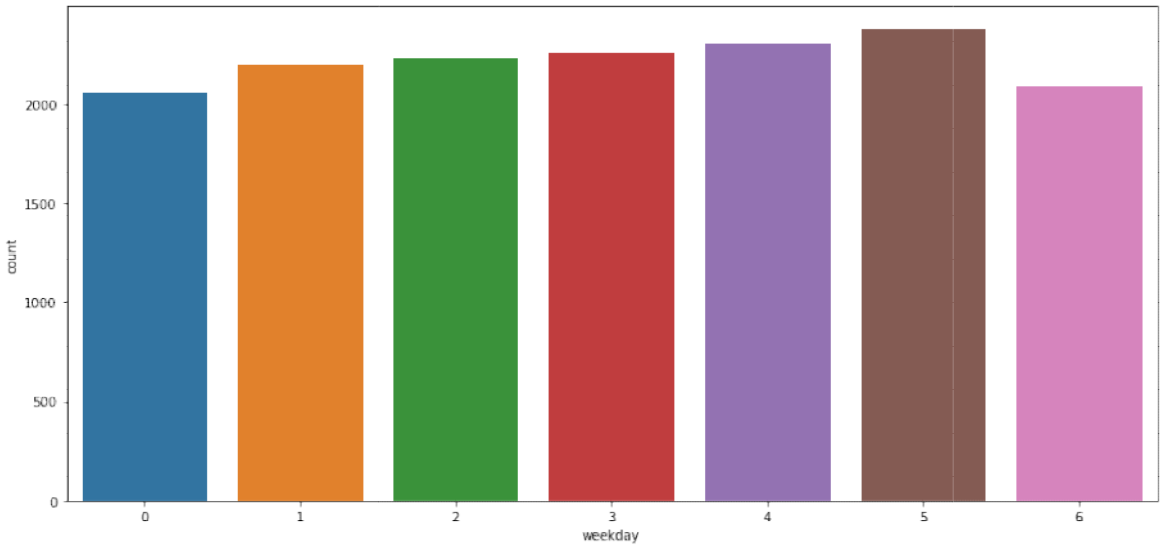


Fig 7 Number of cabs w.r.t. weekday

The above figure shows the count plot of cabs w.r.t. weekday variable. We can see that there is no much effect of weekday on number of cabs.

6. Relation between fare_amount and distance:

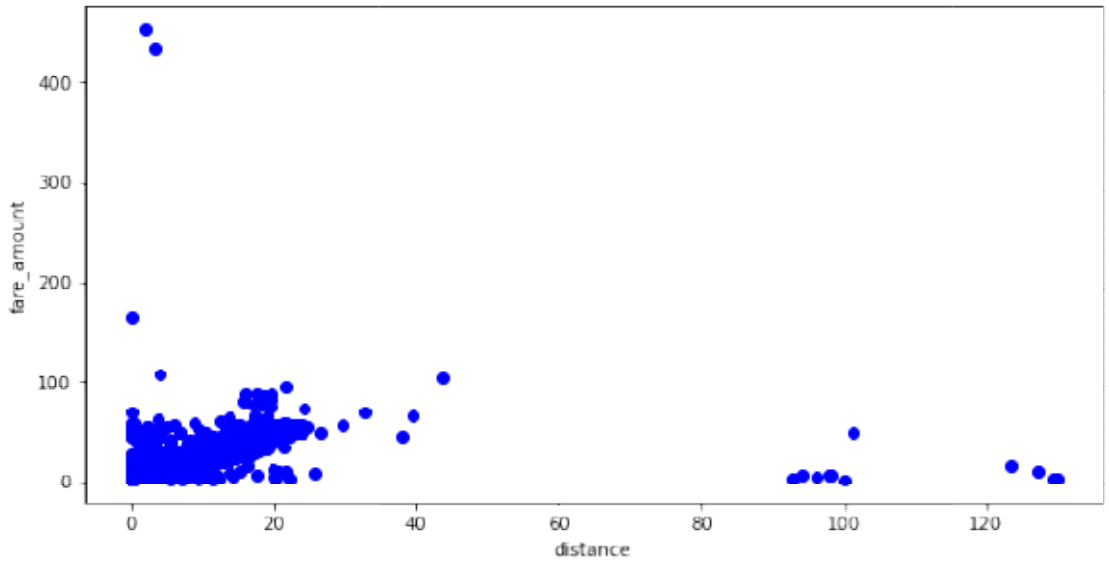


Fig 8 Realation between fare_amount and distance

The above scatter plot is between fare_amount and distance variable. We can see clearly linear relationship between these two variables. So distance is very much important factor as compared to other factors because we have not got this type of linear plot with other variables. There are some data points in the plot which are away from linear relationship but those are very few and we can ignore them.

So we have done the visualizations of the train_cab data with respect to fare_amount variable and found some important information of our data. No we will proceed further.

2.2.5 Feature selection:

This is the most important step in data preprocessing. In this step will we check the dependencies between the variables. So for getting the good results there should be always very high dependency between the predictor and target variable and very low dependency or we can say zero dependency between the one predictor variable to another predictor variable that is between the two independent variables. Because if two predictor variables are having high dependency it will just increase the data size and time to get the results and does not add any important value to get good results from the model. So there is no point in selecting both the variables for the further study so better is to drop one of the variable from them. So this will lead to good results and reduce the time required to get output.

This process is called as feature selection that means selecting only those variables which will be very strongly related to target variable and remove the predictor variables which are having high dependency between each other. This will also decreases the complexity of the model.

There are multiple tools and techniques present to do the feature selection. The main two are Wrapper and Filter methods. Filter method uses statistical method to check the dependency between the variables. For getting the correlation between the numeric variables (predictor and target both numeric) powerful tool used is correlation plot. One more tool used is multicollinearity analysis. To check the multicollinearity between the variables VIF is used which is Variance Inflation Factor. Its value should be below 5 for lesser multicollinearity. For the categorical variables we can either use Chi- square test or Analysis of variance test (ANOVA).

1) Correlation analysis:

This method is very powerful tool to check the correlation between numeric variables. It uses the pearson correlation plot to check the correlation between numeric variables. The variables may contain predictor variables or predictor with target variable.

In our train_cab data we have only two numeric variables those are fare_amount and distance variable. In data visualization we already seen that there is high correlation between these two variables is present. Now we will validate this again by plotting correlation plot for these two variables. So let's see the plot below,

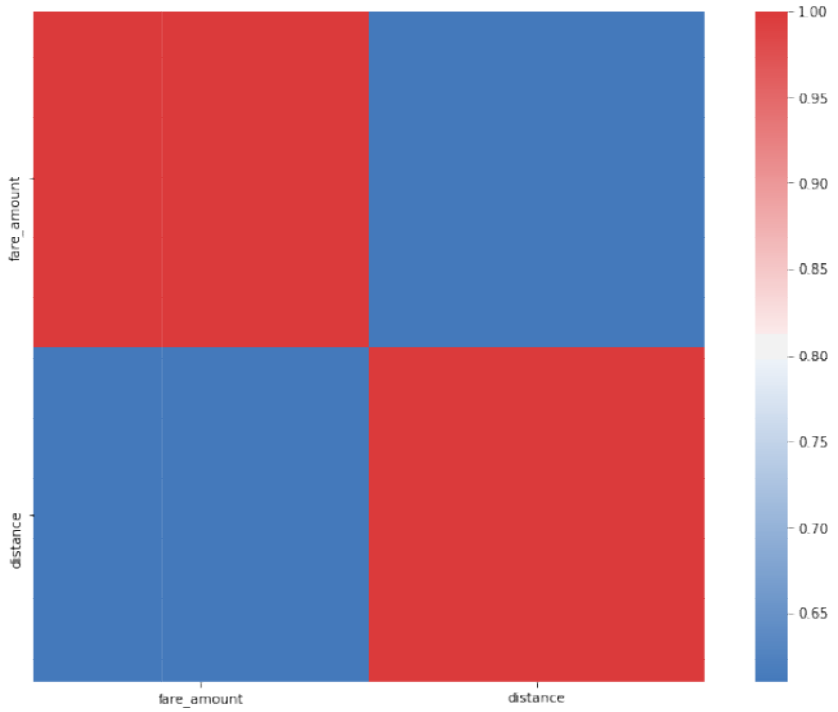


Fig 9 Correlation plot for numeric variables

Fig 9 shows the correlation plot that we have plotted in python. We have also plotted this in R also. The dark blue color indicates that there is very high positive correlation between two variables and dark red color indicates that there is very high negative correlation between two variables. So by analyzing the plot we can see that 'fare_amount' and 'distance' are very highly and positively correlated to each other.

2) ANOVA (Analysis of variance) test:

Now we have some categorical variables in our data. This test is performed to check whether the means of categories in any categorical variables are equal or not. It is decided based on P value which is a probability value that whatever we are assuming is correct up to 95 % confidence level and it is hypothesis based test. We will provide all the categorical variables with the target numeric variable to perform the ANOVA test. The hypothesis for this test is as follows,

Null Hypothesis- Means of categories present in one variable are equal

Alternate Hypothesis- Means of categories present in one variable are not equal

So if the p value which comes as a output of ANOVA test is less than 0.05 then we reject the null hypothesis saying that Means of categories present in particular categorical variable are not equal. That means we have selected correct categories of that variable.

If p value is more than 0.05 we say accept the null hypothesis saying that means are equal. That means the categories in any particular categorical variable are equal. So we need to either change the categories or drop that variable.

Now in our data set categorical variables are date, month, year, weekday, hour and passenger count. The target variable is fare_amount. So we performed ANOVA test on our data i.e. on predictor categorical variables w.r.t. to target variable. We have performed this test in our R code. The results of ANOVA test are as follows.

```
[1] "passenger_count"
               Df  Sum Sq Mean Sq F value   Pr(>F)
factor[, i]     5    2855   571.0   5.032 0.000131 ***
Residuals   15495 1758172   113.5
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
[1] "date"
               Df  Sum Sq Mean Sq F value Pr(>F)
factor[, i]    30    2816   93.85   0.826  0.736
Residuals   15470 1758211  113.65
[1] "month"
               Df  Sum Sq Mean Sq F value   Pr(>F)
factor[, i]    11    5457   496.1   4.377 1.38e-06 ***
Residuals   15489 1755570   113.3
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
[1] "year"
               Df  Sum Sq Mean Sq F value Pr(>F)
factor[, i]     6   19070    3178   28.27 <2e-16 ***
Residuals   15494 1741957     112
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
[1] "hour"
              Df  Sum Sq Mean Sq F value   Pr(>F)
factor[, i]    23    8905   387.2    3.42 5.52e-08 ***
Residuals   15477 1752122   113.2
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
[1] "weekday"
              Df  Sum Sq Mean Sq F value Pr(>F)
factor[, i]    6     974   162.3   1.429  0.199
Residuals   15494 1760053   113.6
```

So from the above results we can see the P-value for variables passenger count, month, year and hour is less than 0.05. So for these variables we will reject null hypothesis i.e. Means of categories present in those variables are not equal. So selected categories are well and we can select those variables for ML modles.

P-values for other variables i.e. date and weekday is greater than 0.05 that means - Means of categories present in those variable are equal there is no difference in categories within that variable and so we need to remove one of them variable or both.

3) Chi-square test for Independance:

We have already performed anova test for categorical variables. But still we will perform the another test to validate our results. Now we have another test to check the dependencies between the categorical variables which is Chi-square test.

It is used to determine the relationship between the two categorical variables. Each category from one variable is compared with all the categories of other variables to check the dependency. The hypothesis for this test are shown below,

Null hypothesis – categorical variables are not dependant (P<0.05-Reject this hypothesis)

Alternate hypothesis- categorical variables are dependant (If P >0.05-Accept this hypothesis)

So we can see the hypothesis for Chi-square test. If the p value for two categorical variables is less than 0.05 then we reject the null hypothesis saying that those two variables are dependent on each other and vice versa.

Now in our data set categorical variables are date, month, year, weekday, hour and passenger count. So we have passed these variables in Python code to do chi square test. For this test we have imported the function called as “chi2_contingency” from scipy.stats library. Finaly we have printed the output as two variables with p value less than 0.05 only.

```
# Lets perform chi square test of independance
from scipy.stats import chi2_contingency

factor_data=train_cab[['passenger_count', 'date', 'weekday', 'month', 'year',
'hour']]
for i in factor_data:
    for j in factor_data:
        if(i!=j):
            chi2,p,dof,
ex=chi2_contingency(pd.crosstab(train_cab[i],train_cab[j]))
            while(p<0.05):
                print(i,j,p)
                 break
```

Output-

```
passenger_count weekday 3.461935285604095e-15
passenger_count year 3.79019481336785e-30
passenger_count hour 4.598145050149016e-12
date weekday 4.693121478663736e-06
date month 4.829504146752254e-23
weekday passenger_count 3.461935285604095e-15
weekday date 4.693121478663466e-06
weekday hour 7.390964905870193e-138
month date 4.829504146752254e-23
month year 1.2770752375010934e-183
year passenger_count 3.790194813367905e-30
year month 1.2770752375010934e-183
hour passenger_count 4.598145050148754e-12
hour weekday 7.390964905871457e-138
```

We can see above the python code and its output for chi-square test for our train_cab data variables. We can see in the output we have printed only those variables whose p value is less than 0.05. We can see 'date' is dependant with 'weekday' and most of the variables. Also in data visualizations we observed that 'date' is not having much impact on 'fare_amount'. So we will remove 'date' variable from our train_cab data.

4) Multicollinearity:

For our Machine learning Regression models we always assume that there should be no relation between one predictor variable to other predictor variables. If this relation is present then our model will be a complex model and we will not get good results. This relation of predictor variables within each other is termed as Multicollinearity.

It is required to have correlation between predictor variable with the target variable. But if Multicollinearity is present in our variables that becomes problem. So we always need to check our

variables selected for ML models for Multicollinearity. If there is Multicollinearity present between two variables better is to drop one variable from them or both.

So to test the Multicollinearity we have used "VIF" for this project. VIF is nothing but variance inflation factor. Which is the measure of how much variance of coefficients in the linear regression model is increased due to Multicollinearity. The value of VIF should be between '1' to '5' Then we can say that there is no or very less Multicollinearity. If the value is above '5' then we say that there is very high Multicollinearity. Then we need to either remove one variable or both.

Now for our train_cab data we have calculated VIF for our variables in both R and python code. First we have created the data frame of predictor variables and passed it to the function "variance_inflation_factor" which is available in "statsmodels.stats.outliers_influence" library to calculate the VIF. The python code and the table of VIF values for our predictor variables is shown below

```
# lets create data frame of predictor variables
outcome, predictors = dmatrices('fare_amount ~
distance+passenger_count+date+weekday+month+year+hour',train_cab, return_type='dataframe')
# Lets calculate VIF for each independant variables form train_cab data
VIF = pd.DataFrame()
VIF["VIF"] = [variance_inflation_factor(predictors.values, i) for i in range(predictors.shape[1])]
VIF["Predictors"] = predictors.columns
VIF
```

Table 7 VIF values for predictor variables from train_cab data.

Predictors	VIF
Intercept	1.173824e+06
distance	1.001229e+00
passenger_count	1.002389e+00
date	1.001290e+00
weekday	1.010434e+00
month	1.014462e+00
year	1.014382e+00
hour	1.009262e+00

We can see in the table value of VIF is within the range that is between '1' to '5'. So we can say that there is no Multicollinearity between the predictor variables that we have selected from train_cab data.

Now after Performing the Feature Selection and performing different tests on the all the variables. We have found that “date” and “weekday” variables are correlated to each other from the results of ANOVA test and also from Chi-square test we have seen that. Also in the Visualizations we found that “date” variable is affecting much on “fare_amount”. So we will remove the “date” variable from our train_cab and test data. The remaining variables we will select for ML models and proceed further. The python code for removing the “date” variable from train_cab data is shown below.

```
# lets drop 'date' variable from train_cab and test data
train_cab= train_cab.drop('date', axis=1)
test_cab= test_cab.drop('date', axis=1)
```

2.2.6 Feature Scaling:

Feature scaling is the method of scaling the data i.e. Converting the data in all the numeric variables in same range. For Example suppose we have the two numeric data variables and all values of one data are far away from the all values of other data then that data is not feasible for machine learning model. The ML model will fail to predict the target variable and it will be more biased towards the variable which is having higher values. So there is a need to scale all the data variables in one common scale. For this we have two methods 1) Normalization 2) Standardization.

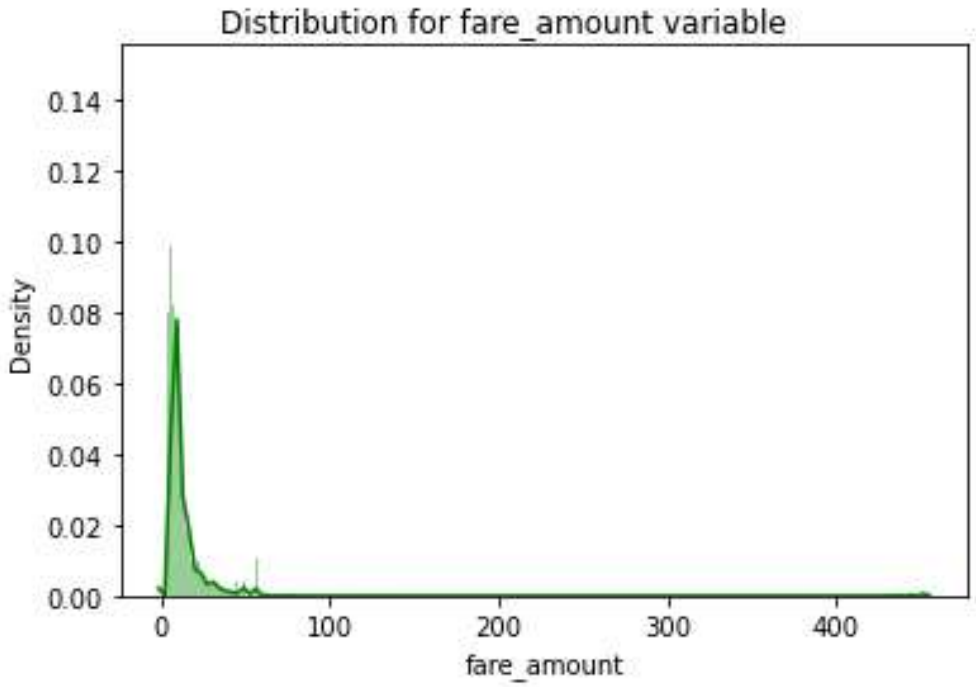
Normalization is used when the variable data in not normally distributed. It converts all the data in the range of “0” to “1”. After converting data to scale of 0 to 1 it is very easy to process that data in machine learning model and predict or classify our target variable more accurately.

Standardization is used when the data is normally distributed. It is also called as Z- score method this score ranges from “-1 to +1” so standardization converts the whole data in the range of “-1 to +1”, Z-score actually gives the information about how much that variable data is deviating from the mean of that same variable data. So positive Z-score means the value is above the mean and negative means value is below the mean.

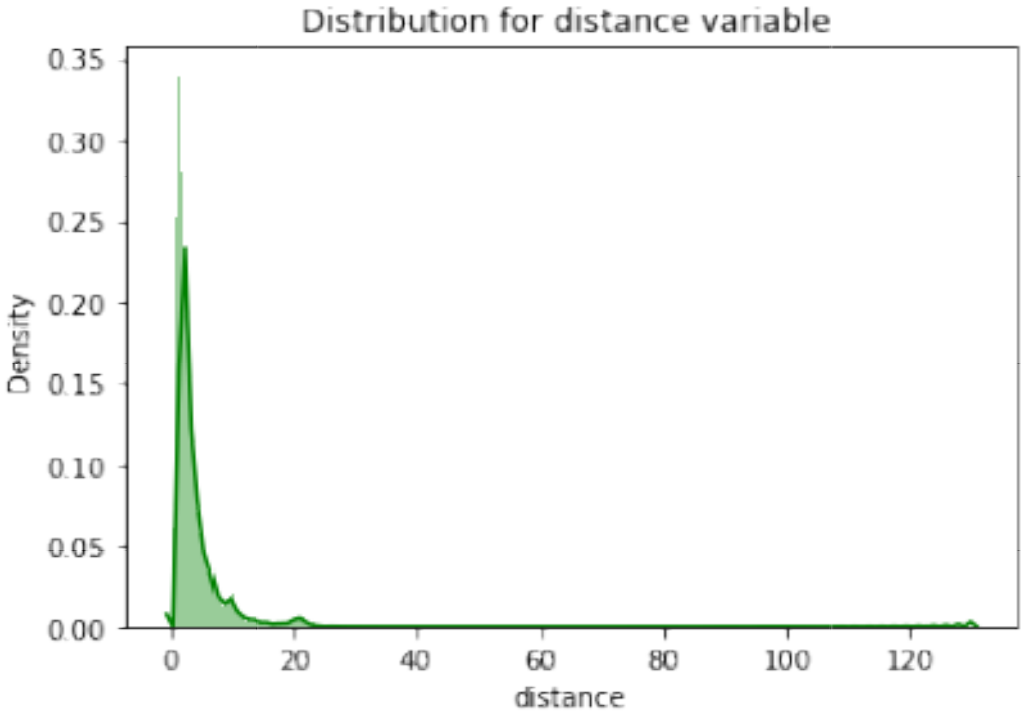
Now if we consider our data variables we have two numeric variables “fare_amount’ and “distance”. We have first plotted the histograms of these variables to see the distribution of data and we have found that the data in both the variables is left skewed and not normally distributed. So as we discussed above for this data we have to use Normalization method of scaling. But data is

highly left skewed it means data contains the outliers. So we know that when there are outliers we can't use Normalization method, because it is sensitive to outliers and not work well with this data. So to solve this issue we will use log transform of numeric variables instead of Normalization method. The log transform will convert the data not normal distributed data or near to normal distributed data. That is what we want by the method of scaling. The python code and histograms of fare_amount and distance variable before log transform are shown below.

```
# lets plot the histogram to see data distribution of fare_amount variable from train_cab data

sns.distplot(train_cab['fare_amount'],bins='auto',color='green')
plt.title("Distribution for distance variable "+i)
plt.ylabel("Density")
plt.show()

# lets plot histogram for distance variable from train data.
sns.distplot(train_cab['distance'],bins='auto',color='green')
plt.title("Distribution for distance variable "+i)
plt.ylabel("Density")
plt.show()

# lets plot the histogram to see data distribution distance variable from test_cab data
sns.distplot(test_cab['distance'],bins='auto',color='green')
plt.title("Distribution for distance variable "+i)
plt.ylabel("Density")
plt.show()
```



A



B

Fig 10 Histograms of fare_amount (A) and distance (B) variables before log transform

We can see the histograms in above figures. No we will apply the log transform to our numeric variables i.e. fare_amount, distance from train_cab and test data sets. The python code of log transform and histogram of fare_amount variable after log transform is shown below.

```
# lets apply log tranform on numeric variables from train and test data
train_cab['fare_amount'] = np.log1p(train_cab['fare_amount'])
train_cab['distance'] = np.log1p(train_cab['distance'])
test_cab['distance'] = np.log1p(test_cab['distance'])

# lets plot the histogram to see data distribution of fare_amount variable from
train_cab data after log transform
sns.distplot(train_cab['fare_amount'],bins='auto',color='green')
plt.title("Distribution for fare_amount variable after log transform ")
plt.ylabel("Density")
plt.show()
```

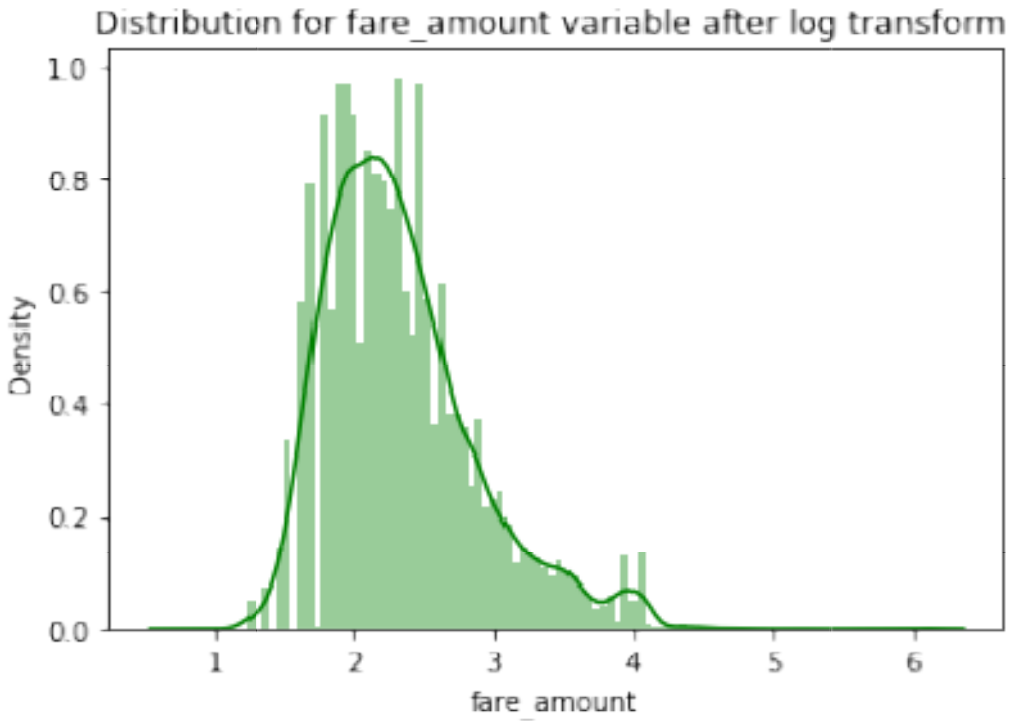


Fig 11 Histograms of fare_amount variable after log transform

We can see the now the data distribution of fare_amount variable which is closer to normal distribution. In the same way distribution of distance variable is also gone closer to normal

distribution. So now our data is clean does not contains missing values and also scaled. It is ready to fit on ML model. So we will proceed further. Next step is sampling.

2.2.7 Sampling:

Sampling is nothing but creating the subset of whole data with some conditions and methods. There are mainly three methods of sampling,

- **Simple Random Sampling:**

This is the most simple method of sampling. It can be used for both categorical and numeric variable. It is used mainly when the Target Variable is numeric. In this method of sampling the subset of data is created by use of rows. Just we need to pass the number of samples we want and randomly that number of samples are taken from original data.

- **Stratified Sampling:**

This method mostly used when the target variable is categorical. In this method we need to give the categorical variable as reference variable for creating samples. Strata's are created which will take some given percentage of data form original data randomly. We need to give the numbers in term of percentage of data to be taken from each category of reference categorical variable

- **Systematic Sampling:**

This sampling method very complicated as compare simple random sampling method. In this method the samples are taken by the index numbers of data points in the original data. Index number is calculated by K value. K value is the ratio of total observations to the no of sample observations we require. So at start any observation from row is taken then the next observation is taken by adding the k value in the index of previous observation. So this is not the random sampling method.

For our study we have used Simple Random Sampling method because our target variable is numeric variable. By using this sampling method we have divided the train_cab data into train data and test data. We divide this data because in machine learning we will train the data by using train data set and then apply that trained model to test data to predict the target variable. By this way we will analyze the performance of model by comparing the predicted values of target variable with the actual values.

So we have first split the train_cab data into two data sets first contain all predictor variables which is stored in ‘x’ and second contains our target variable which is stored in ‘y’ and then we applied sampling technique to split our train_cab data into two parts. So we have 80 % of the data in ‘x’ object and remaining 20 % data in ’y’ object. Again from ‘x’ and ‘y’ data we have separated the predictor and target variables and stored in ‘x_train’, y_train, ‘x_test’, ‘y_test’. The python code for this operation is shown below.

```
# TRAIN TEST SPLITTING OF DATA.
# first store all predictor variables in 'x' and target variable in 'y' from train_cab data
x=train_cab.drop(['fare_amount'],axis=1)  # predictors
y=train_cab['fare_amount']                # target variable
# lets split our train_cab data into train and test data
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.20,random_state=1)
```

CHAPTER 3

MODEL BUILDING AND EVALUATION

3.1 Model Selection:

This is the actual step of creating the Machine Learning models. For building the models we want to consider the type of target variable.

The model may be of two types based on the data type of target variable-

- **Classification model:**

Classification model means target variable is categorical and we build model for classifying the output in the categories of that variable. (e.g. response of bank customer to new scheme ‘yes’ or ‘no’)

- **Regression model:**

Regression model means target variable is numeric or continuous and we want to predict the continuous value of target variable for the sample input given in future cases by using the model.

So for our project as our target variable is numeric we will use regression models to predict the fare_amount. So we have built different regression models on our train data which are,

a) Multiple Linear Regression
b) Lasso Regression
c) Ridge Regression
d) Decision Tree Regression
e) Random Forest Regression
f) XGBoost Regression (Ensemble)

3.2 Defining the Error metrics for models:

Error metrics are nothing but the measures used to evaluate the performance of our ML models. It is called as error metrics because these metrics calculates the different type of ‘errors’ for our model based on the predictions made by our model on the test data set. As our target variable is numeric we will be building regression models on the data. So our main focus is model

should perform well on test data and then we can say that our model is good. There are different error metrics available for evaluating performance of regression models. We have used following error metrics for our project.

1. **MAPE- Mean absolute Percentage Error.**

   This is the error calculated by following formula,

   MAPE= (y-yhat)/y))*100

   Where, y- real values, yhat- predicted values.

   The output of above error is in percentage. Which is the indication of how much difference is there between the real and predicted values.

   This should be lower for better performance. We have used this error metric in both R and Python code.

2. **Accuracy:**

   Accuracy is calculated by subtracting the MAPE value from 100. Which will give us the percentage of accuracy with which real and predicted values predicted by our model. This should be obviously higher for good results. We have used this error metric in both R and Python code

3. **RMSE- Root Mean Squared Error**

   It is the standard deviation of prediction errors. It gives us the value that is measure of how far our data points from the regression line.
   It should be low and between 0.2 to 0.5 is good. We have used this error metric in both R and Python code

4. **RMSLE- Root Mean Squared Logarithmic Error**

   It is modified version of RMSE and uses log transform to get the better results. It is gives good results even if the data contains outliers. Also it gives penalty for the underestimation of target variable. If we plot the RMSE and RMSLE errors with the underestimated and overestimated predicted values we get the linear relationship by RMSE metric but RMSLE gives non linear relationship. So RMSLE is important metric and gives good results than RMSE. . We have used this error metric in Python code

5. **R-square**

It is a measure of how well the predictor variables are explaining the variance of target variable. This is very important error metric to evaluate the performance of model. Because it will give us the information about how correctly we have selected the predictor variables that will help us to predict the target variable more accurately. The formula is as shown below,

R-square= 1-(sum squared regression error/ sum squared total error)

The R-square value should be higher for better results. Generally when it is above 80 then we can say that model is good. We have used this error metric in both R and Python code

6. **Adjusted R-square**

It is modified version of R-square. If we add new variables to our model then even if those variables are not much affecting the target variable then also the score of R-square will increases. So to overcome this issue we use adjusted R-square which will increase the score only if new variable is significant to target otherwise it will decrease the score also so we will get highly accurate value.

Adjusted R-square is always less than R-square. We have used this error metric in our Python code

7. **MAE- Mean Absolute Error**

It is a arithmetic average of absolute errors (prediction error).

MAE= $\frac{1}{n}\sum_{i=1}^{n}|y - x|$

Where, n= no of observations, y= predicted value, x= real value.

MAE should be lower for better result. When we multiply this error by 100 we will get

MAPE. We have used this error metric in R code.

So by using above error metrics we are going to evaluate our regression models. We will be mostly focusing on R-square, Adjusted R-square value, RMSLE and MAPE. To calculate all these error metrics for all models we have defined the function for simplicity of programming. The function not only calculates the error metrics but also gives the predictions of target variable. The python code for this is shown below.

```
def rmsle(yt,yp):    # yt- y_train and yp- y_predicted
    log1 = np.nan_to_num(np.array([np.log(v + 1) for v in yt]))
    log2 = np.nan_to_num(np.array([np.log(v + 1) for v in yp]))
    calc = (log1 - log2) ** 2
    return np.sqrt(np.mean(calc))
# Function to calculate other error metrics
def error_metrics(yt, yp):
    print('r square  ', metrics.r2_score(yt,yp))
    print('Adjusted r square:{}'.format(1 - (1-metrics.r2_score(yt, yp))*(len(yt)-1)/(len(yt)-x_train.shape[1]-
1)))
    print('MAPE:{}'.format(np.mean(np.abs((yt - yp) / yt))*100))
    print('MSE:', metrics.mean_squared_error(yt, yp))
    print('RMSE:', np.sqrt(metrics.mean_squared_error(yt, yp)))

# Function to calculate and print the predictions of models for train and test data with its error metrics
def output_scores(model):
    print('###############   Error Metrics of Train data   ##################')
    print()
    # Applying the model on train data to predict target variable
    y_predicted = model.predict(x_train)
    error_metrics(y_train,y_predicted)
    print('RMSLE:',rmsle(y_train,y_predicted))
    print()
    print('###############   Error Metrics of Test data  ##################')
    print()
    # Applying the model on train data to predict target variable
    y_predicted = model.predict(x_test)
    error_metrics(y_test,y_predicted)
    print('RMSLE:',rmsle(y_test,y_predicted))
```

3.3 Model Building:

Now in this step actually we will build different regressions models on our data that we have listed above in model selection section.

3.3.1 Multiple Linear Regression (MLR):

The linear regression model is nothing but one of the statistical model. This model calculates the coefficients of the independent variables to express the target variable. It means the coefficients or weights of independent variables will be applied on test data to predict target variable. Coefficients will express how target variable is behaving w.r.t. independent variable. i.e. when independent variable is increasing target variable is also increasing or decreasing or not and coefficients will be the amount by which it is deviating. This model will give the line equation of fit between the target and predictor variables.

The MLR is extension of linear regression model. As we know that dependant variable is not dependant only on one independent variable but there are number of factors who are affecting the dependant variable. Linear regression is useful when there is only one dependant and independent variable. So when we need to predict the value of dependant variable based on more than one independent variable we use MLR. It gives us the coefficients for each predictor variable w.r.t. target variable so we will be able to predict the target more accurately than linear regression. There are certain assumptions made before building the MLR model. That is a) There should be linear relation between target and predictor variables b) There should be no correlation between predictor variables. Which is called as Multicollinearity that we have already checked for our predictor variables.

So we have built the MLR model on our train_cab data to predict fare_amount. The python code model summary and error metrics are shown below,

```
# lets build the model on train data
model = sm.OLS(y_train, x_train).fit()
# predict the test data
y_predict = model.predict(x_test)

# lets print model summary
print_model = model.summary()
print(print_model)

# error metrics on train and test data
output_scores(model)
```

Multiple Linear Regression model results summary-

OLS Regression Results

Dep. Variable:	fare_amount	R-squared (uncentered):	0.986
Model:	OLS	Adj. R-squared (uncentered):	0.986
Method:	Least Squares	F-statistic:	1.493e+05
Date:	Sun, 19 Jul 2020	Prob (F-statistic):	0.00
Time:	18:43:16	Log-Likelihood:	-1822.8
No. Observations:	12400	AIC:	3658.
Df Residuals:	12394	BIC:	3702.
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
passenger_count	0.0044	0.002	2.236	0.025	0.001	0.008
month	0.0036	0.001	4.897	0.000	0.002	0.005
year	0.0007	5.22e-06	126.763	0.000	0.001	0.001
weekday	-0.0023	0.001	-1.809	0.071	-0.005	0.000
hour	0.0007	0.000	1.685	0.092	-0.000	0.001
distance	0.7713	0.004	185.953	0.000	0.763	0.779

Omnibus:	6089.656	Durbin-Watson:	2.011
Prob(Omnibus):	0.000	Jarque-Bera (JB):	500296.600
Skew:	1.474	Prob(JB):	0.00
Kurtosis:	33.978	Cond. No.	3.31e+03

As we can see the summary of model we have build. We will see the results one by one,

- We have coefficients for all predictor variables. When it is positive it will lead to increase the target variable and vice versa for e.g. distance variable coef. value is +0.7713 here it means that when the distance will increase fare_amount will also increase by the 0.7713 times of distance variable.
- Then Std error also called as standard deviation error it measures the average amount that the coefficient of variable is deviating from the average of real value. It should be lower and we have low values for all the variables
- Then the 't' value' it gives the information about how many standard deviation of coefficients are away from zero. So that should be away from zero to have good correlation of target and predictor variable. So we can see in our model all variables t value is more than '1' except the weekday variable that means there is less correlation of weekday and fare_amount.

- Now the overall P value of model it gives us the information about whether the predictor variables are significant with the target or not. For our model it is less than 0.05 so our model is significant.
- F-statistic it means whether the predictor and target variable are dependent on each other or not. Its value should be more than 1. Our model is having 1.493e+05 which is good.
- The AIC and BIC for the model should be low to have a good fit that means how well the model is able to predict the future values. But for our model AIC is very high so we need to build other models to get good accuracy.

Error Metrics for MLR model-

```
###############   Error Metrics of Train data   #################

r square   0.7371977235658934
Adjusted r square:0.7370704893483024
MAPE:7.938759625438446
MSE: 0.07856170126387056
RMSE: 0.2802886035212109
RMSLE: 0.08103954699314646

###############   Error Metrics of Test data  #################

r square   0.7389383409702168
Adjusted r square:0.7384319168013909
MAPE:8.132574489529834
MSE: 0.07542823914129743
RMSE: 0.27464201998473836
RMSLE: 0.07972098107952268
```

We can see above the error metrics of MLR model. Adjusted R-square value is 73.7 % which is not good it should be above 80. MAPE is 8.13 that means accuracy of model is 91.87 % which is good all other error metrics are within the range and good but R-square should be above 80 % so we need to improve the value of R-square by building more regression models.

3.3.2 Lasso Regression Model:

This is another type of regression model which try to reduce the complexity in the model by trying to reduce the model to only those variables who actually affecting the target variable. The word Lasso means "least absolute shrinkage and selection operator". From the meaning itself we understand that Lasso regression model acts as a shrinkage and selection tool for our predictor variables. It acts as a good variable selection and regularization tool to improve the accuracy.

It actually creates the subsets of good predictors and we can use them to get the results by using less data than we have at start. It shrinks the data towards central point like 'mean' and produces model with less parameters. Lasso regression actually shrinks the coefficients of all predictor variables based on its importance to create the subsets. It also makes coefficients to zero also when it try to select the best subsets from the data. It will also give us coefficients like linear regression for predicting target variable but these coefficients are shrinked to reduce the complexity of model. That means while doing regularization it gives penalty to the coefficients so the coefficients value is reduced or some may became equal to zero so we get the simple model at final stage. The important parameter in lasso regression is 'λ'. When 'λ' increases it will try to reduce the coefficients that is starts shrinking and making coefficients to zero. So reducing the complexity if models. When λ is zero there will be no shrinkage and the model is like simple linear regression model.

So we have built the Lasso Regression model on our train_cab data. The python code and error metrics of model are shown below,

```
#Lets build the lasso model
lasso_model = Lasso(alpha=0.00021209508879201905, normalize=False, max_iter =
500)

#Lets fit the lasso model on train data
lasso_model.fit(x_train,y_train)

#Lets print the coefficients for predictors
coefficients = lasso_model.coef_
print('coefficients of predictors:{}'.format(coefficients))

# Error metrics of train and test data
output_scores(lasso_model)

Error metrics of Lasso Model
coefficients of predictors:[ 4.37314893e-03  6.05721793e-03  3.79560635e-02 -
2.34521946e-03 7.20653206e-04  7.68741255e-01]
###############   Error Metrics of Train data   #################

r square   0.7533102525687998
Adjusted r square:0.7531908191398813
MAPE:7.5507641730108865
```

```
MSE: 0.0737450470578741
RMSE: 0.27156039302128376
RMSLE: 0.07860234527996554

###############   Error Metrics of Test data  #################

r square   0.7610563332129225
Adjusted r square:0.7605928149456344
MAPE:7.664448277753233
MSE: 0.0690377135681115
RMSE: 0.26275028747484086
RMSLE: 0.0765601987654502
```

We can see above the error metrics of Lasso Regression model. The coefficients are printed with error metrics and we can compare it with the coefficients of MLR model their values are reduced. Adjusted R-square value is 76.05 % which is improved as compared to MLR model but it should be above 80. MAPE is 7.66 that mean accuracy of model is 92.34 % which is good all other error metrics are within the range and good but still the R-square value is not above 80 % so we need to improve the value of R-square by building more regression models.

3.3.3 Ridge Regression Model:

The main problem in linear regression is it does not differentiate the predictor variables that are important, more important or less important. So it will treat all variables in equal range so it will lead to over fitting problem and it will also affect on R-square value. So to solve all these issues of simple linear regression models we use ridge regression models. Which will classify variables as important and less important by applying the penalties to over and under estimators.

So Ridge regression is similar to lasso regression it also do shrinking and regularization but the difference is it does not make coefficients to zero line lasso regression. This model also gives penalty to coefficients based on underestimation and overestimation but it will give penalties in terms of square of the magnitude of coefficients so here all coefficients are reduced or shrink by same range so they don't becomes zero like lasso regression. The shrinkage in this is model is nothing but it creates new ridge coefficients those are closer to true parameters of the data. These coefficients work well even when there is multicollinearity in the variables is present. This model also has λ as a penalty parameter which increases from 0 to infinity to reduce the model complexity. The optimum value of λ should lie between 0 to infinity and should not equal to infinity.

So we have built the Ridge Regression model on our train_cab data. The python code and error metrics of model are shown below,

```
# Lets build Ridge regression model
ridge_reg = Ridge(alpha=0.0005,max_iter = 500)

# Apply model on train data
ridge_reg.fit(x_train,y_train)

# print the coefficients of predictors
ridge_reg_coef = ridge_reg.coef_
print('coefficients of predictors:{}'.format(ridge_reg_coef))

# Error metrics of train and test data
output_scores(ridge_reg)
```

Error Metrics from Ridge Regression model

```
coefficients of predictors:[ 4.50606804e-03  6.07563605e-03  3.80181692e-02 -
2.40628812e-03  7.25761083e-04  7.69316362e-01]

###############   Error Metrics of Train data   #################
r square   0.7533108583201198
Adjusted r square:0.7531914251844724
MAPE:7.548537033354696
MSE: 0.07374486597552242
RMSE: 0.2715600596102498
RMSLE: 0.07860706581805475

###############   Error Metrics of Test data  #################
r square   0.761056341522373
Adjusted r square:0.760592823271204
MAPE:7.662546058483537
MSE: 0.06903771116727174
RMSE: 0.2627502829061688
RMSLE: 0.07656311610978069
```

We can see above the error metrics of Ridge Regression model. We can compare the coefficients with Lasso coefficients the values are increased to some extent. Adjusted R-square value and MAPE are almost same as lasso regression. So we can day that both the models are very

much similar to each other. But still the R-square value is not as per our requirement, so we need to try more models on our data.

3.3.4 Decision Tree Regression Model:

This is another type of regression model. But this model does not create the coefficients like linear regression model to predict the target variable. It can be applied to both classification and regression.

It creates the trees for the data. Which is having the parent node at top and each node under it denotes the class and leaf under it denotes the attributes. It is *rule based method* which creates some rules to predict the target variable. Each branch connects node with "and" operator & multiple branches are connected by "or" operator. For regression model it uses the central tendencies to derive the result of model.

The decision tree regression model can be build by using the statistical measures. We have first built the model on train data which we created by sampling by using the method of ANOVA and stored in 'fit_dt' object these are nothing but the rules of model which we have created. So all this rules we have applied on test data to predict the target variable values. The python code and error metrics of this model are shown below,

```
# lets build the model and apply it on train data
fit_dt=DecisionTreeRegressor(max_depth= 2).fit(x_train, y_train)
print(fit_dt)
# lets print the relative importance score for predictors
tree_features = fit_dt.feature_importances_
print('feature importance score of predictors:{}'.format(tree_features))
# Error metrics of train and test data
output_scores(fit_dt)

Error metrics for Decision Tree model
feature importance score of predictors:[0. 0. 0. 0. 0. 1.]
###############   Error Metrics of Train data   #################
r square   0.7040943722593598
Adjusted r square:0.7039511112437506
MAPE:9.338203767342213
MSE: 0.08845756530075963
RMSE: 0.2974181657208578
RMSLE: 0.08630190300233778
```

```
###############   Error Metrics of Test data  #################
r square   0.7045653518147885
Adjusted r square:0.7039922487145263
MAPE:9.703295909667416
MSE: 0.08535958660784013
RMSE: 0.292163629851219
RMSLE: 0.08618877032287285
```

We can see above the error metrics of Decision Tree model. We have printed the feature importance scores to see the important features form our predictor variables and it is showing that 'distance' variable is very important w.r.t. our target variable. But if we see the error metrics of this model adjusted R-square is not so good which is less than the previous regression models that we have built. MAPE value is also increased and all the error metrics are poor than our previous models. So we need to look for another regression model for improving results.

3.3.5 Random Forest Regression Model:

This model is improved version of Decision Tree model. It can be applied to both classification and regression estimation. It combines many decision trees to get the good result. It works on the principle of bagging and random selection of features. Bagging means error of one tree is feed to another tree to improve the accuracy and random feature selection means for every tree different features and data are used. It also provides the important variables from the predictor variables. It gives the output in terms of mean. The optimum number of trees should be in the range of 64-128 to get the good results.

So we have built the Random Forest Regression model on our train_cab data with the default parameters of model. The python code and error metrics are shown below,

```
# Lets build the Random forest model
rf_model = RandomForestRegressor(n_estimators = 70, random_state=0)
# Aply model on train data
rf_model.fit(x_train, y_train)

# lets print the relative importance score for predictors
Forest_features = rf_model.feature_importances_
print('feature importance score of predictors:{}'.format(Forest_features))
```

```
# Error metrics of train and test data
output_scores(rf_model)
```

Error metrics of Random Forest model

```
feature importance score of predictors:[0.01109391 0.02965522 0.03297079
0.02609098 0.0437125  0.8564766 ]

###############   Error Metrics of Train data   #################
r square   0.9689804737158708
Adjusted r square:0.9689654557898073
MAPE:2.83950387461845
MSE: 0.009272928645622137
RMSE: 0.09629604688470933
RMSLE: 0.02839154830486228

###############   Error Metrics of Test data  #################
r square   0.7904446544508457
Adjusted r square:0.7900381455361043
MAPE:7.666904419634807
MSE: 0.0605465803602189
RMSE: 0.24606214735350682
RMSLE: 0.0738504561602566
```

We can see above the error metrics of Random Forest model. We have printed the feature importance scores for this model also w.r.t our target variable. We can see the result is somewhat different from Decision Tree model. It has given some scores to all predictors and given maximum score to 'distance' variable. The Adjusted R square value is good from all the models we have built until now also the MAPE and other error metrics also good. We can say that RF model is good model. But still the R- square is below 80 so we need to improve it.

We have applied all popular regression algorithms to our train_cab data but still we have not got the value of R-square above 80. Even we have applied the advanced level algorithm that is Random Forest Regression model that has given good results but still not up to the mark. So we need to apply more advanced algorithm and techniques to improve our results. Which is an ensemble learning algorithms.

3.3.6 Ensemble Learning Algorithms:

This is most advanced algorithm techniques that are used in machine learning now days. It can be used for both Classification and Regression models. Ensemble learning is nothing but using number of base learners or weak learners to predict or classify our target variable. So these can be done by two ways,

- By using the number of popular models (e.g. Decision Tree, Navie Bayes, KNN, etc) as weak learners for predicting or classifying the target variable on the same train data.
- By using any one machine learning model as a weak learner many times on the train data by changing the same data for every model.

So by using one of the techniques listed above numbers of models are created and finally we combine the results of all models to create one super model or final model which is having high accuracy as compared to all previous models. This method is called as ensemble learning. There are again two methods in ensemble learning which are Bagging and Boosting. So we will see these one by one in detail.

I. Bagging-

It is also called as bootstrap aggregation. In this method the train data is divided in number of subsets suppose we have train data as 'd' then in bagging method the subsets of this data are created by using random sampling method with replacement as d1, d2, d3…..dn after this classification or regression models called as weak learners are created on these subset data sets. Finally the ensemble classifier or regressor model will combine the outputs of all weak learners and gives his final output. So that ensemble model will be our final model with improved accuracy.

II. Boosting-

This method is somewhat different than bagging and it is sequential learning mehod. We know that we train our models based on train data. So we have train data with n observations in it now for boosting method equal weights are given to each observation in data set and after that the first subset of data set is created by using random sampling. Now on that subset of data we build our first weak learner classification or regression model and use that model to predict the target variable in train data for all observations. Now in next step we will increase the weight of the observations wrongly predicted by our first model from train data so that they can be selected in next subset of data for training second weak

learner model. Now again we use this second model to predict the train data target variable. So this process is repeated to build many models until we get good accuracy. Finally ensemble model will use these weak learners to predict on test data and combines their results to given final output for test data.

So from the above two methods of ensemble learning we have used boosting method for our project to predict the frae_amount. Boosting method is more accurate than bagging because it considers error of model one to train the second model and this way it increases the accuracy.

3.3.7 XGBoost Regression Model:

It is also called as Extreme Gradient Boosting model. It is a decision tree based ensemble machine learning algorithm. This uses gradient boosting framework. It can be used for both classification and regression purpose. This model can handle the missing values automatically that means we don't need to do find and impute missing values in data. Also it prevents overfitting problem. It can do data scaling itself and find optimal number of iterations to be done for better results. As it uses the decision trees so it automatically optimizes the number of tress that should be used also the depth of tress. It takes the error of one tree and give it as a input to next tree so that error can be minimized and finally we will get the optimized and improved result.

So we have used this model to improve accuracy of our predictions. We have built this model on our train_cab data. The python code and error metrics of this model are shown below.

```
# Lets build XGboost model and apply on train data
xgb_model = GradientBoostingRegressor(n_estimators= 70, max_depth= 2)

xgb_model.fit(x_train, y_train)

# Error metrics of train and test data
output_scores(xgb_model)
```

Error metrics of XGBoost model

```
###############   Error Metrics of Train data   #################

r square   0.8043130333591385
Adjusted r square:0.8042182926345484
MAPE:7.194576348545246
MSE: 0.058498355581508905
RMSE: 0.2418643330082154
RMSLE: 0.07095212873004349

###############   Error Metrics of Test data  #################

r square   0.8068060920211688
Adjusted r square:0.8064313220735861
MAPE:7.417837944190711
```

```
MSE: 0.05581928938100637
RMSE: 0.2362610619230481
RMSLE: 0.07104722076672282
```

We can see above the error metrics of XGBoost model. R-square and adjusted R-square values are above 80%. MAPE is 7.19 % which is lowest among all the models we have built. So the accuracy and other error metrics are also better than all models. So we have got very good results by this model. We know that RF model also got closer results to this model. But still we will check that can we able to improve our results. So that can be possible by hyper parameter tuning method. So we will apply this method on RF model and XGBoost model.

3.3.8 Hyper Parameter Tuning:

This is the method to improve the performance of the model by selecting best parameters for each model we have taken for study. So we know that any model we are going to build have some default parameters set to it and by using that default parameters model will be created but sometimes those default parameters are not optimum parameters to get good result from that model. So we can improve the results by selecting the optimum parameters for that model and use those parameters to build the model which will give us the highest accuracy from that model. This method of finding and selecting the best parameters is called as hyper parameter tuning.

So for this we first need to know the different parameters that are present in each model. After getting the parameters we create the random grid of those parameters. This is nothing but the dictionary we can say, in that dictionary we take all the parameters with their possible ranges. Now the next step is actually to take different values of each parameter and build the model for that parameters and see the result. But if we try to do this process manually it is very tedious task and will requires huge time. So this is not possible to do it manually. So doing all this process we have two methods or function available and those are GridSearchCv and RandomizedSearchCv.

- GridSearchCv-

  This function will be doing the selection of parameters from the random grid and provide us the best parameters. The output of this function is actually the combination of parameters with their different values. That means this method will provide us different combinations of parameters with best their values by changing the values of each parameter at a time and keeping the values of other parameters constant at that time. Because of this this method is very time consuming and it requires lots of time to give output.

- RandomizedSearchCv-

  This method is also used for selecting the best parameters for the model the only difference in this method is that selection of parameters and their values is done randomly and the optimum combination of parameters with their values is given as a output. This method is faster than GridSearchCv and can be useful for huge amount of data is present.

The methods above not only give best parameters but also do ***K fold cross validation*** and provide the best score for that model.

❖ K fold cross validation-

It is a technique to generate and get the best possible score from the model. In this method actually the train data is divided in number of equal subsets. The number of subsets is called as folds and it is denoted by 'k'. So need to give how many subsets of data are to be created. The standard or more common value is '5' which is the value of 'k'. now after creating subsets of data for e.g. we will take k=5. Then from the 5 subsets 4 subsets are taken as train data sets and first model is trained on that and 5^{th} data set is kept as a test data set. So the prediction is done by first model on that test data set and the score is recorded. Now the process is repeated and we will get the 5 scores because we have taken k=5. Finally the average of those 5 scores is calculated so that average value will be the best value for that model.

So by considering above points we have done hyper parameter tuning of our RF model and XGBoost model that we have built already. We have used RandomizedSearchCv method of tuning for our models. So we will see one by one the tuned models and their best score values.

3.3.9 Hyper Parameter Tuning of Random Forest Model:

First of all let's see the default parameters that our RF model used previously for building the model. `Random Forest current parameters`

```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'criterion': 'mse',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': 42,
 'verbose': 0,
 'warm_start': False}
```

We can see above default parameters from these parameters we have created random grid of parameters. Then e have built the RF model again and applied this random grid with RandomizedSearchCv to that model. The output from then model we got is best score of model and best parameters. The python code and output of model is shown below.

```
##Lets create Random hyper parameter grid and apply Random Search CV on Random
Forest Model

# lets build model again as RRF
RRF = RandomForestRegressor(random_state = 0)

# Create the random grid
random_grid_RRF = {'n_estimators': range(80,120,10), 'max_depth':
range(4,12,2)}

# appply Random Search CV on model with cross validation score 5
RRF_cv = RandomizedSearchCV(Forest, random_grid_RRF, cv=5)

# lets apply model on train data
RRF_cv.fit(x_train, y_train)
# Print the tuned parameters and score
print("Tuned Random Forest Parameters: {}".format(RRF_cv.best_params_))
print("Best score {}".format(RRF_cv.best_score_))
```

Output of model-

```
Tuned Random Forest Parameters: {'n_estimators': 110, 'max_depth': 8}
Best score 0.7934025911021153
```

We can see the output of model the best score by cross validation method that we got is 0.7934. Now we will do tuning of XGboost model and compare the RF model score with XGBoost score

3.3.10 Hyper Parameter Tuning of Xgboost Model:

First of all let's see the default parameters that our XGBoost model used previously for building the model.

```
XGBoost currrent Parameters
{'alpha': 0.9,
 'ccp_alpha': 0.0,
```

```
 'criterion': 'friedman_mse',
 'init': None,
 'learning_rate': 0.1,
 'loss': 'ls',
 'max_depth': 3,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_iter_no_change': None,
 'presort': 'deprecated',
 'random_state': 42,
 'subsample': 1.0,
 'tol': 0.0001,
 'validation_fraction': 0.1,
 'verbose': 0,
 'warm_start': False}
```

We can see above default parameters for XGBoost model from these parameters we have created random grid of parameters for XGBoost model. Then we have built the XGBoost model again and applied this random grid with RandomizedSearchCv to that model. The output from then model we got is best score of model and best parameters. The python code and output of model is shown below.

```
##Lets create Random hyperparameter grid and apply Random Search CV on XGBoost
model
# lets build model again as XBG
XGB = GradientBoostingRegressor(random_state = 0)

# Create the random grid
random_grid_XGB = {'n_estimators': range(90,120,10),'max_depth': range(1,10,1)}



# Apply Random Search CV on model with cross validation score 5
XGB_cv = RandomizedSearchCV(XGB, random_grid_XGB, cv=5)
```

```
# lets apply model on train data
XGB_cv.fit(x_train, y_train)


# Print the tuned parameters and score
print("Tuned XGBoost Parameters: {}".format(XGB_cv.best_params_))
print("Best score {}".format(XGB_cv.best_score_))
```

Output of model-

```
Tuned XGBoost Parameters: {'n_estimators': 90, 'max_depth': 3}
Best score 0.8000230840834843
```

So we can see the output of XGboost tuned model. It is giving the best parameters for model and the best score of this model is 0.80 which is greater than our tuned RF model score. So now we will build the XGBoost model again with these tuned parameters.

3.3.11 XGBoost Tuned Regression Model:

So we will now build the model using best parameters that we got above on our train_cab data and we will use this model for predicting fare_amount for given test data. The python code and error metrics for our final model is shown below.

```
# lets apply the tunned parameters and build our final XGBoost model on
train_cab data.
XGB_Final = GradientBoostingRegressor( n_estimators= 110, max_depth= 3)

# Apply the model on train data.
XGB_Final.fit(x_train,y_train)

# lets create and print the important features
XGB_Final_Features = XGB_Final.feature_importances_
print(XGB_Final_Features)

# Sorting important features in descending order
indices = np.argsort(XGB_Final_Features)[::1]
# Rearrange feature names so they match the sorted feature importances
Sorted_names = [test_cab.columns[i] for i in indices]
# create and set plot size
```

```
fig = plt.figure(figsize=(20,10))
plt.title("Feature Importance")

# Add horizontal bars
plt.barh(range(pd.DataFrame(x_train).shape[1]),XGB_Final_Features[indices],alig
n = 'center')
plt.yticks(range(pd.DataFrame(x_train).shape[1]), Sorted_names)
plt.savefig('Final XGBoost Model Important Features plot')
plt.show()

Important Parameters score
[4.90261652e-04 2.18917092e-03 2.35962514e-02 3.25181310e-03
 8.94719636e-03 9.61525307e-01]
```

So we can see the important features score above which is more accurate than we have got it from our previous models. We have also plotted the bar graph of these features which is shown in fig 12. We can see in that plot again the 'distance' variable is very important for predicting fare_amount, below that 'year' and 'hour' variables are also important as compared to other variables except 'distance' variable

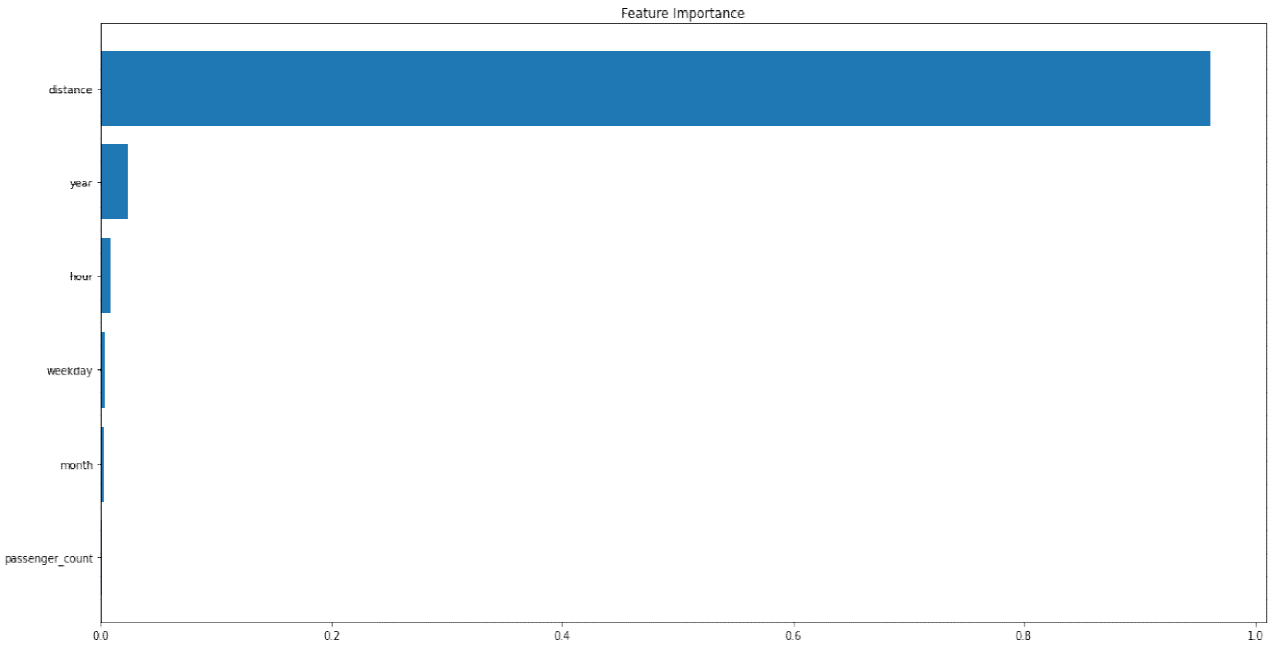


Fig 12 Important Features plot of Final XGboost model.

Now let's see the error metrics of XGBoost tuned model.

```
###############   Error Metrics of Train data   #################

r square   0.8300662373894389
Adjusted r square:0.8299839649311428
MAPE:6.78013203465076
MSE: 0.05079973306929771
RMSE: 0.22538796123417443
RMSLE: 0.06624089869160966

###############   Error Metrics of Test data  #################
r square   0.8111312051763754
Adjusted r square:0.8107648253610046
MAPE:7.16407785541231
MSE: 0.0545696395067333
RMSE: 0.2336014544191309
RMSLE: 0.07007535090944972
```

We can see the values of R-square and Adjusted R-square is 81.11 % and 81.07 % respectively. This is above 80 % that is what we want from our ML models so we have got this value by using techniques such as tuning of models and using cross validation technique.

So in next section we will do model evaluation based on error metrics that we have got for each models.

CHAPTER 4 -VISUALIZATIONS FROM 'R'

In this section we have added the visualizations that we created in our R code.

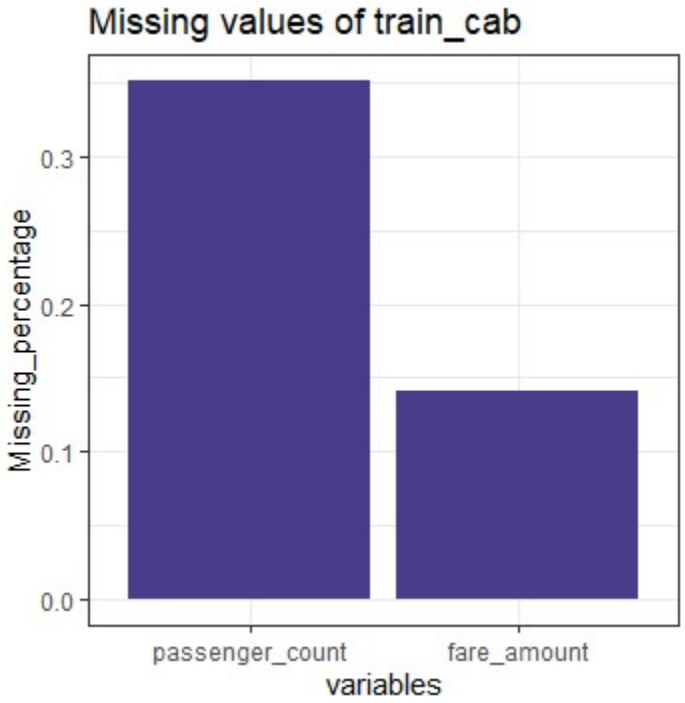


Fig 13 Missing values in train_cab data.

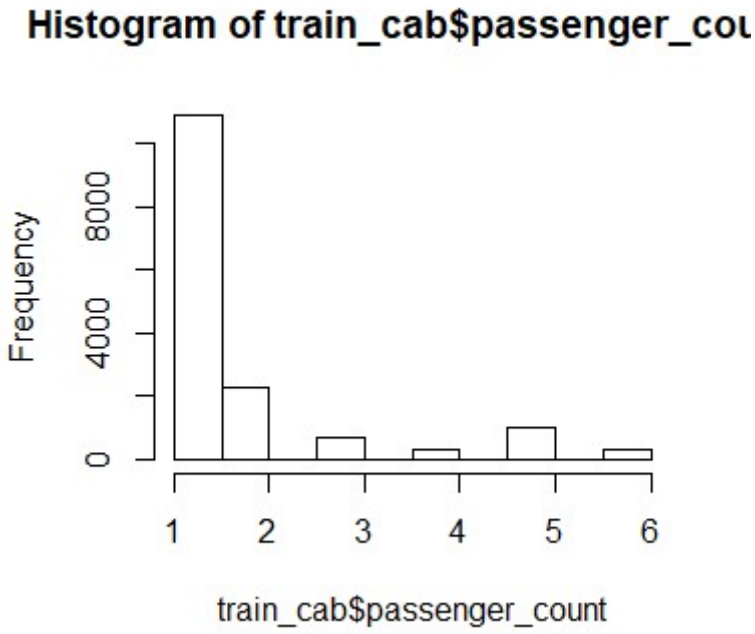


Fig 14 Histogram of passenger count variable from train_cab data.

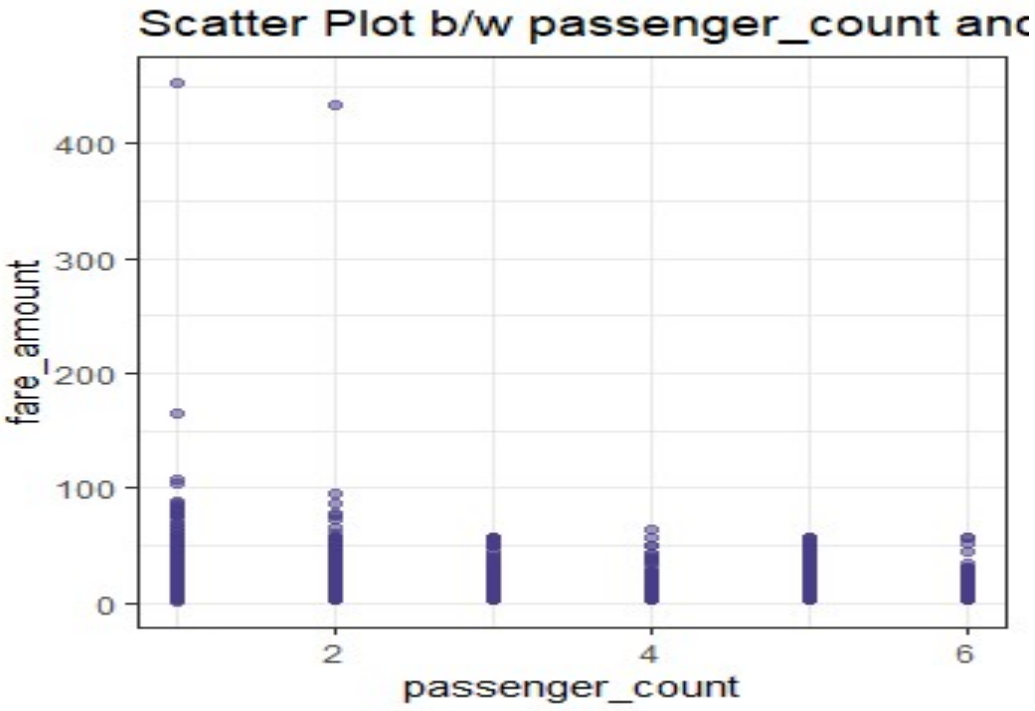


Fig 15 scatter plot of passenger_count and fare_amount

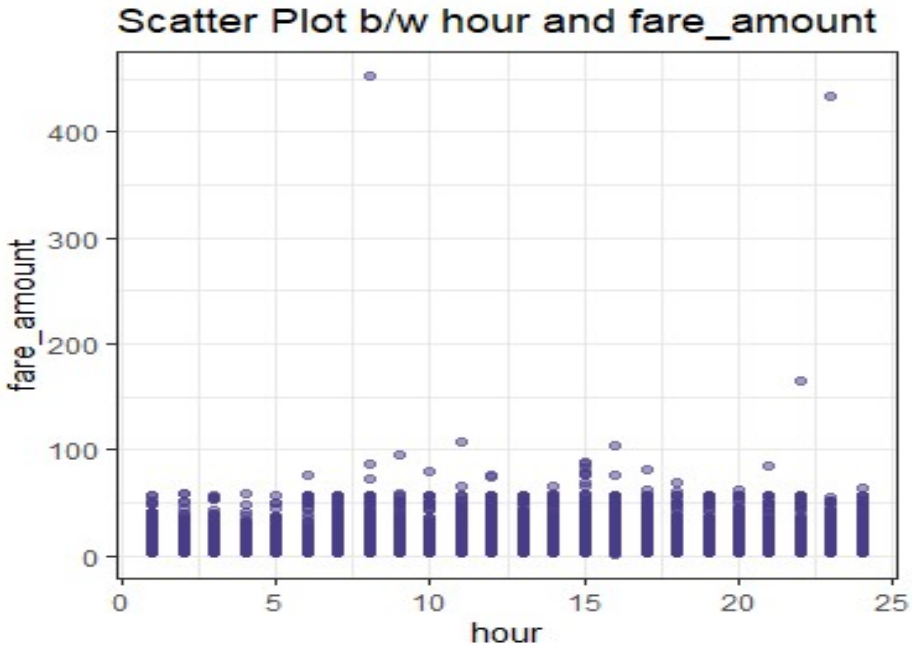


Fig 16 scatter plot of hour variable and fare_amount

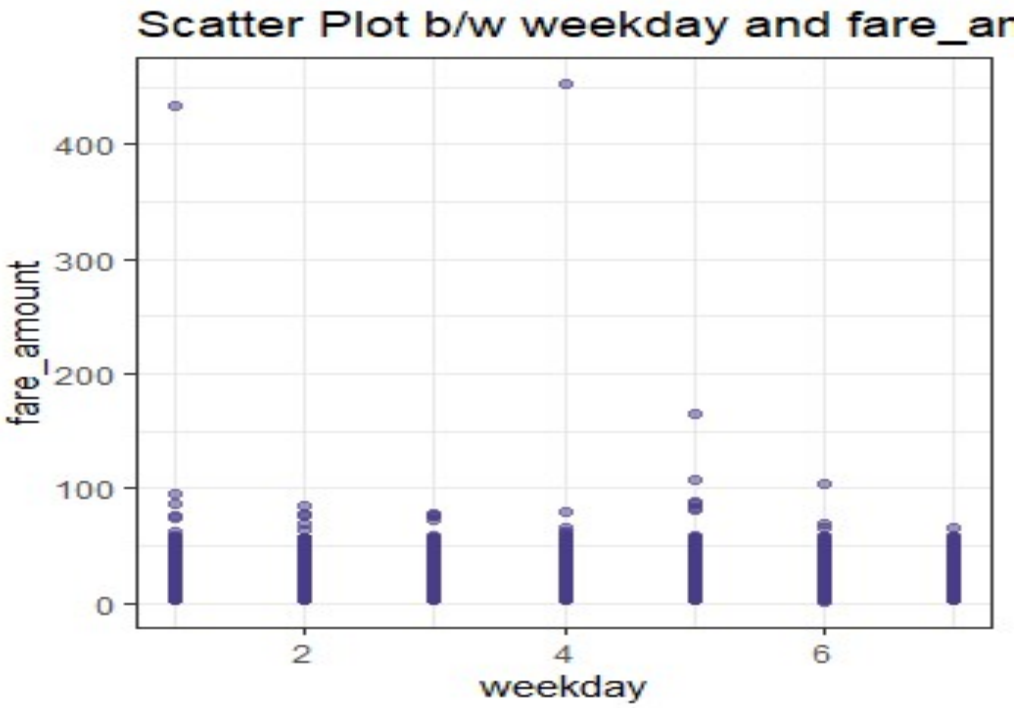


Fig 17 scatter plot of weekday and fare_amount

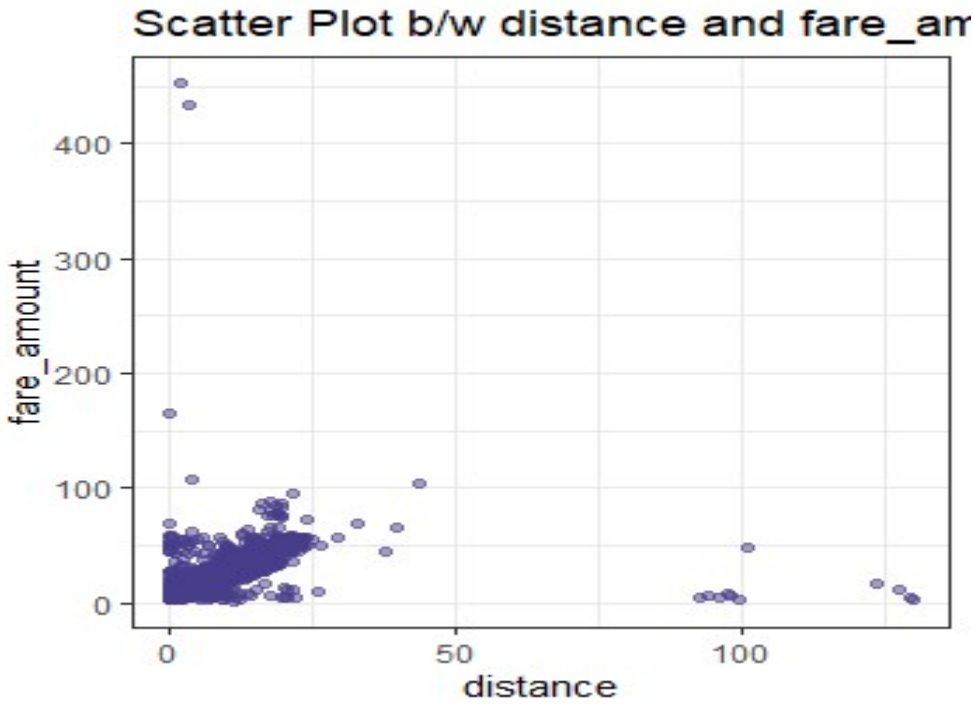


Fig 17 scatter plot of distance and fare_amount

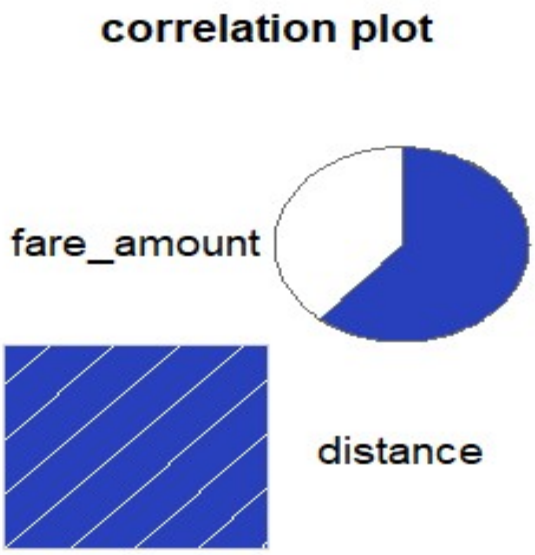


Fig 18 Correlation Plot between Numeric Variables

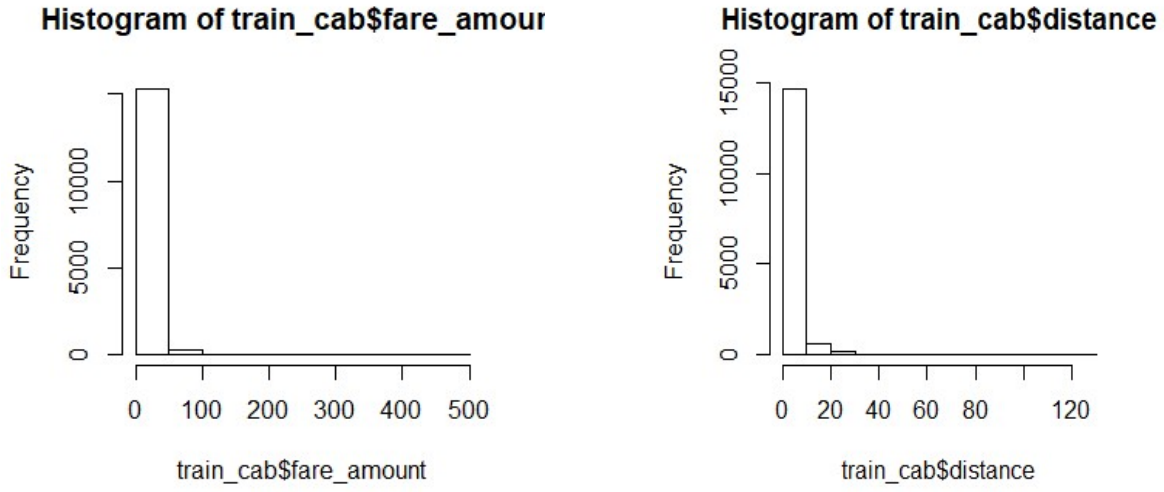


Fig 18 Histograms of all fare_amount and distance variable before log transform

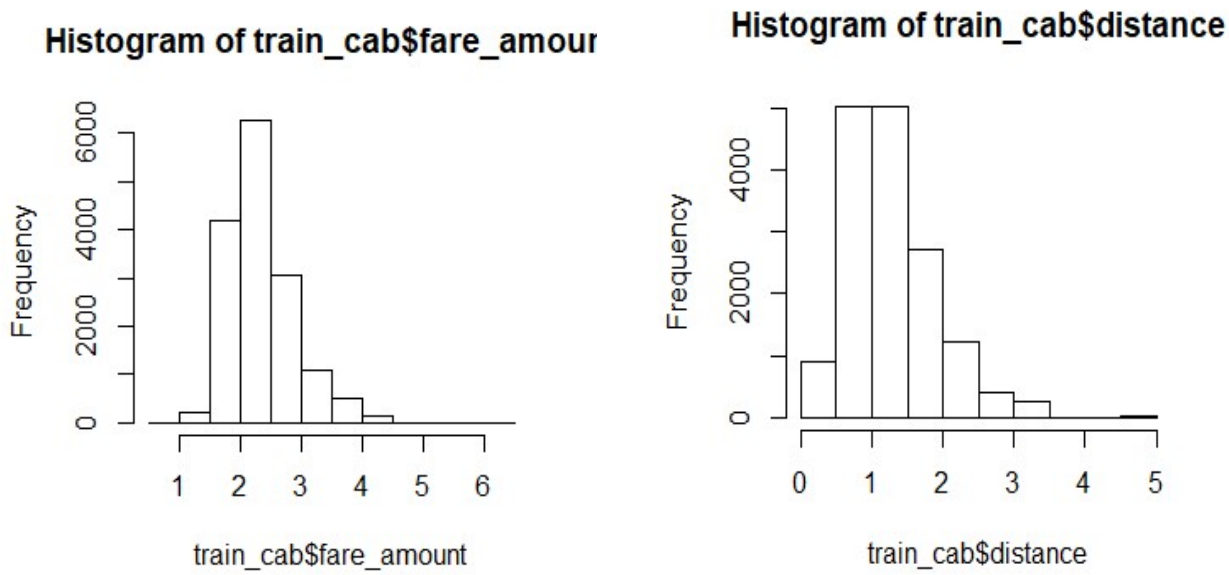


Fig 19 Histograms of all fare_amount and distance variable before log transform

Visualization for Finalized XGBoost Regression Model-

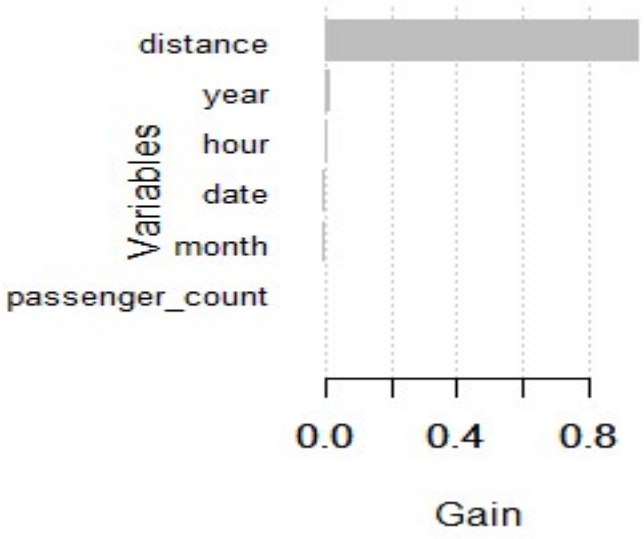


Fig 21 Variable Importance Plot of XGBoost Model

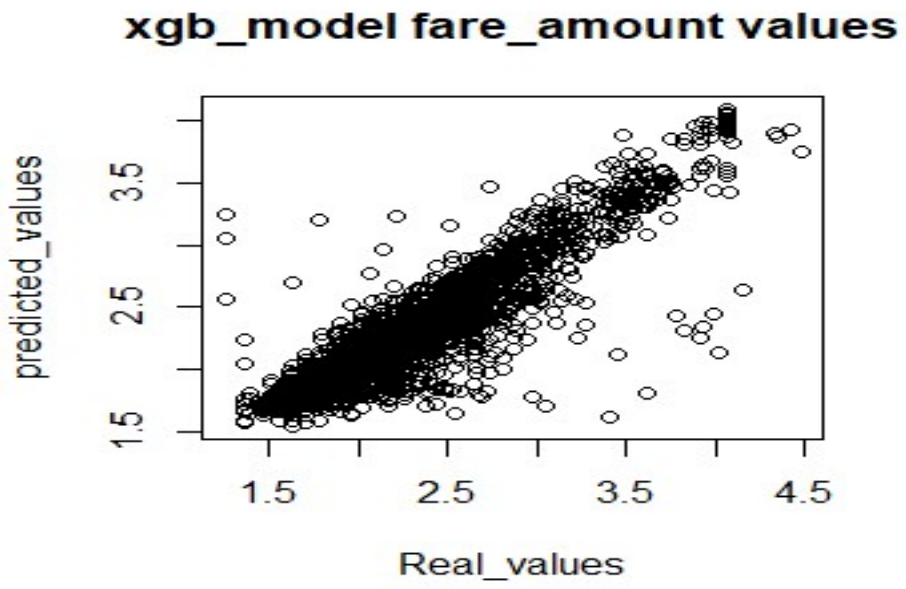


Fig 22 Scatter plot of real vs predicted values by XGBoost model

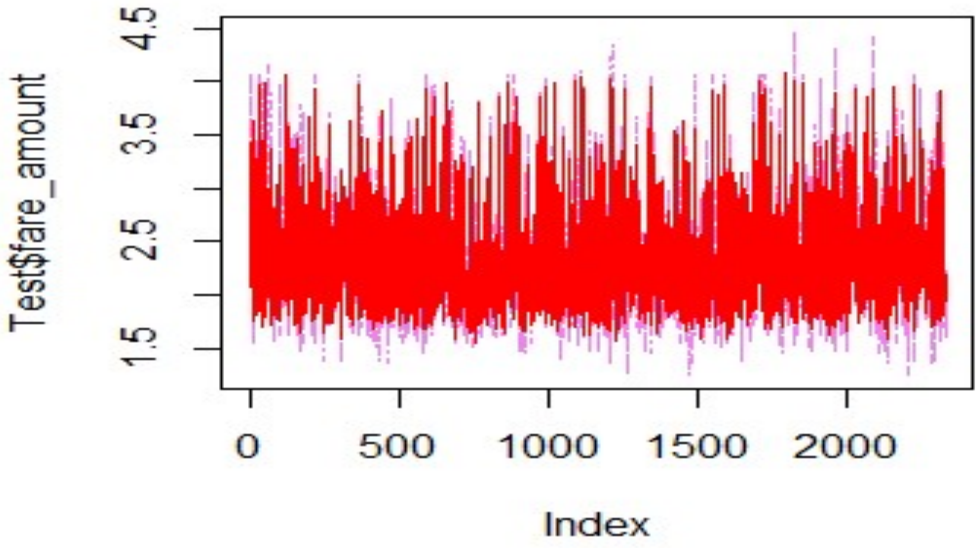


Fig 22 Line plot of real (Red color) vs predicted values (Violet color) by XGBoost model

CHAPTER 5

CONCLUSION

5.1 Model Evaluation:

In this section we will do evaluation of all the models and find the model with high performance and at last we will use that model for predicting the fare_amount for given test data that is the objective of this project.

❖ *Error metrics in Python-*

The error metrics that we will use for evaluating the model performance in python code are listed below.

- Adjusted R-square
- MAPE
- RMSLE
- Accuracy

We have already calculated these error metrics for all models we have built. So we have listed the all the error metrics for comparison and evaluation of our models. We have taken all the values of the subset test data that we created from our main train_cab data. The table below showing the error metrics of all the models that we have built in python

Table Error metrics of models in python code

Error Metrics	MLR Model	Lasso Model	Ridge model	Decision Tree model	Random Forest Model	XGBoost Model	XGBoost Tuned Model
Adjusted R-square (%)	73.84	76.10	76.10	70.39	79.00	80.68	**81.11**
MAPE (%)	8.13	7.66	7.66	9.7	7.66	7.41	**7.16**
RMSLE	0.079	0.07	0.076	0.086	0.073	0.071	**0.07**
Accuracy (%)	91.87	92.34	92.34	90.3	92.34	92.59	**92.84**

We can see the error metrics for all the models. Form the table we can easily see that the error metrics for Random Forest model and XGboost model are good. But error metrics of XGBoost tuned model are very good as compared to all other the models. We

have got the Adusted R-square value 81.11 which is above 80 % and MAPE is 7.16 which is lowest form all the models. So the accuracy of the model is 92.84 %. So we will finalize XGBoost Tuned Regression model to predict the fare_amount for the given test data set.

❖ *Error metrics in R-*

The error metrics that we will use for evaluating the model performance in R code are listed below.

- R-square
- MAPE
- RMSE
- MAE
- Accuracy

We have calculated these error metrics for all models that we have built in R which are shown below.

Error Metrics	**Linear Regression model**	**Decision Tree model**	**Decision Tree Tuned model**	**Random Forest Model**	**XGBoost Regression Model**
R-square (%)	74.93	73.92	73.92	78.72	**80.10**
MAPE(%)	7.85	8.58	8.58	7.55	**7.14**
RMSE	0.2730	0.2784	0.2784	0.2521	**0.2436**
MAE	0.180	0.1963	0.1963	0.1716	**0.1632**
ACCURACY (%)	92.15	91.42	91.42	92.45	**92.65**

We can see above the error metrics for all the models that we have built R. From the table we can easily see that the error metrics for Random Forest model and XGboost model are good. But error metrics of XGBoost model are very good as compared to all other the models. We have got the R-square value 80.10 which is above 80 % and MAPE is 7.14 which is lowest form all the models. So the accuracy of the model is 92.65 %. We can see almost same results we have got in R code. So we will finalize XGBoost Regression Model from R code to predict the fare_amount for the given test data set

5.2 Selecting the Final Model:

- **Final model in python code-**

  We have already discussed the error metrics for all our models that we have built and we have found that the performance of XGBoost tuned regression model very good as compared to other models so we have finalized **XGBoost Tuned Regression Model** as our final model for our project in **python code.**

- **Final model in R code-**

  We have already discussed the error metrics for all our models that we have built in our R code and we have found that the performance of XGBoost regression model very good as compared to other models so we have finalized **XGBoost Regression Model** as our final model for our project in **R code.**

5.3 Predicting the fare_amount for given test data set and saving the output

So now we will predict the fare_amount that is our target variable for the actual given test data set given to us in our problem statement. The python code for that is shown below.

```
# Now we will predic cab fare for given test data using our XGB_Final model
#Apply XGB_Final model on test data
Cab_fare_test = XGB_Final.predict(test_cab)

# lets see the predicted  array
Cab_fare_test

# lets add predicted values in our Given test data
test_cab['predicted_fare_amount'] = Cab_fare_test

# Save output of our project to csv file test_predicted in our working
directory
# That is with predicted cab fare amount variable in given test data.
test_cab.to_csv('test_predicted.csv')
```

We have predicted fare_amount for test data and saved the values by adding a new variable 'predicted_fare_amount' in the original test data. Now we have created new data frame names as 'test_preidtced' which includes the predicted fare amount variable. The top 5 observations of this new data are shown below.

passenger_count	month	year	weekday	hour	distance	predicted_fare_amount
1	1	2015	1	13	1.200263	2.375037
1	1	2015	1	13	1.230751	2.425167
1	10	2011	5	11	0.481303	1.719234
1	12	2012	5	21	1.085078	2.208984
1	12	2012	5	21	1.853612	2.780676

So now we will predict the fare_amount by using our finalized model in R code. The R code for that is shown below.

```
## We will predict fare_amount for given test data using XGBoost model
test_cab_new = as.matrix(sapply(test_cab,as.numeric))
test_cab_new
# lets predict fare_amount for given test data with problem statement.
xgb_predict_test = predict(xgboost_model,test_cab_new)
## lets store predicted fare amount in given test data
test_cab$predicted_fare= with(test_cab,xgb_predict_test)
head(test_cab)
## Now lets save the given test data with added predicted fare variable  as
"test_predicted_R" in our working dir.
write.csv(test_cab,"test_predicted_R.csv",row.names = FALSE)
```

We followed the same steps that we have done in python code. We have added new variable 'predicted_fare' in test data set. This variable contains the predicted values by our final model in R code. So the top 5 observations of test_predicted_R data frame with fare_amount predicted values in R code are shown below.

passenger_count	date	month	year	hour	weekday	distance	predicted_fare
1	27	1	2015	13	2	2.3232595	10.141560
1	27	1	2015	13	2	2.4253529	10.141560
1	8	10	2011	11	6	0.6186279	5.534328
1	1	12	2012	21	6	1.9610325	9.309200
1	1	12	2012	21	6	5.3873013	18.298740
1	1	12	2012	21	6	3.2225489	12.125039

APPENDIX A – COMPLETE R CODE FILE

```
# Project 2 Edwisor - Cab Fare Prediction

# Instruction to run code
# You need to set the working directory path in the following code
# And the working directory that you have set must have train_cab and test data sets
# Lets clear the environment
rm(list=ls())
# set working directory
setwd('F:\\data Scientist\\project 2 Cab Fare Prediction  R')
getwd()
# lets load the libraries
# lets save the libraries in x variable
x= c("ggplot2","corrgram","DataCombine","scales","psych","gplots",
     "Metrics" ,"inTrees","DMwR","car", "caret","rlang","usdm", "Information",
"randomForest",
     "unbalanced", "C50", "dummies", "e1071", "MASS", "ROSE", "rpart",
"gbm",'xgboost','stats')
# lets load the libraries
#install.packages("psych")
lapply(x, require, character.only=TRUE)
rm(x)
# lets load the data
# the data given to us is train_cab and test data
train_cab= read.csv("train_cab.csv", header=T, na.strings = c(' ','', 'NA'))
test_cab= read.csv('test.csv', )
#### features in given data are as follows in problem statement
# 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

# lets observe the given data
str(train_cab)
str(test_cab)
# we can see the shape of the train_cab data there are 16067 rows and 7 variables
including fare_amount as a target varibale
# test data has 9914 rows and 6 variables
# summary of train_cab data
summary(train_cab)
##### We can see summary of train data
##### Findings of summary
# pickup_datetime variable is timestamp variable which in object data type
# Also we know that longitude ranges from (-180 to +180) and latitude from (-90 to +90)
# pickup_latitude max value is 401.08 which is above 90 so we need to look into this
issue.
# passenger_count variable has minimum value 1 and max 5345 which is not possible
# fare_amount is price variable it should be numeric also it contains negative values
# there are missing values

# summary of train_cab data
summary(test_cab)
# Findings for test_cab data
# all varibales are within range
# just we need to convert pickup_datetime variable to timestamp variable
# lets see top 5 observation of train and test data
head(train_cab)
head(test_cab)
# lets first convert variables in both data to proper shapes
# lets convert pickup_datetime to date time variable form both data
str(train_cab$pickup_datetime)
# now lets convert fare_amount to numeric
train_cab$fare_amount = as.numeric(as.character(train_cab$fare_amount))
str(train_cab$fare_amount)
```

```r
# lets round off the passenger_count variable values from both data sets as it is person
number
train_cab$passenger_count=round(train_cab$passenger_count)
test_cab$passenger_count=round(test_cab$passenger_count)
# check the data types again
str(train_cab)
str(test_cab)
# now all the variables are in proper data types lets move further
######## EXPLORATORY DATA ANALYSIS
# We observed that variables are not within the standard range
# so lets do some data cleaning operations on the data
### passenger_count varibale
summary(train_cab$passenger_count)
# we can see minimum value is 0 and maximum is 5345 which is not possible so we will
remove those observations
# We will assume maximum 6 passengers in cab so outside form 1 to 6 we will remove all
observations
# lets check count of values below 1
nrow(train_cab[which(train_cab$passenger_count<1),])
# count is 58
# lets check count of values above 6
nrow(train_cab[which(train_cab$passenger_count>6),])
# count is 20
# We can not impute these values of we will remove these 78 observations
train_cab = train_cab[-which(train_cab$passenger_count > 6),]
train_cab = train_cab[-which(train_cab$passenger_count < 1 ),]
# unique values in train data for passenger_count
unique(train_cab$passenger_count)
# unique values in test data for passenger_count
unique(test_cab$passenger_count)
# So we have cleaned the passenger_count variable form train and test data lets move
further
## fare_amount variable
summary(train_cab$fare_amount)
# we can see minimum value is negative and maximum is 54343 so we will remove these
observations with negative values
# lets sort in descending order
sort(-train_cab$fare_amount)
# we can see values above 453 are above range or not practical
# lets check count of values below 1
nrow(train_cab[which(train_cab$fare_amount<1),])
# count is 5
# lets check count of values above 6
nrow(train_cab[which(train_cab$fare_amount>453),])
# count is 2
# So we will remove the  7 observations in fare_amount below 1 and above 453
train_cab = train_cab[-which(train_cab$fare_amount > 453),]
train_cab = train_cab[-which(train_cab$fare_amount < 1 ),]
summary(train_cab$fare_amount)
# now we can see the values are in the range.
#### Lets do EDA on latitude and longitude variables from train and test data
# we already seen that only pickup_latitude variable is having value above range
# pickup_latitude varibale
summary(train_cab$pickup_latitude)
# we can see maximum value is 401.88 which is above range
nrow(train_cab[which(train_cab$pickup_latitude>90),])
# we can see there is only one observation above 90 so we will remove it
train_cab = train_cab[-which(train_cab$pickup_latitude > 90),]
summary(train_cab)
# Now we will check all 4 latitude and longitude variables for presence of "0" values.
nrow(train_cab[which(train_cab$pickup_longitude == 0 ),])
nrow(train_cab[which(train_cab$pickup_latitude == 0 ),])
nrow(train_cab[which(train_cab$dropoff_longitude == 0 ),])
nrow(train_cab[which(train_cab$pickup_latitude == 0 ),])
# we can see there are observations with "0" values 311,311,312 and 311
# so we will remove all these observations from each variable which are having "0"
values
train_cab = train_cab[-which(train_cab$pickup_longitude == 0),]
```

```
train_cab = train_cab[-which(train_cab$dropoff_longitude == 0),]
# Lets check for test data
summary(test_cab)
# we can see there is no variable outside the range for test data so we will proceed
further
#### Now we have cleaned both the data sets lets check it for missing values
#### MISSING VALUE ANALYSIS
# In this step we will find the variables with missing values.
# If missing values are present then we will impute them.
# create dataframe with missing value count
missing_val = data.frame(apply(train_cab,2,function(x){sum(is.na(x))}))
#convert row names into columns
missing_val$Columns= row.names(missing_val)
row.names(missing_val) = NULL
# rename the missing value count variable name as missing percentage
names(missing_val)[1]= "Missing_Percentage"
# calculate missing values percentage
missing_val$Missing_Percentage= (missing_val$Missing_Percentage/nrow(train_cab))*100
# lets arrange the Missing_Percentage column in descending order
missing_val=missing_val[order(missing_val$Missing_Percentage),]
# rearrange the columns
missing_val= missing_val[,c(2,1)]
# lets save the data frame of missing values in our working directory
write.csv(missing_val, "Missing_percentage.csv", row.names=F)
# lets plot the bar graph of missing values w.r.t variable
ggplot(data=missing_val[6:7, ], aes(x=reorder(Columns, -Missing_Percentage),
y=Missing_Percentage))+
  geom_bar(stat='identity', fill="DarkSlateBlue") +
  xlab("variables")+ ylab("Missing_percentage")+  ggtitle("Missing values of
train_cab")+theme_bw()
# lets create copy of the data set to use it for checking which method is good to impute
missing values.
train_cab_missingdata=train_cab
#train_cab=train_cab_missingdata
# Procedure to Impute missing values is as follows
# 1) Select any random observation from data and equal it with "NA"
# 2) Now impute that value by using mean, mode, median, KNN
# 3) Compare the value imputed by above methods with actual value.
# 4) select the method which will give more accurate result
# 5) Now choose that method and find all missing values in that variable.
# 6) Repeat above steps t0 impute all missing values.
## Mean and median are used for numeric variales
## Mode and KNN are used for Categorical variables
# Lets impute missing values in passenger_count variable
# we will check only with KNN
# we will not use mode because we have most of the observations having passenger count
value '1'
# passenger_count-
# Actual `value-1
# KNN- 1
# lets select any random observation
train_cab$passenger_count[900]
train_cab$passenger_count[900]= NA
# lets impute with KNN
#train_cab= knnImputation(train_cab, k=5)
train_cab$passenger_count[900]
# so we have finilized the KNN as imputation method for passenger_count now lets check
for fare_amount variable.
# Lets impute missing values in fare_amount variable
# first lets check which method is giving good accuracy from mean, median, KNN method
# lets select any random observation
train_cab$fare_amount[900]
train_cab$fare_amount[900]= NA
##### fare_amount
# Actual value=49.8
# mean= 11.3667
# median= 8.5
# KNN= 48.0639
```

```
# lets impute with mean
train_cab$fare_amount[is.na(train_cab$fare_amount)]=mean(train_cab$fare_amount, na.rm=T)
train_cab$fare_amount[900]
train_cab$fare_amount[900]= NA
# lets impute with medain
train_cab$fare_amount[is.na(train_cab$fare_amount)]=median(train_cab$fare_amount,
na.rm=T)
train_cab$fare_amount[900]
train_cab$fare_amount[900]= NA
# lets make passenger_count to numeric
#train_cab$passenger_count=as.numeric(train_cab$passenger_count)
# lets impute with KNN
#train_cab= knnImputation(train_cab, k=5)
train_cab$fare_amount[900]
#-------------------------------------------------------------
### We can see KNN imputation method is giving great accuracy to impute missing values
# So we will freeze this method for calculating missing values.
# lets impute all missing values with KNN
str(train_cab)
train_cab= knnImputation(train_cab, k=5)
# lets recheck the missing values
sum(is.na((train_cab)))
# lets check test data for missing values present or not
sum(is.na((test_cab)))
# lets round off the passenger_count variable values from both data sets
train_cab$passenger_count=round(train_cab$passenger_count)
test_cab$passenger_count=round(test_cab$passenger_count)
####### FEATURE ENGINEERING
# If we see our variables we have date_time variable which contains information of
day,month,year and time at that point.
# But These all are enclosed in one format.
# so we can not use that as it is and our ML model didnt regognise this variable
# so we will split that variable
# create new variables like date, month, year, weekday, hour and minute.
# Similarly if we see other four variables
pickup_longitude,pickup_latitude,dropoff_longitude, dropoff_latitude
# these are nothing but the coordinates of passenger where from he is picked up and
dropped by cab.
# But we cannot use these variables also for our ML model.
# As we are having passenger pickup and drop coordinates.
# we can create new variable from this that is 'DISTANCE 'distance' variable.
# It will give us the distance travelled by cab during each ride.
# Distance variable will be more important to decide Cab fare amount.
######## Feature Engineering on train data
# lets first do feature engineering on pickup_datetime varible from train data
str(train_cab$pickup_datetime)
#lets first convert variable from factor to date time variable
# lets create raw_date variable to extract and create date, month and year variables
from given date.
train_cab$raw_date= as.Date(train_cab$pickup_datetime)
# lets check it again for missing values
sum(is.na(train_cab$raw_date))
# we can see there are NA value in raw_date variable
# lets remove that rows having NA values
train_cab=train_cab[complete.cases(train_cab[,8]),]
# lets check again for missing values
sum(is.na(train_cab$raw_date))
sum(is.na(train_cab))
# so now we dont have missing values
# lets go further for creating new features from raw_date and pickup_datetime varibale
head(train_cab)
# Now lets extract date, month, year from this raw_date time variable
# lets extract date from pickup_datetime
train_cab$date= as.integer(format(train_cab$raw_date,'%d' ))
# lets extract month from pickup_datetime
train_cab$month= as.integer(format(train_cab$raw_date,'%m' ))
# lets extract year from pickup_datetime
train_cab$year= as.integer(format(train_cab$raw_date,'%Y' ))
```

```r
# lets create hour and weekday variables from given pickup_datetime variable
# lets extract hour from pickup_datetime
train_cab$hour = substr(as.factor(train_cab$pickup_datetime),12,13)
# lets extract weekday from pickup_datetime
train_cab$weekday= as.factor(format(train_cab$raw_date, '%u'))
# lets see the unique values in each new variables that we have created
head(train_cab)
unique(train_cab$date)
unique(train_cab$month)
unique(train_cab$year)
unique(train_cab$hour)
unique(train_cab$weekday)
# Here in weekday we will consider value 1 as monday and so on.
# lets check no of unique values in each new created variables
table(train_cab$date)
table(train_cab$month)
table(train_cab$year)
table(train_cab$hour)
table(train_cab$weekday)
###### Feature Engineering on test data
# lets first do feature engineering on pickup_datetime varible from test data
str(test_cab$pickup_datetime)
# lets see first 5 observations of data
head(test_cab)
#lets first convert variable from factor to date time variable
# lets create raw_date variable to extract and create date, month and year variables
from given date time variable.
test_cab$raw_date= as.Date(test_cab$pickup_datetime)
# lets check it again for missing values
sum(is.na(test_cab$raw_date))
# no missing values lets proceed further
# Now lets extract date, month, year from this raw_date variable
# lets extract date from pickup_datetime
test_cab$date= as.integer(format(test_cab$raw_date,'%d' ))
# lets extract month from pickup_datetime
test_cab$month= as.integer(format(test_cab$raw_date,'%m' ))
# lets extract year from pickup_datetime
test_cab$year= as.integer(format(test_cab$raw_date,'%Y' ))
# lets create hour and weekday variables from given pickup_datetime variable
# lets extract hour from pickup_datetime
test_cab$hour = substr(as.factor(test_cab$pickup_datetime),12,13)
# lets extract weekday from pickup_datetime
test_cab$weekday= as.factor(format(test_cab$raw_date, '%u'))
head(test_cab)
##### Now we have finished feature engineering of pickup_datetime variable for bot train
and test data
# So now we dont require pickup_datetime variable as we have created new feature form it
# So we will remove it
# Also we have created raw_date variable, we will remove that also
# we will remove above 2 variables from train and test data
train_cab=subset(train_cab, select= -c(pickup_datetime,raw_date))
test_cab=subset(test_cab, select= -c(pickup_datetime,raw_date ))
head(train_cab)
####### Lets do feature engineering of latitude and longitude variables
# we already discussed above that we can create distance variable from latitude and
longitude variables
# So lets create distance variable in both train and test data
# we will use haversine formula to calculate distance between two points of lat. and
long.
# Haversine formula calculates great circle distance on sphere with given lat and long.
# lets create function to convert decimal degrees to radians
Radians= function(Deg){
  (Deg*pi)/180
}
### lets write Haversine formulla to calculate distance
# lets create function for Haversine formula and store in Haversine object
Haversine= function(pi_lon,pi_lat, dr_lon, dr_lat){
```

```
  ### where pi_lon=pickup_longitude,    pi_lat= pickup_latitude,
  ###       dr_lon= dropoff_longitude,  dr_lat=dropoff_latitude

  # now lets convert pickup_latitude and dropoff_latitude to radians and save in
pi_lon_rad and dr_lon_rad to objectpi_lon_rad=Radians(pi_lon)
  dr_lon_rad=Radians(dr_lon)

  # lets substract them and save in sub_lon
  sub_lon= Radians(dr_lon-pi_lon)

  # now lets convert pickup_latitude and dropoff_latitude to radians and save in
pi_lat_rad and dr_lat_rad to object
  pi_lat_rad=Radians(pi_lat)
  dr_lat_rad=Radians(dr_lat)
  # lets substract them and save in sub_lat
  sub_lat= Radians(dr_lat-pi_lat)
  # lets write the formula
  H=sin(sub_lat/2)*sin(sub_lat/2)+ cos(pi_lat_rad)*cos(dr_lat_rad)* sin(sub_lon/2)*
sin(sub_lon/2)
  A=2*atan2(sqrt(H), sqrt(1-H))
  R=6371e3
  R*A/1000
}
## lets apply the Haversine formula on our train data
train_cab$distance= Haversine(train_cab$pickup_longitude, train_cab$pickup_latitude,
train_cab$dropoff_longitude, train_cab$dropoff_latitude)
# lets apply on test data
test_cab$distance= Haversine(test_cab$pickup_longitude, test_cab$pickup_latitude,
test_cab$dropoff_longitude,test_cab$dropoff_latitude)
# lest see top 5 observations of train data
head(train_cab)
# lets see structure of distance variable from train and test data
str(train_cab$distance)
str(test_cab$distance)
# lets see the summary of distance variable
summary(train_cab$distance)
## we can see minimum value is 0.00 and maximum is 5420.989 which is not practical
## lets sort the distance variable values in descending order
train_cab[order(-train_cab$distance),]
## we can see the value after 129.95 are increased highly so we will remove values above
130
## so we need to remove these observations
#### lets check no of values above '130' and equal to '0' in train data
nrow(train_cab[which(train_cab$distance == 0 ),])
nrow(train_cab[which(train_cab$distance >130 ),])
### there are total 157 observations so we will remove them.
# so we will remove all these observations which are having "0" values and above'130'
train_cab = train_cab[-which(train_cab$distance == 0),]
train_cab = train_cab[-which(train_cab$distance >130),]
str(train_cab)
## lets convert variables to factor
factor_data= colnames(train_cab[,c('date','month', 'year', 'weekday', 'hour',
'passenger_count' )])
for (i in factor_data){
  train_cab[,i]=as.factor(train_cab[,i])
}

# lets check the missing values
sum(is.na(train_cab))
##### so now we have completed the feature engineering of longitude ant latitude
variables
# we can now remove pickup_longitude, pickup_latitude, dropoff_longitude,
dropoff_latitude  variables
# Because we have created distance variable from those variables
# lets remove from train data adn test data
train_cab=subset(train_cab, select= -c(pickup_longitude, pickup_latitude,
dropoff_longitude, dropoff_latitude ))
test_cab=subset(test_cab, select= -c(pickup_longitude, pickup_latitude,
```

```
dropoff_longitude, dropoff_latitude ))
# top 5 observations
head(train_cab)
head(test_cab)
# now we have 8 variables in train data and 7 variables in test data after feature
engineering
## lets convert the factor variable to numeric for data visualization.
factor_data= colnames(train_cab[,c('date','month', 'year', 'weekday', 'hour',
'passenger_count' )])
for (i in factor_data){
  train_cab[,i]=as.numeric(train_cab[,i])
}
########## DATA VISUALIZATION
## In this section we will visualize our train data with respect to fare_amount variable
##### 'passenger_count' variable
## Lets plot histogram of passenger_count variable
hist(train_cab$passenger_count)
# Finding-  single travelling passengers are most frequent travellers.
## Lets plot scatter plot of passenger_count variable and fare_amount
ggplot(train_cab,aes(passenger_count,fare_amount)) +
  geom_point(alpha=0.5,color="DarkSlateBlue") +
  labs(title = "Scatter Plot b/w passenger_count and fare_amount", x =
"passenger_count", y = "fare_amount")+
  scale_color_gradientn(colors=c('light green')) +
  theme_bw()
# Finding- We can see highest fare is from single passenger.
##### 'weekday' variable
### Lets plot scatter plot of weekday variable adn fare_amount
ggplot(train_cab,aes(weekday,fare_amount)) +
  geom_point(alpha=0.5,color="DarkSlateBlue") +
  labs(title = "Scatter Plot b/w weekday and fare_amount", x = "weekday", y =
"fare_amount")+
  scale_color_gradientn(colors=c('blue')) +
  theme_bw()
# Finding- Monday to friday cab fare is high
##### 'hour' variable
### Lets plot scatter plot of hour variable and fare_amount
ggplot(train_cab,aes(hour,fare_amount)) +
  geom_point(alpha=0.5,color="DarkSlateBlue") +
  labs(title = "Scatter Plot b/w hour and fare_amount", x = "hour", y = "fare_amount",
xlab("hour"), ylab("fare_amount"))+
  scale_color_gradientn(colors=c('blue')) +
  theme_bw()
# Finding- After 8 pm cab fare charge is higher
#### 'distance' variable
### Lets plot scatter plot of distance variable and fare_amount
ggplot(train_cab,aes(distance,fare_amount)) +
geom_point(alpha=0.5,color="DarkSlateBlue") +
  labs(title = "Scatter Plot b/w distance and fare_amount",x = "distance",y =
"fare_amount")+
  scale_color_gradientn(colors=c('blue')) +  theme_bw()
# Finding- There is a linear relatinship between distance and fare_amount
# so distance variable is very important w.r.t fare_amount
############ FEATURE SELECTION
# lets convert our categorical variables to factor agian for feature selection
techniques.
factor_data= colnames(train_cab[,c('date','month', 'year', 'weekday', 'hour',
'passenger_count' )])
for (i in factor_data){
  train_cab[,i]=as.factor(train_cab[,i])
}
###### Correlation plot
## By this plot we can check the correlation between variables
### lets select the numeric vaiables from train data
numeric_index = sapply(train_cab,is.numeric)
numeric_data = train_cab[,numeric_index]
# Correlation plot to check correlation of variables
corrgram(train_cab[,numeric_index],order=F, upper.panel=panel.pie , main="correlation
```

```
plot")
#### we can see in plot distance and fare_amount are very highly correlated with each
other
##### ANOVA test
### This test is perfomed to check weather the means of categories of independant
variables are equal or not
### It is decided based on P value and it is hypothesis based test.
## if p value is less than 0.05 then we reject the null hypothesis saying that means are
not equal
### if p value is more than 0.054 we say accept the null hypothesis saying that means
are equal.
### lets store all categorical variables in one variable.
factor_index=sapply(train_cab, is.factor)
factor=train_cab[,factor_index]
for(i in 1:6){
  print(names(factor)[i])
  cnt= train_cab[,1]
  anova=aov(cnt~factor[,i], data=factor)
  print(summary(anova))
}
### we can see weekday and date variable p value is more than 0.05
# so we will remove weekday variable from train and test data
train_cab=subset(train_cab, select=-c(weekday))
head(train_cab)
test_cab=subset(test_cab, select=-c(weekday))
# lets convert facor variable to numeric for VIF calculation
factor_data= colnames(train_cab[,c('date','month', 'year', 'hour', 'passenger_count' )])
for (i in factor_data){
  train_cab[,i]=as.numeric(train_cab[,i])
}
#### Multicollinearity test of all independant variables
### it gives whether there is dependancy between each independant variable with other is
present or not
#### lets load the library
library(usdm)
# we will check the Multicollinearity by VIF that is variance inflation factor
## if VIF is between 1 to 5 there is no Multicollinearity
# if VIF is more than 5 there is high Multicollinearity
# lets calculate VIF for all independant variables
vif(train_cab[,-1])
# lets print the results of VIF
vifcor(train_cab[,-1], th = 0.9)
### we can see VIF of all variables is between 1 to 5
## SO we can say that there is now multicollinearity between independant variables.
### FEATURE SCALING
### lets plot the histogram to see the dtaa distribution of distance and fare_amount
variable
hist(train_cab$fare_amount)
hist(train_cab$distance)
## We know that Normalizatoin and Standardization are the two methods for doing scaling
## But our data has outliers and normalization is sensititve to outliers
## so we will do log transform of fare_amount and distance variable.
train_cab$fare_amount=log1p(train_cab$fare_amount)
test_cab$distance=log1p(test_cab$distance)
train_cab$distance=log1p(train_cab$distance)
## Lets plot histograms again to see the data distribution.
hist(train_cab$distance)
hist(train_cab$fare_amount)
# lets create copy of scalled data.
train_cab_scaled= train_cab
#### Lets split train data into Train and Test data to build ML models on it
set.seed(1200)
Train.index = sample(1:nrow(train_cab), 0.85 * nrow(train_cab))
Train = train_cab[ Train.index,]
Test  = train_cab[-Train.index,]
# lets remove the unwanted data from R environment
rmExcept(c('train_cab', 'test_cab', 'Train', 'Test', 'train_cab_scaled',
'train_cab_missingdata'))
```

```
##### ERROR METRICS
### we have different error metrics to analize the ML model
## We will use rsquare,MAPE,Accuracy, rmse and mae for our models
# rsquare- It will tell us how much variation of target variable is explained by
independant variables
# MAPE- it is a percentage error between real and predicted values of target variable
# Accuracy- It is the accuracy of model in percentage,  which is (100- MAPE)
# rmse- It is Standard deviation of residuals i.e. prediction errors. It should be low
and between 0.2 to 0.5 is good
# mae- It is comparision of predited versus observed value of target variable. It gives
the error and it should be less.
## So we will calculate all these error metrics for each model
## We require rsquare value high and MAPE value as much less as possible
## SO we will select the model whose performance is like high rsquarew and low MAPE
value.
#### Now lets build the regression models on Train and Test data as our target variable
is numeric variable.
######### LINEAR REGRESSION MODEL
# Lets build linear regression model
lr_model=lm(fare_amount~., data=Train)
#summary of regression model
summary(lr_model)
# prediction on test data
predictions_lr=predict(lr_model, Test[,2:7])
Predictions_LR_train = predict(lr_model,Train)
# Function to calculate r square error metric
rsquare=function(y,y1){
  cor(y,y1)^2
}
# Function to calculate error metric mape which is Mean Absolute Percentage Error.
mape= function(y, yhat){
  mean(abs((y-yhat)/y))*100
}
# calculate error metrics to evaluate the model for train data
rsquare(Train[,1],Predictions_LR_train)
mape(Train[,1],Predictions_LR_train)
rmse(Train[,1],Predictions_LR_train)
mae(Train[,1],Predictions_LR_train)
##Predictive performance of model using error metrics
# rsquare= 0.7569
# mape(error rate)=7.53
# Accuracy =92.47
# rmse=0.2694
# mae=0.1729
# calculate error metrics to evaluate the model for test data
rsquare(Test[,1],predictions_lr)
mape(Test[,1],predictions_lr)
rmse(Test[,1],predictions_lr)
mae(Test[,1],predictions_lr)
##Predictive performance of model using error metrics
# rsquare= 0.7493
# mape(error rate)=7.85
# Accuracy =92.15
# rmse=0.2730
# mae= 0.180
## We can see the model is doing well on train data as compare to test data.
## But there is no much difference between error metrics of train and test data
## So we will calculate error metrics on test data only for further models
######### DECISION TREE MODEL OF REGRESSION
#Decision tree for regression
fit=rpart(fare_amount~., data= Train, method="anova")
# predict for new test data
predictions_dt= predict(fit, Test[,-1])
# compare real and predicted values of target variable
comparision_dt=data.frame("Real"=Test[,1], "Predicted"= predictions_dt)
#lets print error metrics
rsquare(Test[,1],predictions_dt)
mape(Test[,1], predictions_dt)
```

```
rmse(Test[,1],predictions_dt)
mae(Test[,1],predictions_dt)
##Predictive performance of model using error metrics  on test data
# rsquare= 0.7392
# mape(error rate)=8.58
# Accuracy =91.42
# rmse=0.2784
# mae= 0.1963
#### lets do parameter tunning of decision tree model using random search cv
## lets first set the model with default parameters
control = trainControl(method="repeatedcv", number=5, repeats=1, search='random')
maxdepth = c(1:30)
params = expand.grid(.maxdepth=maxdepth)
# Lets build a model using above parameters on train data
DT_model = caret::train(fare_amount~., data=Train,
method="rpart2",trControl=control,tuneGrid= params)
print(DT_model)
#lets look best parameters
best_parameters = DT_model$bestTune
print(best_parameters)
# maxdepth= 7
# Now lets build Decision tree model again using best parameters that we got above
DT_tunned = rpart(fare_amount~.,Train, method = 'anova',maxdepth=7)
print(DT_tunned)
#lets predict for test data
predictions_DT_tunned = predict(DT_tunned,Test)
# calculate error metrics to evaluate the model
rsquare(Test[,1],predictions_DT_tunned)
mape(Test[,1], predictions_DT_tunned)
rmse(Test[,1],predictions_DT_tunned)
mae(Test[,1],predictions_DT_tunned)
##Predictive performance of Decision tree tunned model using error metrics on test data
# rsquare= 0.7392
# mape(error rate)=8.58
# Accuracy =91.42
# rmse=0.2784
# mae= 0.1963
####### RANDOM FOREST MODEL OF REGRESSION
# creating the model
RF_model= randomForest(fare_amount~., Train, importance=TRUE, ntree= 80)
RF_model
# convert rf object to trees
treeList= RF2List(RF_model)
# extract the rules from the model
rules=extractRules(treeList, Train[,-1])
rules[1:2,]
# make rule readable
readablerules=presentRules(rules,colnames(Train))
# get rule metrics
ruleMetric=getRuleMetric(rules, Train[,-1], Train$fare_amount)
#predict the test data using RF model
predictions_rf=predict(RF_model,Test[,-1])
# Calculate error metrics to evaluate the performance of model
rsquare(Test[,1],predictions_rf)
mape(Test[,1],predictions_rf)
rmse(Test[,1],predictions_rf)
mae(Test[,1],predictions_rf)
##Predictive performance of model using error metrics
# rsquare= 0.7872
# mape(error rate)= 7.55
# Accuracy =92.45
# rmse=0.2521
# mae= 0.1716
#### Now lets improve the accuracy using XGBoost ensemble model
## lets store our train and test data in matrix form
train_data_matrix = as.matrix(sapply(Train[-1],as.numeric))
test_data_data_matrix = as.matrix(sapply(Test[-1],as.numeric))
# lets build the XGBoost model on train data
```

```
xgboost_model = xgboost(data = train_data_matrix,label = Train$fare_amount,nrounds =
15,verbose = FALSE)
# lets see the summary of our model
summary(xgboost_model)
# lets apply the model on test data to predict.
xgb_predictions = predict(xgboost_model,test_data_data_matrix)
# Lets create data frame of real and predicted values of target variable by xgboost
model
comparison_xgb=data.frame("Real"=Test[,1], "Predicted"= xgb_predictions)
## lets see the relation between real and predicted values by plot.
plot(Test$fare_amount, xgb_predictions, xlab= 'Real_values', ylab= 'predicted_values',
main='xgb_model fare_amount values')
## we can see there is linear relationship between real and predicted values
# plotting the line graph for real and predicted values of target variable
plot(Test$fare_amount, type="l", lty=4, col="violet")
lines(xgb_predictions, col="red")
# lets plot variable importance plot w.r.t fare_amount variable.
# lets store imporantat variables gain and frequency in 'imp' object
imp= xgb.importance(feature_names = colnames(Test[,2:7]), model =xgboost_model )
imp
# lets plot the variable importance plot.
xgb.plot.importance(importance_matrix = imp[1:7], xlab= 'Gain', ylab= "Variables")
# Calculate error metrics to evaluate the performance of model
rsquare(Test[,1],xgb_predictions)
mape(Test[,1],xgb_predictions)
rmse(Test[,1],xgb_predictions)
mae(Test[,1],xgb_predictions)
##Predictive performance of model using error metrics
# rsquare= 0.8010
# mape(error rate)= 7.1479
# Accuracy =92.85
# rmse=0.2436
# mae= 0.1632
### CONCLUSIUONS
# We have built (linear regression, Decision tree regression,
#          Decision tree tunned model, Random Forest regression and XGBoost Ensemble
Regression model.)
# We have used rsquare, MAPE, rmse and mae error metrics to evaluate the performance of
each model
# ALso we have calculated accuracy of each model
# Now From all the models we have built we seen that Random forest and XGBoost models
are good
## But XGBoost model is having rquare value above '80' and MAPE '7.14'
## we got the accuracy '92.86'% which is best amongst all the models we built
## So on the basis of these error metrics we will finalize 'XGBoost model as final model
## We will predict fare_amount for given test data using XGBoost model
test_cab_new = as.matrix(sapply(test_cab,as.numeric))
# lets predict fare_amount for given test data with problem statement.
xgb_predict_test = predict(xgboost_model,test_cab_new)
## lets store predicted fare amount in given test data
test_cab$predicted_fare= with(test_cab,xgb_predict_test)
head(test_cab)
## Now lets save the given test data with added predicted fare variable  as
"test_predicted_R" in our working dir.
write.csv(test_cab,"test_predicted_R.csv",row.names = FALSE)
```

APPENDIX B – COMPLETE PYTHON CODE FILE

```
Project Name - Cab Fare Prediction

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 forfare prediction. You need to design a system
that predicts the fare amount for a cab ride in the city.
# Instruction to run code
# You need to set the working directory path in the following code
# And the working directory that you have set must have train_cab and test data
sets
# loading libraries
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
import statsmodels.api as sm
from geopy.distance import geodesic
from patsy import dmatrices
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import Ridge,Lasso
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import GradientBoostingRegressor
from sklearn import metrics
# Lets Set working directory
os.chdir("F:\\data Scientist\\Project 2 Edwisor")
os.getcwd()
#For the given project we have provided the train and test data sets
seperately. so now we will load the data sets and observe it.
# load given train data
train_cab= pd.read_csv('train_cab.csv', na_values={"pickup_datetime":"43"})
EXPLORATORY DATA ANALYSIS
In this section we will observe/visualize the data and do some data cleaning
operations on both train and test data sets
# Lets see first five rows of train data
train_cab.head()
# lets see the shape of train data
train_cab.shape
# Statistical measures of data
train_cab.describe()
-We can see the train_cab data statistical measures in the above table.
-we know that latitude ranges from -90 to +90 and longitude ranges from -180 to
+180.
-But in above train_cab data table pickup_latitude variable maximum value is
above 90 so we need to remove those values. All other values of latitide and
longitude are within range.
-Also passenger_count variable contains minimum value as'0.0' and maximum value
as '5345.0'and its data type is float. So we also need to solve this issue.
# data types of train data variables
train_cab.dtypes
-We can see that fare_amount is in 'object' data type so it need to be
```

```
converted to 'numeric'also  pickup_datetime vatiable is a 'timestamp varibale'
so it is also need to be converted to 'date_time'
#Convert fare_amount varible from "object" data type to "numeric"
train_cab["fare_amount"] = pd.to_numeric(train_cab["fare_amount"],errors =
"coerce")
# Now convert pickup_datetime variable to date_time data type in train data
train_cab['pickup_datetime']=pd.to_datetime(train_cab['pickup_datetime'],errors
='coerce')
# load given
test_cab=pd.read_csv('test.csv')
# To see first five rows of test data
test_cab.head()
# lets see the shape of test data
test_cab.shape
# Statistical measures of data
test_cab.describe()
# data types of test data variables
test_cab.dtypes
In test data also we need to convert the pickup_datetime varibale to
'date_time' as it is related to time and date.
# Now convert pickup_datetime variable to date_time data type in train data
test_cab['pickup_datetime']=pd.to_datetime(test_cab['pickup_datetime'],errors='
coerce')
Now we will see some visualizations to understand the train_cab data in better
way
we will plot histogram of fare_amount to see the distribution of data
train_cab['passenger_count'].value_counts()
train_cab['fare_amount'].sort_values(ascending=False)
-By domain knowledge of this project we can say that fare_amount should not be
'0' or negative.
-we can see above the values beyond 453.00 are very high which is not
practicaly possible. so we need to remove the observations having values more
than '453'
# lets check that is there any values below '1' in fare_amount and 'negative'
print('values below 1=''={}'.format(sum(train_cab['fare_amount']<1)))
print('values above 453=''={}'.format(sum(train_cab['fare_amount']>453)))
train_cab[train_cab['fare_amount']<1]
# lets drop all '7' observations which are below  '1' and above '453'
train_cab = train_cab.drop(train_cab[train_cab['fare_amount']<1].index, axis=0)
train_cab = train_cab.drop(train_cab[train_cab['fare_amount']>453].index,
axis=0)
# lets plot box plot of passenger_count variable
plt.figure(figsize=(20,5))
plt.xlim(0,100)
sns.boxplot(x=train_cab['passenger_count'],data=train_cab,orient='h')
plt.title('Boxplot of passenger_count')
-We will assume maximum 6 passengers could travel in one cab
-From the above plot we can see that passenger_count in train_cab data has some
values more than '6' which is not possible practically.
train_cab["passenger_count"].describe()
we can see above passenger_count is having minimum value as '0' and maximum as
'5345' which is not practical. There should be at least one passenger and
maximum six. so we will remove the observations who have passenger_count more
than '6' and less than '1'.
# Remove passenger_count above '6' and below '1'
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)
# recheck
sum(train_cab['passenger_count']<1)
```

```
train_cab["passenger_count"].unique()
-As we can see the unique values of passenger_count varibale.
-It contains 'NA' values that is missing values so we will deal with it in
missing value analysis.
-It is having 1.3 as a unique value, But it is not practical as passenger can
not be 1.3
-so we will also remove the obersvations with value 1.3.
# Removing the observations in passenger_count with 1.3 value.
train_cab = train_cab.drop(train_cab[train_cab['passenger_count']==1.3].index,
axis=0)
-Now passenger_count will have unique values 1.0,2.0,3.0,4.0,5.0,6.0 and 'NA'
-It contains float values.
-We will convert the variable in proper data type and category after the
missing value analyis
#check the above values for test_cab data
test_cab["passenger_count"].unique()
-we can see that test_cab data does not contain outliers like train data.So we
have completed the processing of passenger_count varibale of both train and
test data.
-Now in next step we will look into longitude and latitude variables
-We already observed in train_cab data that pickup_latitude is above 90.
-so we will reomove those observations who are outside the limit because we can
not impute them.
train_cab[train_cab['pickup_latitude']>90]
train_cab = train_cab.drop((train_cab[train_cab['pickup_latitude']>90]).index,
axis=0)
# Now let us check the all 4 latitude and longitude variables for presence of
'0' value
location_var=
['pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitude']
for i in location_var:
    print('Zero value count in',i,'={}'.format(sum(train_cab[i]==0)))
-We can see all the four varibales contains '0' values so we will remove all
those observations which contains '0' value from that respective varibales.
# Remove observations with '0' value
for i in location_var:
    train_cab = train_cab.drop((train_cab[train_cab[i]==0]).index, axis=0)
Missing value analysis-
In this step we will find the variables with missing values. If missing values
are present then we will impute them.
# create dataframe with missing values
missing_val= pd.DataFrame(train_cab.isnull().sum())
missing_val
# reset the inde of rows
missing_val = missing_val.reset_index()
missing_val
we can see we have missing values in three variables
fare_amount,pickup_datetime,passenger_count so we need to impute them.
# lets rename the variables
missing_val=missing_val.rename(columns= {"index":"Variables",
0:"Missing_Percentage"})
missing_val
# calculate the percentage of missing values
missing_val["Missing_Percentage"]=
(missing_val["Missing_Percentage"]/len(train_cab))*100
missing_val
# lets sort the Missing percentage in descending order
missing_val=missing_val.sort_values("Missing_Percentage",
ascending=False).reset_index(drop=True)
missing_val
# Procedure to Impute missing values is as follows
```

```
# 1) Select any random observation from data having its value and equal it with
"NA"
# 2) Now impute that value by using mean, mode, median
# 3) Compare the value imputed by above methods with actual value.
# 4) select the method which will give more accurate result
# 5) Now choose that method and find all missing values in that variable.
# 6) Repeat above steps t0 impute all missing values.
# Now we will calculate missing values for fare_amount variable.
# we will use mean, median
# Mode is not useful as it is of numeric data type value
# lets selecet any random observation
train_cab['fare_amount'].loc[7230]
# fare_amount location 7230 value
# Actual value-6.9
# Mean-11.36
# Median-8.5
# Now make this value equal to "NA"
train_cab['fare_amount'].loc[7230]=np.nan
train_cab['fare_amount'].loc[7230]
# lets impute with mean
train_cab['fare_amount']=train_cab['fare_amount'].fillna(train_cab['fare_amount
'].mean())
train_cab['fare_amount'].loc[7230]
# now again make that value as equal to 'NA'
train_cab['fare_amount'].loc[7230]=np.nan
train_cab['fare_amount'].loc[7230]
# lets impute with median
train_cab['fare_amount']=train_cab['fare_amount'].fillna(train_cab['fare_amount
'].median())
train_cab['fare_amount'].loc[7230]
#we can see median is giving better result than mean so we will calculate all
missing values using median
train_cab['fare_amount']=
train_cab['fare_amount'].fillna(train_cab['fare_amount'].median())
# Now lets calculate missing values for passenger count variable
# we will use mode method for imputing missing values
# lets selecet any random observation
train_cab['passenger_count'].loc[2]
# passenger_count location '2'
# Actual value-2.0
# mode= 1.0
# mean= 1.64
# median= 1.64
# Now make this value equal to "NA"
train_cab['passenger_count'].loc[2]=np.nan
train_cab['passenger_count'].loc[2]
# lets impute with mode
train_cab['passenger_count']=
train_cab['passenger_count'].fillna(train_cab['passenger_count'].mode()[0])
train_cab['passenger_count'].loc[2]
# now again make that value as equal to 'NA'
train_cab['passenger_count'].loc[2]=np.nan
# lets impute with mean
train_cab['passenger_count']=train_cab['passenger_count'].fillna(train_cab['pas
senger_count'].mean())
train_cab['passenger_count'].loc[2]
# now again make that value as equal to 'NA'
train_cab['passenger_count'].loc[2]=np.nan
# lets impute with median
train_cab['passenger_count']=train_cab['passenger_count'].fillna(train_cab['pas
senger_count'].median())
```

```
train_cab['passenger_count'].loc[2]
# we can see mean and median are giving good results than mode. so we will
freeze mean method.
# Also the value is of float data type but passenger_count can not be 1.64
# So we will convert passenger_count to int data type and make round off of its
values.
# lets impute all missing value with mean
train_cab['passenger_count']=train_cab['passenger_count'].fillna(train_cab['pas
senger_count'].mean())
# let us check the details of passenger_count variable
train_cab['passenger_count'].unique()
# lets round off the values
train_cab['passenger_count'].round()
train_cab['passenger_count'].unique()
# we know that there is missing value in pickup_datetime variable.
# which is only one missing value, so we will remove that observation instead
of imputing it
train_cab =
train_cab.drop(train_cab[train_cab['pickup_datetime'].isnull()].index, axis=0)
missing_val= pd.DataFrame(train_cab.isnull().sum())
missing_val
test_cab.isnull().sum()
-SO WE HAVE IMPUTED ALL THE MISSING VALUES. OUR BOTH DATA ARE FREE FROM MISSING
VALUES. SO NOW WE WILL PROCEED FURTHER.
-Now next step after missing value analysis is actually outlier analysis.
-But before going ot outlier analysis we will do some FEATURE ENGINEERING
FEATURE ENGINEERING
-If we see our variables we have date_time variable which contains information
of day,month,year and time at that point. But these all are enclosed in one
format there so we can not use that as it is and our ML model didnt regognise
this variable so we will split that variable and create new variables like
date,month,year, weekday, hour and minute.
-Similarly if we see other four variables
pickup_longitude,pickup_latitude,dropoff_longitude, dropoff_latitude these are
nothing but the coordinates of passenger where from he is picked up and dropped
by cab.
-But we cannot use these variables also for our ML model. As we are having
passenger pickup and drop coordinates we can create new variable from this that
is DISTANCE variable. It will give us the distance travelled by cab during each
ride.
-Distance variable will be more important to decide Cab fare amount.
-So we will create Distance variable
FEATURE ENGINEERING FOR TRAIN DATA
# First we will convert pickup_datetime variable from train_cab data to
date_time format
train_cab['pickup_datetime']=pd.to_datetime(train_cab['pickup_datetime'],
format='%Y-%m-%d %H:%M:%S UTC')
# Now we will seperate pickup_datetime variable
# create new variables like date, month, year, weekday, hour, minute
train_cab['date']= train_cab['pickup_datetime'].dt.day
train_cab['month']= train_cab['pickup_datetime'].dt.month
train_cab['year']= train_cab['pickup_datetime'].dt.year
train_cab['weekday']= train_cab['pickup_datetime'].dt.dayofweek
train_cab['hour']= train_cab['pickup_datetime'].dt.hour
train_cab['minute']= train_cab['pickup_datetime'].dt.minute
# lets see the top 5 observations of train_cab data to see newly extracted
variables
train_cab.head()
train_cab.dtypes
# Now we will create Distance variable
# There are two formulas to calculate distance Haversine formula and Vincenty
```

```
formulla
# Haversine calculate great circle distance between latitude/longitude points
assuming a spherical earth
# Vincenty calculate geodesic distance between latitude/longitude points on
ellipsoidal model of earth
# But we know earth is not complete sphere so haversine formula will not give
accurate results
#  so we will use vincenty to create distance variable in km
# lets create distance variable
train_cab['distance']=train_cab.apply(lambda y:
geodesic((y['pickup_latitude'],y['pickup_longitude']), (y['dropoff_latitude'],
y['dropoff_longitude'])).km, axis=1)
# lets see the top 5 observations
train_cab.head()
# lets see the details of distance variable
train_cab['distance'].describe()
-we can see that distance is having minimum value as 0.0 and maximum as 5434.77
but it is not practical.
# now lets arrange the distance values in descending order to see itin detail
train_cab['distance'].sort_values(ascending=False)
-we can se values after 129.37 are increased very drasticaly so we can call it
as a outlier. So we will remove those observations above 130 and having '0'
values
# lets see the observaions above
sum(train_cab['distance']==0),sum(train_cab['distance']>130)
# lets remove observations with '0' and more than '130' values from train data
train_cab=train_cab.drop(train_cab[train_cab['distance']==0].index,axis=0)
train_cab=train_cab.drop(train_cab[train_cab['distance']>130].index,axis=0)
FEATURE ENGINEERING FOR TEST DATA
we will just repeat all the operations to test_cab data
# pickup_datetime variable from test_cab data to date_time format
test_cab['pickup_datetime']=pd.to_datetime(test_cab['pickup_datetime'],
format='%Y-%m-%d %H:%M:%S UTC')
# Now we will seperate pickup_datetime variable
# create new variables like date, month, year, weekday, hour, minute in
test_cab data
test_cab['date']= test_cab['pickup_datetime'].dt.day
test_cab['month']= test_cab['pickup_datetime'].dt.month
test_cab['year']= test_cab['pickup_datetime'].dt.year
test_cab['weekday']= test_cab['pickup_datetime'].dt.dayofweek
test_cab['hour']= test_cab['pickup_datetime'].dt.hour
test_cab['minute']= test_cab['pickup_datetime'].dt.minute
test_cab.head()
# lets create distance variable in test data
test_cab['distance']=test_cab.apply(lambda y:
geodesic((y['pickup_latitude'],y['pickup_longitude']), (y['dropoff_latitude'],
y['dropoff_longitude'])).km, axis=1)
test_cab.head()
test_cab['distance'].describe()
# lets see the observaions above
sum(test_cab['distance']==0),sum(test_cab['distance']>130)
# lets remove observations with '0' and more than '130' values from test data
test_cab=test_cab.drop(test_cab[test_cab['distance']==0].index,axis=0)
test_cab=test_cab.drop(test_cab[test_cab['distance']>130].index,axis=0)
# lets see shape of train_cab and test_cab data
train_cab.shape, test_cab.shape
# lets make copy of train and test data sets
train_cab_splitted= train_cab.copy()
test_cab_splitted= test_cab.copy()
Removing the variables used for feature engineering
-We have applied feature engineering techniques to both train_cab and test_cab
```

```
data sets.
-Now we will remove the variables which we have used to create new variables.
-We have used pickup_datetime, pickup_longitude, pickup_latitude,
dropoff_longitude, dropoff_latitude so we will remove all these from both train
and test data
-Also we will remove minute varible as it is not looking so important w.r.t. to
target variable.
# lets store variables to drop in one variable
drop_variables= ['pickup_datetime', 'pickup_longitude', 'pickup_latitude',
'dropoff_longitude', 'dropoff_latitude', 'minute']
# lets drop above variables from train_cab data
train_cab= train_cab.drop(drop_variables,axis=1)
test_cab= test_cab.drop(drop_variables,axis=1)
train_cab.head()
train_cab.dtypes
# we can see in data the new variables that we created are of 'float' data type
# so we will convert all to 'int'
train_cab['passenger_count']= train_cab['passenger_count'].astype('int64')
train_cab['date']= train_cab['date'].astype('int64')
train_cab['month']= train_cab['month'].astype('int64')
train_cab['year']= train_cab['year'].astype('int64')
train_cab['weekday']= train_cab['weekday'].astype('int64')
train_cab['hour']= train_cab['hour'].astype('int64')
#we can see in data the new variables that we created are of 'flaot' data type
#so we will convert all variables to 'int' data type
test_cab['passenger_count']= test_cab['passenger_count'].astype('int64')
test_cab['date']= test_cab['date'].astype('int64')
test_cab['month']= test_cab['month'].astype('int64')
test_cab['year']= test_cab['year'].astype('int64')
test_cab['weekday']= test_cab['weekday'].astype('int64')
test_cab['hour']= test_cab['hour'].astype('int64')
DATA VISUALIZATIONS
- In this section we will visualize our data to understand it in better way.
#lets see how many passengers travelling in single ride
plt.hist(train_cab['passenger_count'],color='green')
-Single passengers are using cab service highly.
# lets see the relationship between passenger count and fare amount.
plt.figure(figsize=(10,5))
plt.scatter(x="passenger_count",y="fare_amount", data=train_cab,color='blue')
plt.xlabel('No. of passengers')
plt.ylabel('Fare_amount')
plt.show()
-Cab fare is more for passenger_count of 1 and 2 as compared to other.
# lets see the relationship between date and fare amount.
plt.figure(figsize=(10,5))
plt.scatter(x="date",y="fare_amount", data=train_cab,color='blue')
plt.xlabel('date')
plt.ylabel('Fare_amount')
plt.show()
-We can see there is no much effect of 'date' on 'fare_amount' and we are also
considering 'weekday' variable which is related to 'date'
-So we can drop date variable by observing its dependancy with other varibles
in chi square test
#Relationship between hour and Fare_amount
plt.figure(figsize=(10,5))
plt.scatter(x="hour",y="fare_amount", data=train_cab, color='blue')
plt.xlabel('hour')
plt.ylabel('fare_amount')
plt.show()
-from the above plot we can say that generaly cab fare is higher after 8 pm
# lets see no of cabs w.r.t. hour in day
```

```
plt.figure(figsize=(15,7))
train_cab.groupby(train_cab["hour"])['hour'].count().plot(kind="bar")
plt.show()
-we can see in above plot highest no of cabs are from 7 pm to 12 pm
-Also no of cabs are less upto 5 am
# lets see the realation between fare_amount and distance variable
plt.figure(figsize=(10,5))
plt.scatter(x="distance",y="fare_amount", data=train_cab,color='blue')
plt.xlabel('distance')
plt.ylabel('fare_amount')
plt.show()
-we can see there is a linear relatioship of distance with the fare_amount
#lest see the number of cab rides w.r.t. weekday
plt.figure(figsize=(15,7))
sns.countplot(x="weekday", data=train_cab)
-From the above plot we can say that weekday dosent have much impact on number
of cab rides
FEATURE SELECTION
-In this step we will see the corelations between target variable and
independant variables
-Also we will perform different tests to check the relation between the
depedant and indepedant variables
-Using the results of the test finally we decide which variables should be
selected and which should be dropped
train_cab.dtypes
# Even if we see the data types of all variables as 'float' and 'int'
# we know that the variables fare_amount and distance are only numeric
variables
# Other varables expect those are having some unique values only.
# for e.g. passenger_count has unique values 1,2,3,4,5,6.
# They are actually categorical variables
# So we will treat these variables as categorical variables but thier data type
remains 'int'
## Correlation analysis-  it is used to check the Correlation between the
variables.
# This can be done mostly on numeirc variables
# lets save our numeric variables in one variable
numeric= ['fare_amount','distance']

train_cab_corr= train_cab.loc[:,numeric]
# setting the  height and width of plot
f, ax=plt.subplots(figsize=(13,9))
# correaltion matrix
cor_matrix=train_cab_corr.corr()
# plotting correlation plot
sns.heatmap(cor_matrix,mask=np.zeros_like(cor_matrix, dtype=np.bool),
            cmap=sns.diverging_palette(250,12,as_cmap=True),square=True, ax=ax)

-We can see that their is very high corelation between distance and fare_amount
Chi-Square test-
-We have already performed anova test and we got thre result that all our
categorical variables are not dependant on each other.
-But still we will perform the Chi-square test to validate our results.
-This test is performed to check the dependancies between cateorical variables.
-Null hypothesis - variables are not dependant (P<0.05-Reject)
-Alternate hypothesis- variables are dependant (P<0.05-Accept)
# Lets perform chi square test of independance
from scipy.stats import chi2_contingency
factor_data=train_cab[['passenger_count', 'date', 'weekday', 'month', 'year',
'hour']]
for i in factor_data:
```

```
    for j in factor_data:
        if(i!=j):
            chi2,p,dof,
ex=chi2_contingency(pd.crosstab(train_cab[i],train_cab[j]))
            while(p<0.05):
                print(i,j,p)

                break


-we can 'date' is correlated with 'weekday' and most of the variables.
-Also in data visualizations we observed that 'date' is not having much impact
on 'fare_amount'
-So we will remove 'date' variable
Multicollinearity test using VIF(variance inflation factor)-
-VIF detects correlation between predictor variables i.e. relationship between
them.
-If two predictor variables are correlated then we can say there is presence of
Multicollinearity
-Multicollinearity affetcs the regression models so it should not present in
our variables
-So for this we do this test using VIF
-If VIF is between 1 to 5 then we say that there is no Multicollinearity
-If VIF>5 then there is a multicollinearity and we need to remove it or
reconsider the variables.
# lets create dataframe of predictor variables
outcome, predictors = dmatrices('fare_amount ~
distance+passenger_count+date+weekday+month+year+hour',train_cab,
return_type='dataframe')
# Lets calculate VIF for each independant variables form train_cab data
VIF = pd.DataFrame()
VIF["VIF"] = [variance_inflation_factor(predictors.values, i) for i in
range(predictors.shape[1])]
VIF["Predictors"] = predictors.columns
VIF
-We can see VIF for all the predictors is within the required range i.e. from
1-5
-So we can say that multicollinearity is not present in our independant
variables
# SO AFTER PERFORMING VARIOUS TESTS ON OUR DATA FOR FEATURE SELECTION WE HAVE
FOLLOWING OBSERVATIONS
# There is no multicollinearity in our data
# We will remove 'date' variable from both train and test data.
# Select all other variables for our ML models
# lets create a copy of our data selected for Machine learning
train_cab_selected= train_cab.copy()
test_cab_selected= test_cab.copy()
# lets drop 'date' variable from both the data sets
train_cab= train_cab.drop('date', axis=1)
test_cab= test_cab.drop('date', axis=1)
FEATURE SCALING-
-In this step actualy we need to do either Normalization or stadardization on
numeric variables.
-The scaling method is decided by observing histogram of variables.
# lets plot the histogram to see data distribution of fare_amount variable from
train_cab data
sns.distplot(train_cab['fare_amount'],bins='auto',color='green')
plt.title("Distribution for fare_amount variable ")
plt.ylabel("Density")
plt.show()
# lets plot histogram for distance variable from train data.
```

```
sns.distplot(train_cab['distance'],bins='auto',color='green')
plt.title("Distribution for distance variable ")
plt.ylabel("Density")
plt.show()
# lets plot the histogram to see data distribution distance variable from
test_cab data
sns.distplot(test_cab['distance'],bins='auto',color='green')
plt.title("Distribution for distance variable from test data")
plt.ylabel("Density")
plt.show()
-We can see the histogram for all the variables is left skewed.
-That means it contains the values which will impact more on ML model.
-As we know that Normalization is sensitive for these data type of data so we
can not use normalizatoin or standardization method of scaling
-So we will apply log tranform to both the variables to remove the effect pf
skewness.
# lets apply log tranform on numeric variables from train and test data
train_cab['fare_amount'] = np.log1p(train_cab['fare_amount'])
train_cab['distance'] = np.log1p(train_cab['distance'])
test_cab['distance'] = np.log1p(test_cab['distance'])
# lets plot the histogram to see data distribution of fare_amount variable from
train_cab data after log transform
sns.distplot(train_cab['fare_amount'],bins='auto',color='green')
plt.title("Distribution for fare_amount variable after log transform ")
plt.ylabel("Density")
plt.show()
train_cab.dtypes
-So now we have done all the data preprocessing operations on given train and
test data. Our data is clean and ready to be used for ML models. So lets
proceed further
# TRAIN TEST SPLITTING OF DATA.
# first store all predictor variables in 'x' and target variable in 'y' from
train_cab data
x=train_cab.drop(['fare_amount'],axis=1)  # predictors
y=train_cab['fare_amount']                  # target variable
# lets split our train_cab data into train and test data
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.20,random_state=
1)
# lets check teh shape of all the data sets we have created
x_train.shape, x_test.shape, y_train.shape, y_test.shape
MACHINE LEARNING MODEL BUILDING
#Before procedding to model building lets decide the error metrics
#The error metrics that we will use are RMSLE, RMSE, R-square, Adjusted R-
square, MAPE.
#We will use RMSLE metric because it uses log transform and gives best results
for the data
#we will also use Adjusted R-square value as it is more optimized that R-square
to see the performance of model
#We will create the functions to calculate above selected error metrics for
simplicity
#we will also define a function to calculate the predicions of models for train
and test data
# Function to calculate RMSLE
def rmsle(yt,yp):    # yt- y_train and yp- y_predicted
    log1 = np.nan_to_num(np.array([np.log(v + 1) for v in yt]))
    log2 = np.nan_to_num(np.array([np.log(v + 1) for v in yp]))
    calc = (log1 - log2) ** 2
    return np.sqrt(np.mean(calc))
# Function to calculate other error metrics
def error_metrics(yt, yp):
    print('r square  ', metrics.r2_score(yt,yp))
```

```
    print('Adjusted r square:{}'.format(1 - (1-metrics.r2_score(yt,
yp))*(len(yt)-1)/(len(yt)-x_train.shape[1]-1)))
    print('MAPE:{}'.format(np.mean(np.abs((yt - yp) / yt))*100))
    print('MSE:', metrics.mean_squared_error(yt, yp))
    print('RMSE:', np.sqrt(metrics.mean_squared_error(yt, yp)))

# Function to calculate and print the predictions of models for train and test
data with its error metrics
def output_scores(model):
    print('###############   Error Metrics of Train data   #################')
    print()
    # Applying the model on train data to predict target variable
    y_predicted = model.predict(x_train)
    error_metrics(y_train,y_predicted)
    print('RMSLE:',rmsle(y_train,y_predicted))
    print()
    print('###############   Error Metrics of Test data  #################')
    print()
    # Applying the model on train data to predict target variable
    y_predicted = model.predict(x_test)
    error_metrics(y_test,y_predicted)
    print('RMSLE:',rmsle(y_test,y_predicted))
MULTIPLE LINEAR REGRESSION
# lets build the model on train data
model = sm.OLS(y_train, x_train).fit()
# predict the test data
y_predict = model.predict(x_test)
# lets print model summary
print_model = model.summary()
print(print_model)
# error metrics on train and test data
output_scores(model)
Model results for test data-
-Adjusted r square for test data-0.7384
-MAPE is 8.13 % for test data
LASSO REGRESSION MODEL
#Lets build the lasso model
lasso_model = Lasso(alpha=0.00021209508879201905, normalize=False, max_iter =
500)
#Lets fit the lasso model on train data
lasso_model.fit(x_train,y_train)
#Lets print the coefficients for predictors
coefficients = lasso_model.coef_
print('coefficients of predictors:{}'.format(coefficients))
# Error metrics of train and test data
output_scores(lasso_model)
Lasso Model results for test data-
-Adjusted r-square for test data-0.7605
-MAPE is 7.66 %
RIDGE REGRESSION MODEL
# Lets build Ridge regression model
ridge_reg = Ridge(alpha=0.0005,max_iter = 500)
# Apply model on train data
ridge_reg.fit(x_train,y_train)
# print the coefficients of predictors
ridge_reg_coef = ridge_reg.coef_
print('coefficients of predictors:{}'.format(ridge_reg_coef))
 # Error metrics of train and test data
output_scores(ridge_reg)
Model results-
-Adjusted r-square for test data 0.7605
```

```
-MAPE is 7.66 %
-we can see reults of Lasso and ridge regression are same
-Now we will build decision tree model.
DECISION TREE REGRESSION MODEL
-This model creates the decision tree like flow chart and gives the rules to
predict the target variable.
# lets build the model and apply it on train data
fit_dt=DecisionTreeRegressor(max_depth= 2).fit(x_train, y_train)
# lets print the relative importance score for predictors
tree_features = fit_dt.feature_importances_
print('feature importance score of predictors:{}'.format(tree_features))
# Error metrics of train and test data
output_scores(fit_dt)
Model resultsfor test data-
-Adjusted r-square is 0.7039
-MAPE is 9.70 %
-Results are not good as compared to linear regression models
-So we need to improve accuracy of model.
RANDOM FOREST REGRESSION MODEL
-This is the improved version of decision tree model it uses many trees in one
model to improve the accuracy.
-It feeds error of one tree to another tree to improve the accuracy.
# Lets build the Random forest model
rf_model = RandomForestRegressor(n_estimators = 70, random_state=0)
# Aply model on train data
rf_model.fit(x_train, y_train)
# lets print the relative importance score for predictors
Forest_features = rf_model.feature_importances_
print('feature importance score of predictors:{}'.format(Forest_features))
# Error metrics of train and test data
output_scores(rf_model)
Model results for test data-
-Adjusted r-square is 0.79
-MAPE is 7.66 %
-Results are good as compared to linear regression models and Decision tree
model
XGBoost MODEL
-This model is nothing but the ensemble technique which provides optimized
results
-It is implementation of gradient boosted decision trees designed for speed and
performance
# Lets build XGboost model and apply on train data
xgb_model = GradientBoostingRegressor(n_estimators= 70, max_depth= 2)
xgb_model.fit(x_train, y_train)
# Error metrics of train and test data
output_scores(xgb_model)
-Results of XGBoost model are very good as compared to all other models
-But still we will do hyper parameter tuning of Random Forest model and XGBoost
model to improve the results
-The RandomizedSearchCV function will also do cross validation of model to get
best score
-cross validation means the train data is splited into subsets for our case its
is 5
-So from those 5 data sets one will be test and other will be train
-The model will take all these subset data one by one as test and calculate 5
scores
-the best score will be average of these 5 scores
PARAMETER TUNING OF RF AND XGBOOST MODELS TO IMPROVE RESULTS
# Build random forest model for tuning
RF = RandomForestRegressor(random_state = 42)
from pprint import pprint
```

```
# Lets see the parameters of our current RF model
print('Random Forest current parameters')
pprint(RF.get_params())
##Lets create Random hyperparameter grid and apply Randomized Search CV on
Random Forest Model
# lets build model again as RRF
RRF = RandomForestRegressor(random_state = 0)
# Create the random grid
random_grid_RRF = {'n_estimators': range(80,120,10), 'max_depth':
range(4,12,2)}
# appply Random Search CV on model with cross validation score 5
RRF_cv = RandomizedSearchCV(RRF, random_grid_RRF, cv=5)
# lets apply model on train data
RRF_cv.fit(x_train, y_train)
# Print the tuned parameters and score
print("Tuned Random Forest Parameters: {}".format(RRF_cv.best_params_))
print("Best score {}".format(RRF_cv.best_score_))
# Build xgboost model for tuning
xgb_t = GradientBoostingRegressor(random_state = 42)
from pprint import pprint
# Lets see the parameters of our current xgboost model
print('XGBoost currrent Parameters \n')
pprint(xgb_t.get_params())
##Lets create Random hyperparameter grid and apply Randomized  Search CV on
XGBoost model
# lets build model again as XBG
XGB = GradientBoostingRegressor(random_state = 0)
# Create the random grid
random_grid_XGB = {'n_estimators': range(90,120,10),'max_depth': range(1,10,1)}

# Apply Random Search CV on model with cross validation score 5
XGB_cv = RandomizedSearchCV(XGB, random_grid_XGB, cv=5)
# lets apply model on train data
XGB_cv.fit(x_train, y_train)
# Print the tuned parameters and score
print("Tuned XGBoost Parameters: {}".format(XGB_cv.best_params_))
print("Best score {}".format(XGB_cv.best_score_))
FINAL CONCLUSIONS-
-We have built Multiple linear Regression models, Decision tree, Random forest
and XGBoost models.
-But the results i.e. r-square, Adjuested r-square and MAPE metrics for test
data of Random forest and XGBoost model was good
-So we have done parameter tuning of these models to still see if we improve
results.
-After parameter tunning we can see that XGBoost model is giving highest best
score that is 0.80
-So we will finalize XGBoost model and apply this model with tuned parameters
again on train_cab data.
-Then finaly we will use that model to predict cab fare in our given test data
that is our objective of this project.
FINAL REGRESSION MODEL
# lets apply the tunned parameters and build our final XGBoost model on
train_cab data.
XGB_Final = GradientBoostingRegressor( n_estimators= 110, max_depth= 3)
# Apply the model on train data.
XGB_Final.fit(x_train,y_train)
# lets create and print the important features
XGB_Final_Features = XGB_Final.feature_importances_
print(XGB_Final_Features)
# Sorting important features in descending order
indices = np.argsort(XGB_Final_Features)[::1]
```

```
# Rearrange feature names so they match the sorted feature importances
Sorted_names = [test_cab.columns[i] for i in indices]
# create and set plot size
fig = plt.figure(figsize=(20,10))
plt.title("Feature Importance")
# Add horizontal bars
plt.barh(range(pd.DataFrame(x_train).shape[1]),XGB_Final_Features[indices],alig
n = 'center')
plt.yticks(range(pd.DataFrame(x_train).shape[1]), Sorted_names)
plt.savefig('Final XGBoost Model Important Features plot')
plt.show()
# "XGB_Final" train_cab and test_cab data scores
output_scores(XGB_Final)
XGB_Final Model results on test data-
-Adjusted r-square is 0.81  ## which is best from the all models that we have
applied on data. and above 80
-r square for is also 0.81
-MAPE is 7.1 %
-Accuracy  is 92.9
-RMSLE: 0.069 is also less.
PREDICTION OF CAB FARE FOR GIVEN TEST DATA
# Now we will predic cab fare for given test data using our XGB_Final model
#Apply XGB_Final model on test data
Cab_fare_test = XGB_Final.predict(test_cab)
# lets see the predicted  array
Cab_fare_test
# lets add Predicted values in our Given test data
test_cab['predicted_fare_amount'] = Cab_fare_test
test_cab.head()
# Save output of our project to csv file test_predicted in our working
directory
# That is with predicted cab fare amount variable in given test data.
test_cab.to_csv('test_predicted_python.csv')
```