Project title- Predict the City Taxi Trip Duration (By Princy Sahu)

Business Goal

To improve the efficiency of electronic taxi dispatching systems it is important to be able to predict how long a driver will have his taxi occupied.

If a dispatcher knew approximately when a taxi driver would be ending their current ride, they would be better able to identify which driver to assign to each pickup request.

ML Goal: To build a model that predicts the total ride duration of taxi trips in New York City

Data: The taxi trip records include fields capturing pick up and drop off dates/times, pick up and drop off locations, trip distances, itemized fares, rate types, payment types, and driver reported passenger counts. The data used in the attached datasets were collected and provided to the NYC Taxi and Limousine Commission (TLC) by technology providers authorized under the Taxicab.

Challenge

A regression model was developed to predict the duration of the taxi trip. The model was trained on a large dataset of over 1.5 million taxi trips, which were randomly split into training and testing sets.

The features used in the regression model included distance, pickup and dropoff coordinates, pickup datetime, day of the week, and weather conditions such as temperature, precipitation, and wind speed.

The model was evaluated using various metrics such as Mean Squre Error (MSE) and Root Mean Squared Error (RMSE),R2 Score,Adjusted R2-Score and was compared to other machine learning algorithms such as Linear Regression,Decision Tree Random Forest, Gradient Boosting andXgboost. The regression model outperformed the other algorithms in terms of accuracy, with an R2 score of 67%.

Overall, the NYC Taxi Time Prediction project demonstrates the potential for regression models to accurately predict the duration of taxi trips in New York City, using a combination of various features such as location, time, and distance.

#GitHub Link

-https://github.com/Princysahu11/Predict_Taxi_Trip_Duration_using_Machine_Learning

Step 1 : Introduction

Problem Statement-

To Build a machine learning model that predicts the duration of NYC taxi trip using the dataset which includes pickup time, geo-coordinates, the number of passengers, and several other variables

About Data

id	a unique identifier for each trip
vendor_id	a code indicating the provider associated with the trip record
pickup_datetime	date and time when the meter was engaged
dropoff_datetime	date and time when the meter was disengaged
passenger_count	the number of passengers in the vehicle (driver entered value)
pickup_longitude	the longitude where the meter was engaged
pickup_latitude	the latitude where the meter was engaged
dropoff_longitude	the longitude where the meter was disengaged
dropoff_latitude	the latitude where the meter was disengaged
store_and_fwd_flag	This flag indicates whether the trip record was held in vehicle memory before sending to the vendor because the vehicle did not have a connection to the server Y=store and forward; N=not a store and forward trip
trip_duration	duration of the trip in seconds

Step 2 : Data Exploration

Importing necessary libraries

```
import pandas as pd
import numpy as np
```

```
import matplotlib.pyplot as plt
import seaborn as sns
from numpy import math
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
import warnings
warnings.filterwarnings("ignore")

df=pd.read_csv("NYC Taxi Data.csv",sep=",")

df

                id  vendor_id      pickup_datetime
dropoff_datetime  \
0        id2875421          2  2016-03-14 17:24:55  2016-03-14
17:32:30
1        id2377394          1  2016-06-12 00:43:35  2016-06-12
00:54:38
2        id3858529          2  2016-01-19 11:35:24  2016-01-19
12:10:48
3        id3504673          2  2016-04-06 19:32:31  2016-04-06
19:39:40
4        id2181028          2  2016-03-26 13:30:55  2016-03-26
13:38:10
...            ...        ...                  ...                  ..
.
1458639  id2376096          2  2016-04-08 13:31:04  2016-04-08
13:44:02
1458640  id1049543          1  2016-01-10 07:35:15  2016-01-10
07:46:10
1458641  id2304944          2  2016-04-22 06:57:41  2016-04-22
07:10:25
1458642  id2714485          1  2016-01-05 15:56:26  2016-01-05
16:02:39
1458643  id1209952          1  2016-04-05 14:44:25  2016-04-05
14:47:43

         passenger_count  pickup_longitude  pickup_latitude  \
0                      1        -73.982155        40.767937
1                      1        -73.980415        40.738564
2                      1        -73.979027        40.763939
3                      1        -74.010040        40.719971
4                      1        -73.973053        40.793209
...                  ...               ...              ...
```

```
1458639                4         -73.982201        40.745522
1458640                1         -74.000946        40.747379
1458641                1         -73.959129        40.768799
1458642                1         -73.982079        40.749062
1458643                1         -73.979538        40.781750

         dropoff_longitude  dropoff_latitude store_and_fwd_flag
trip_duration
0               -73.964630         40.765602                  N
455
1               -73.999481         40.731152                  N
663
2               -74.005333         40.710087                  N
2124
3               -74.012268         40.706718                  N
429
4               -73.972923         40.782520                  N
435
...                    ...               ...                ...
...
1458639         -73.994911         40.740170                  N
778
1458640         -73.970184         40.796547                  N
655
1458641         -74.004433         40.707371                  N
764
1458642         -73.974632         40.757107                  N
373
1458643         -73.972809         40.790585                  N
198

[1458644 rows x 11 columns]
```

Dataset Rows & Columns count

```
# finding no of rows and no of columns in data set
print('no of rows:',df.shape[0])
print('no of columns:',df.shape[1])

no of rows: 1458644
no of columns: 11
```

Step 3 : Data preprocessing

Dataset Information

```
df.info

<bound method DataFrame.info of                  id  vendor_id
pickup_datetime     dropoff_datetime  \
0        id2875421          2  2016-03-14 17:24:55  2016-03-14
17:32:30
1        id2377394          1  2016-06-12 00:43:35  2016-06-12
00:54:38
2        id3858529          2  2016-01-19 11:35:24  2016-01-19
12:10:48
3        id3504673          2  2016-04-06 19:32:31  2016-04-06
19:39:40
4        id2181028          2  2016-03-26 13:30:55  2016-03-26
13:38:10
...            ...        ...                 ...                  ..
.
1458639  id2376096          2  2016-04-08 13:31:04  2016-04-08
13:44:02
1458640  id1049543          1  2016-01-10 07:35:15  2016-01-10
07:46:10
1458641  id2304944          2  2016-04-22 06:57:41  2016-04-22
07:10:25
1458642  id2714485          1  2016-01-05 15:56:26  2016-01-05
16:02:39
1458643  id1209952          1  2016-04-05 14:44:25  2016-04-05
14:47:43

         passenger_count  pickup_longitude  pickup_latitude  \
0                      1        -73.982155        40.767937
1                      1        -73.980415        40.738564
2                      1        -73.979027        40.763939
3                      1        -74.010040        40.719971
4                      1        -73.973053        40.793209
...                  ...               ...              ...
1458639                4        -73.982201        40.745522
1458640                1        -74.000946        40.747379
1458641                1        -73.959129        40.768799
1458642                1        -73.982079        40.749062
1458643                1        -73.979538        40.781750

         dropoff_longitude  dropoff_latitude store_and_fwd_flag
trip_duration
0               -73.964630         40.765602                  N
455
```

```
1                  -73.999481          40.731152                   N
663
2                  -74.005333          40.710087                   N
2124
3                  -74.012268          40.706718                   N
429
4                  -73.972923          40.782520                   N
435
...                       ...                 ...                 ...
...
1458639            -73.994911          40.740170                   N
778
1458640            -73.970184          40.796547                   N
655
1458641            -74.004433          40.707371                   N
764
1458642            -73.974632          40.757107                   N
373
1458643            -73.972809          40.790585                   N
198

[1458644 rows x 11 columns]>
```

Missing Values Analysis

By above operation we know that there is no missing value in our data set.Almost all data type is in their proper fotmat only pickup_date time and dropoff date time in string format which we have to change in their correct format.

Duplicate Values

```
# make a function to check null values and unique values.
def information():
 x=pd.DataFrame(index=df.columns)
 x["data type"]=df.dtypes
 x["null values"]=df.isnull().sum()
 x["unique values"]=df.nunique()
 return x

information()
```

```
                   data type  null values  unique values
id                    object            0        1458644
vendor_id              int64            0              2
pickup_datetime       object            0        1380222
dropoff_datetime      object            0        1380377
```

```
passenger_count        int64           0           10
pickup_longitude     float64           0        23047
pickup_latitude      float64           0        45245
dropoff_longitude    float64           0        33821
dropoff_latitude     float64           0        62519
store_and_fwd_flag    object           0            2
trip_duration          int64           0         7417
```

By above we can see that there is no null value in our data set.

Understanding Your Variables

```
# Dataset Columns
df.columns

Index(['id', 'vendor_id', 'pickup_datetime', 'dropoff_datetime',
       'passenger_count', 'pickup_longitude', 'pickup_latitude',
       'dropoff_longitude', 'dropoff_latitude', 'store_and_fwd_flag',
       'trip_duration'],
      dtype='object')
```

Column Details

Id: A unique identifier for each trip

Vendor Id: A unique identifier for vendor

Pickup Datetime: Date and time of pickup

Dropoff Datetime: Date and time of dropoff

Passenger Count: The number of passengers in the vehicle (driver entered value)

Pickup Longitude: The longitude where the meter was engaged

Pickup Latitude: The latitude where the meter was engaged

Dropoff Longitude: The longitude where the meter was disengaged

Dropoff Latitude: The latitude where the meter was disengaged

Store and Fwd Flag: This flag indicates whether the trip record was held in vehicle memory before sending to the vendor because the vehicle did not have a connection to the server - Y=store and forward; N=not a store and forward trip.

Trip Duration: Duration of time in seconds

Let us finally check for a statistical summary of our dataset.

Note that this function can provide statistics for numerical features only.

```
df.describe()
```

```
          vendor_id  passenger_count  pickup_longitude
pickup_latitude  \
count  1.458644e+06     1.458644e+06      1.458644e+06
1.458644e+06
mean   1.534950e+00     1.664530e+00     -7.397349e+01
4.075092e+01
std    4.987772e-01     1.314242e+00      7.090186e-02     3.288119e-
02
min    1.000000e+00     0.000000e+00     -1.219333e+02
3.435970e+01
25%    1.000000e+00     1.000000e+00     -7.399187e+01
4.073735e+01
50%    2.000000e+00     1.000000e+00     -7.398174e+01
4.075410e+01
75%    2.000000e+00     2.000000e+00     -7.396733e+01
4.076836e+01
max    2.000000e+00     9.000000e+00     -6.133553e+01
5.188108e+01

       dropoff_longitude  dropoff_latitude  trip_duration
count       1.458644e+06      1.458644e+06   1.458644e+06
mean       -7.397342e+01      4.075180e+01   9.594923e+02
std         7.064327e-02      3.589056e-02   5.237432e+03
min        -1.219333e+02      3.218114e+01   1.000000e+00
25%        -7.399133e+01      4.073588e+01   3.970000e+02
50%        -7.397975e+01      4.075452e+01   6.620000e+02
75%        -7.396301e+01      4.076981e+01   1.075000e+03
max        -6.133553e+01      4.392103e+01   3.526282e+06
```

Some insights from the above summary:

- Vendor id has a minimum value of 1 and a maximum value of 2 which makes sense as we saw there are two vendor ids 1 and 2.
- Passenger count has a minimum of 0 which means either it is an error entered or the drivers deliberately entered 0 to complete a target number of rides.

Data Cleaning or Data Wrangling

```
# converting into proper date format
df["pickup_datetime"]=pd.to_datetime(df["pickup_datetime"])
df["dropoff_datetime"]=pd.to_datetime(df["dropoff_datetime"])

df["dropoff_datetime"].dtypes

dtype('<M8[ns]')

# finding pickup and drop month
df["pickup_month"]=df["pickup_datetime"].dt.month
df["dropoff_month"]=df["dropoff_datetime"].dt.month

#finding pickup and drop
df["pickup_date"]=df["pickup_datetime"].dt.day
df["dropoff_date"]=df["dropoff_datetime"].dt.day

# Creating pickup and dropoff weekdays
df['pickup_weekday'] =df['pickup_datetime'].dt.weekday
df['dropoff_weekday']=df['dropoff_datetime'].dt.weekday

# Creating pickup and dropoff hours
df['pickup_hour'] = df['pickup_datetime'].dt.hour
df['dropoff_hour'] =df['dropoff_datetime'].dt.hour

#creating pickup and dropoff day name
df['pickup_day']=df['pickup_datetime'].dt.day_name()
df['dropoff_day']=df['dropoff_datetime'].dt.day_name()

df.head()

          id  vendor_id     pickup_datetime    dropoff_datetime  \
0  id2875421          2 2016-03-14 17:24:55 2016-03-14 17:32:30
1  id2377394          1 2016-06-12 00:43:35 2016-06-12 00:54:38
2  id3858529          2 2016-01-19 11:35:24 2016-01-19 12:10:48
3  id3504673          2 2016-04-06 19:32:31 2016-04-06 19:39:40
4  id2181028          2 2016-03-26 13:30:55 2016-03-26 13:38:10

   passenger_count  pickup_longitude  pickup_latitude
dropoff_longitude  \
0                1        -73.982155        40.767937         -
73.964630
1                1        -73.980415        40.738564         -
73.999481
2                1        -73.979027        40.763939         -
74.005333
3                1        -74.010040        40.719971         -
74.012268
4                1        -73.973053        40.793209         -
73.972923
```

```
   dropoff_latitude store_and_fwd_flag  ...  pickup_month
dropoff_month  \
0         40.765602                  N  ...             3
3
1         40.731152                  N  ...             6
6
2         40.710087                  N  ...             1
1
3         40.706718                  N  ...             4
4
4         40.782520                  N  ...             3
3

   pickup_date  dropoff_date  pickup_weekday  dropoff_weekday
pickup_hour  \
0           14            14               0                0
17
1           12            12               6                6
0
2           19            19               1                1
11
3            6             6               2                2
19
4           26            26               5                5
13

   dropoff_hour  pickup_day dropoff_day
0            17      Monday      Monday
1             0      Sunday      Sunday
2            12     Tuesday     Tuesday
3            19   Wednesday   Wednesday
4            13    Saturday    Saturday

[5 rows x 21 columns]
```

```
# calculate trip duration in minute
df["trip_duration_in_minute"]=df["trip_duration"]/60

# calculate the distance by given geospatial co ordinate in kilometer
from geopy.distance import great_circle

df['distance'] = df.apply(lambda row:
great_circle((row['pickup_latitude'], row["pickup_longitude"]),
(row['dropoff_latitude'], row['dropoff_longitude'])).kilometers,
axis=1)
```

```
----------------------------------------------------------------------
-----
ModuleNotFoundError                          Traceback (most recent call
```

```
last)
Cell In[21], line 2
      1 # calculate the distance by given geospatial co ordinate in
kilometer
----> 2 from geopy.distance import great_circle
      4 df['distance'] = df.apply(lambda row:
great_circle((row['pickup_latitude'], row["pickup_longitude"]),
(row['dropoff_latitude'], row['dropoff_longitude'])).kilometers,
axis=1)

ModuleNotFoundError: No module named 'geopy'
```

```
# calculate the distance by given geospatial co ordinate in kilometer
from geopy.distance import great_circle

df['distance'] = df.apply(lambda row:
great_circle((row['pickup_latitude'], row["pickup_longitude"]),
(row['dropoff_latitude'], row['dropoff_longitude'])).kilometers,
axis=1)
```

```
-----------------------------------------------------------------------
-----
ModuleNotFoundError                       Traceback (most recent call
last)
Cell In[22], line 2
      1 # calculate the distance by given geospatial co ordinate in
kilometer
----> 2 from geopy.distance import great_circle
      4 df['distance'] = df.apply(lambda row:
great_circle((row['pickup_latitude'], row["pickup_longitude"]),
(row['dropoff_latitude'], row['dropoff_longitude'])).kilometers,
axis=1)

ModuleNotFoundError: No module named 'geopy'
```

Step 4 : Data Vizualization, Storytelling & Experimenting with charts : Understand the relationships between variables

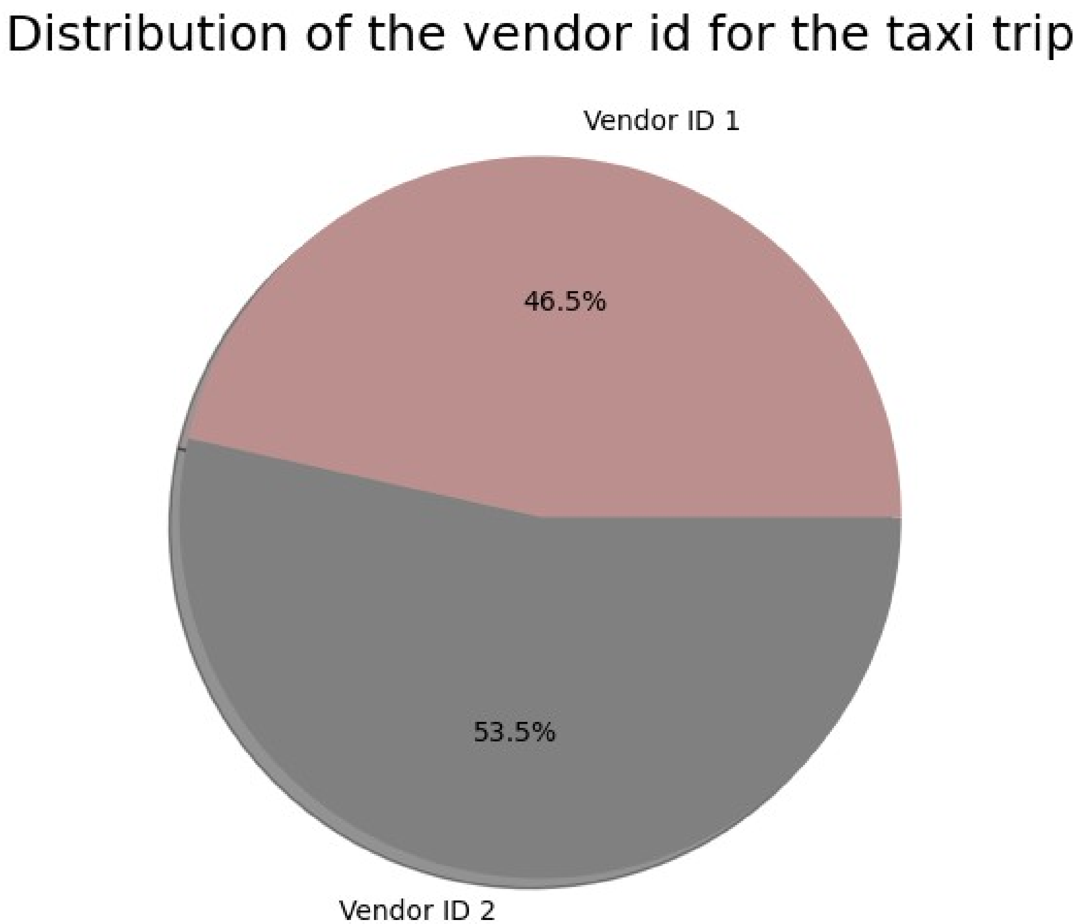
Chart-1

```
# percentage of trip by vendor
percentage_of_vend_1=round(len(df[df["vendor_id"]==1])/len(df)*100,1)
percentage_of_vend_2=round(len(df[df["vendor_id"]==2])/len(df)*100,1)
```

```
total_percentage=[percentage_of_vend_1,percentage_of_vend_2]
total_percentage
```

```
[46.5, 53.5]
```

```
plt.figure(figsize = (6,8))
c=['rosybrown','gray']
plt.pie(total_percentage, labels = ['Vendor ID 1','Vendor ID 2'],autopct='%.1f%%',colors=c,shadow=True)
plt.title('Distribution of the vendor id for the taxi trip',fontsize=18)
plt.show()
```



We can observe that vendor 2 has a higher number of bookings (54%).

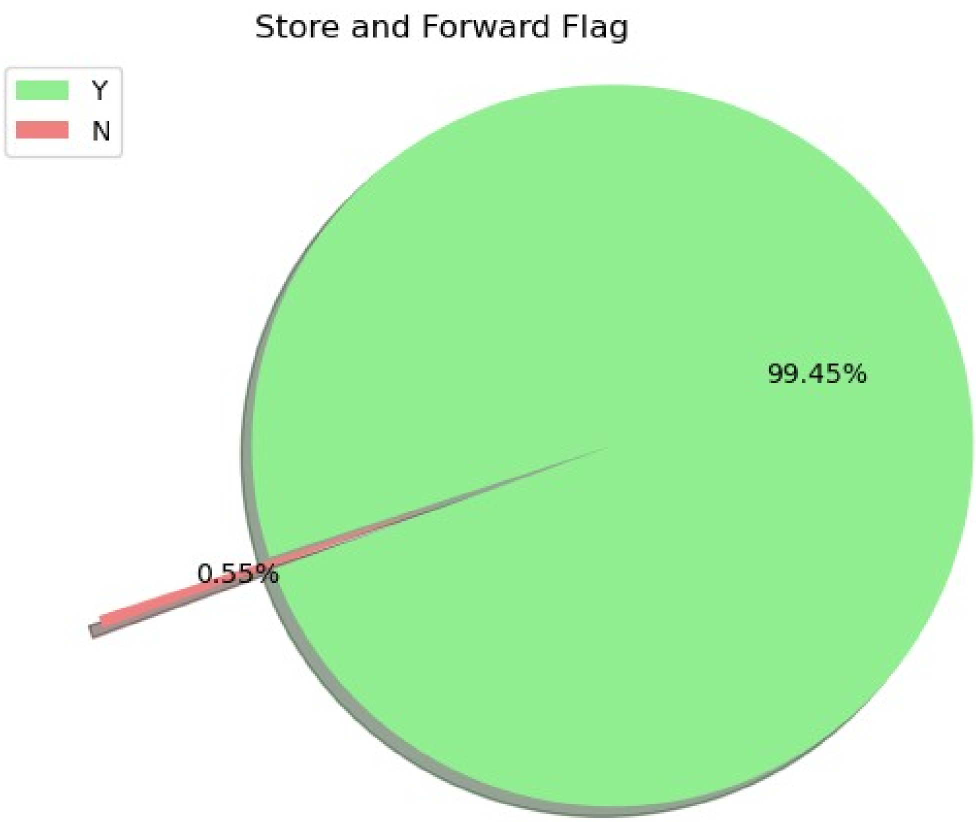
Chart-2

```
df["store_and_fwd_flag"].value_counts()
```

```
N    1450599
Y       8045
Name: store_and_fwd_flag, dtype: int64
```

```
#Store & Forward flag

plt.figure(figsize=(6,8))
plt.pie(df['store_and_fwd_flag'].value_counts(), colors=['lightgreen',
'lightcoral'], shadow=True, explode=[0.5,0], autopct='%1.2f%%',
startangle=200)
plt.legend(labels=['Y','N'])
plt.title("Store and Forward Flag")
```

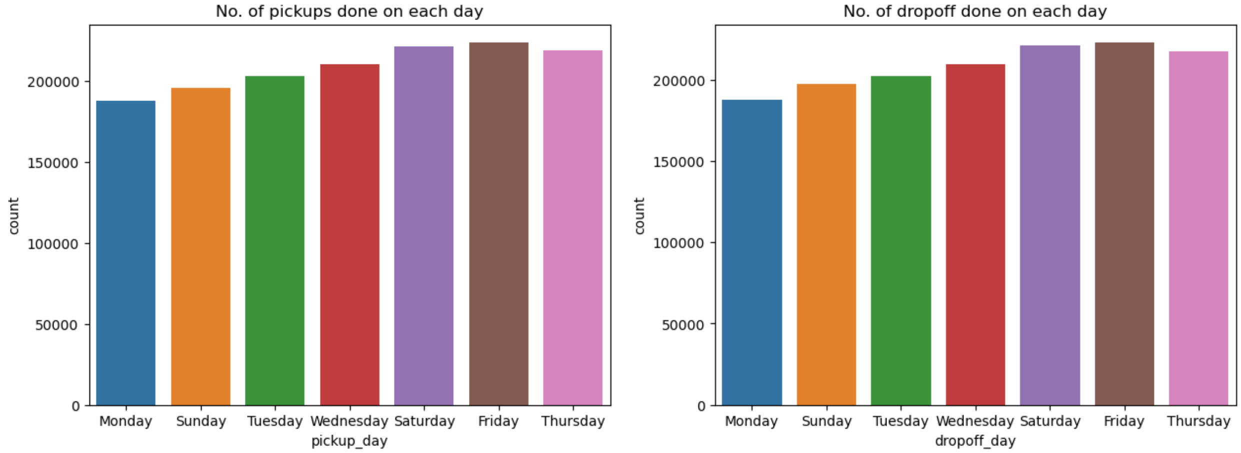
```
Text(0.5, 1.0, 'Store and Forward Flag')
```



We see there are less than 1% of trips that were stored before forwarding.

Chart-3

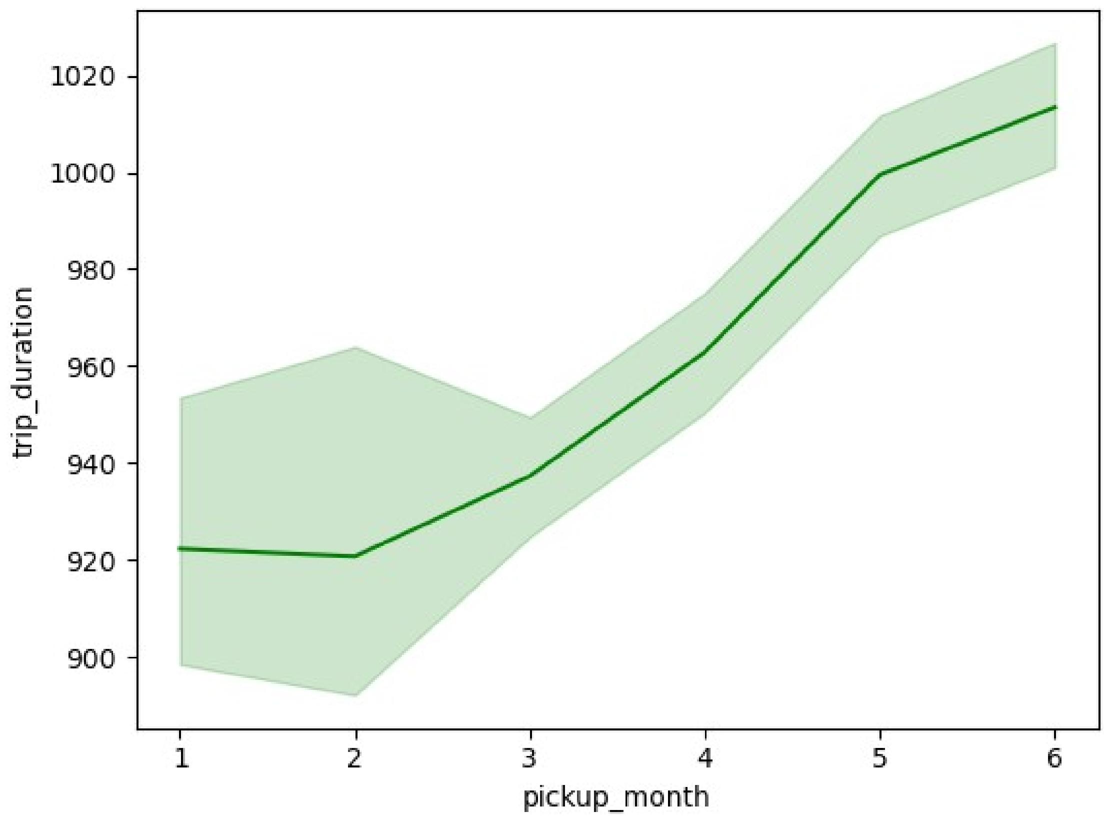
```
##Number of Pickups and Dropoff on each day of the week
figure,ax=plt.subplots(nrows=1,ncols=2,figsize=(15,5))
sns.countplot(x="pickup_day",data=df,ax=ax[0])
ax[0].set_title('No. of pickups done on each day')
sns.countplot(x="dropoff_day",data=df,ax=ax[1])
ax[1].set_title('No. of dropoff done on each day')
plt.show()
```



- Above plots interpret that in a week, "friday", and "saturday" have higher number of pickups and dropoffs.
- We can see that compared to other days, taxi booking rates are higher on the weekends (Friday and Saturday).This suggests that individuals used to go out on weekends for their celebrations, parties, or even other personnel work.

Chart-4

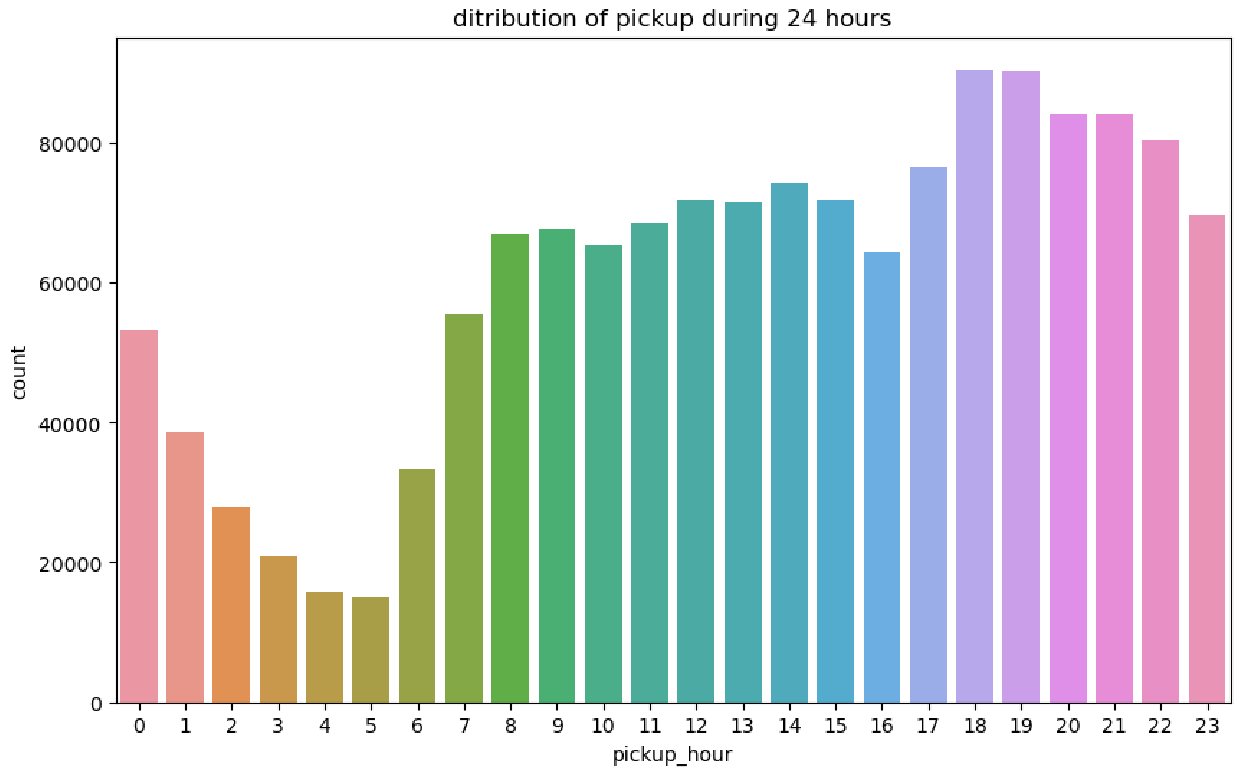
```
#Trip Duration by the month.
sns.lineplot(x='pickup_month',y='trip_duration',data=df,color='green')
plt.show()
```



* From February, we can see trip duration rising every month.

Chart-5

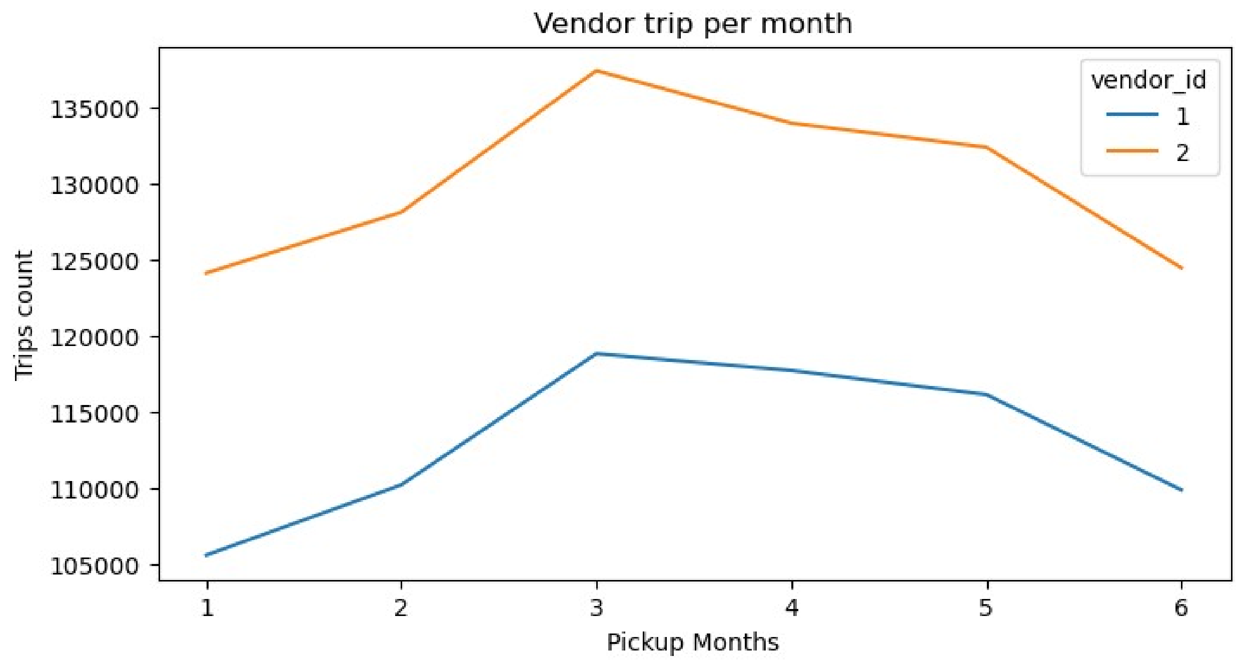
```
# distribution of ride in complete 24 hours hourly basis
plt.figure(figsize=(10,6))
sns.countplot(x=df["pickup_hour"])
plt.title("ditribution of pickup during 24 hours")
plt.show()
```



- Distribution of pickup and dropoff hours follows same pattern, it shows that most of the pickups and dropoffs are in the evening. We can see that people often use taxi services to get to their workplaces in the mornings after 10:00 AM. and busiet time is 6PM to 7PM.

Chart-6

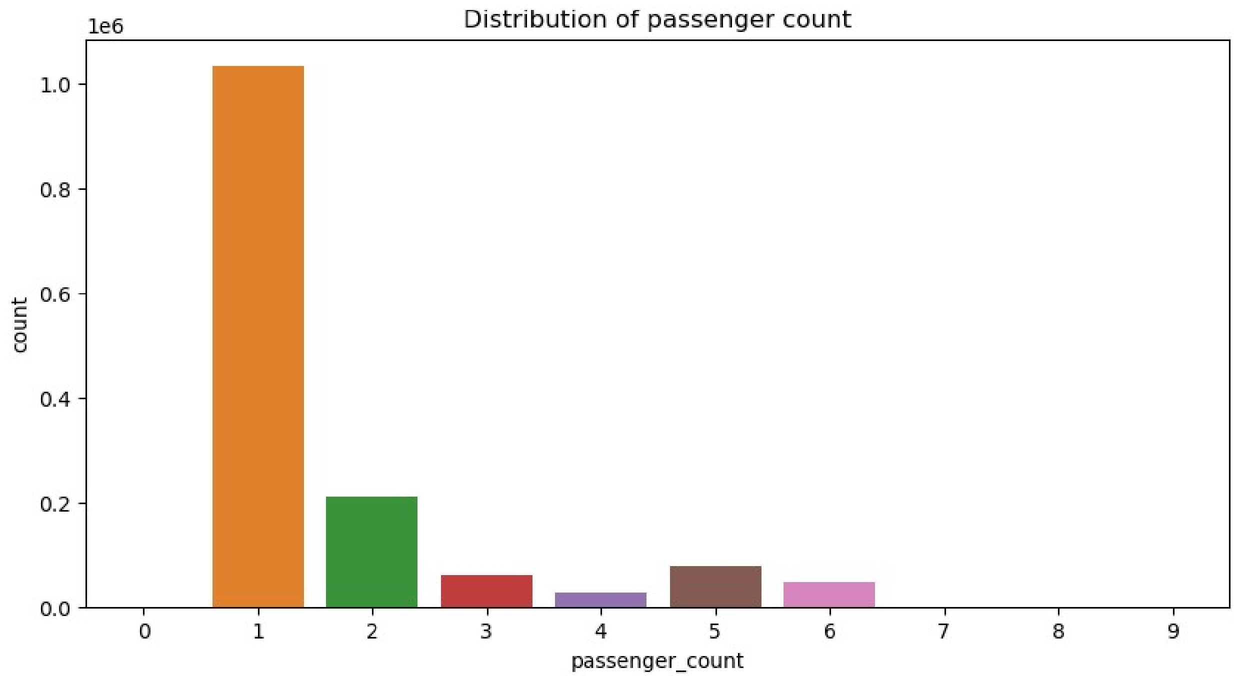
```
#aggegate vendor id by pickup month
monthly_pickup_by_vendor=df.groupby(["pickup_month","vendor_id"]).size
()
monthly_pickup_by_vendor = monthly_pickup_by_vendor.unstack()

monthly_pickup_by_vendor.plot(kind = 'line', figsize = (8,4))
plt.title('Vendor trip per month')
plt.xlabel('Pickup Months')
plt.ylabel('Trips count')
plt.show()
```



- We can see that both vendors' trips are at their maximum in the month of March and their lowest in the month of January, February, and after June.

Chart-7

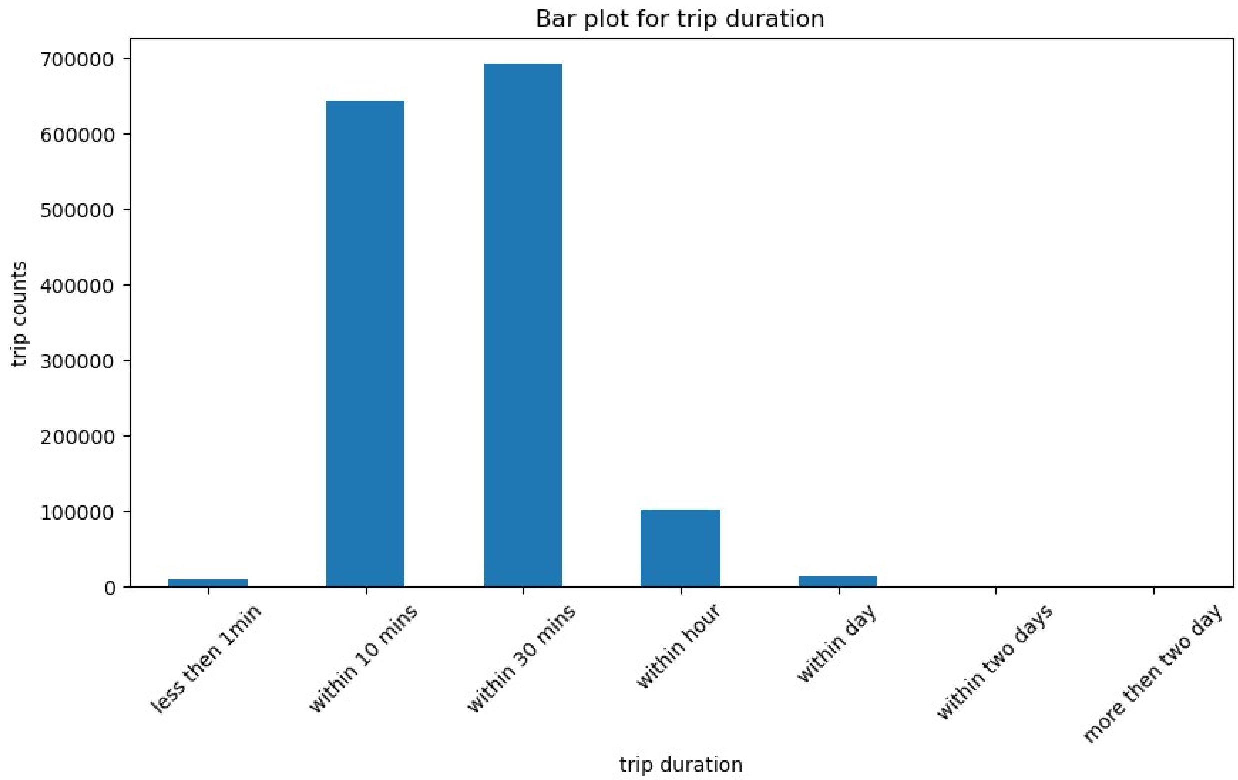
```
#Passenger Count
df.passenger_count.value_counts()

1    1033540
2     210318
5      78088
3      59896
6      48333
4      28404
0         60
7          3
9          1
8          1
Name: passenger_count, dtype: int64

# distribution of passenger
plt.figure(figsize=(10,5))
sns.countplot(x=df["passenger_count"])
plt.title('Distribution of passenger count')
plt.show()
```



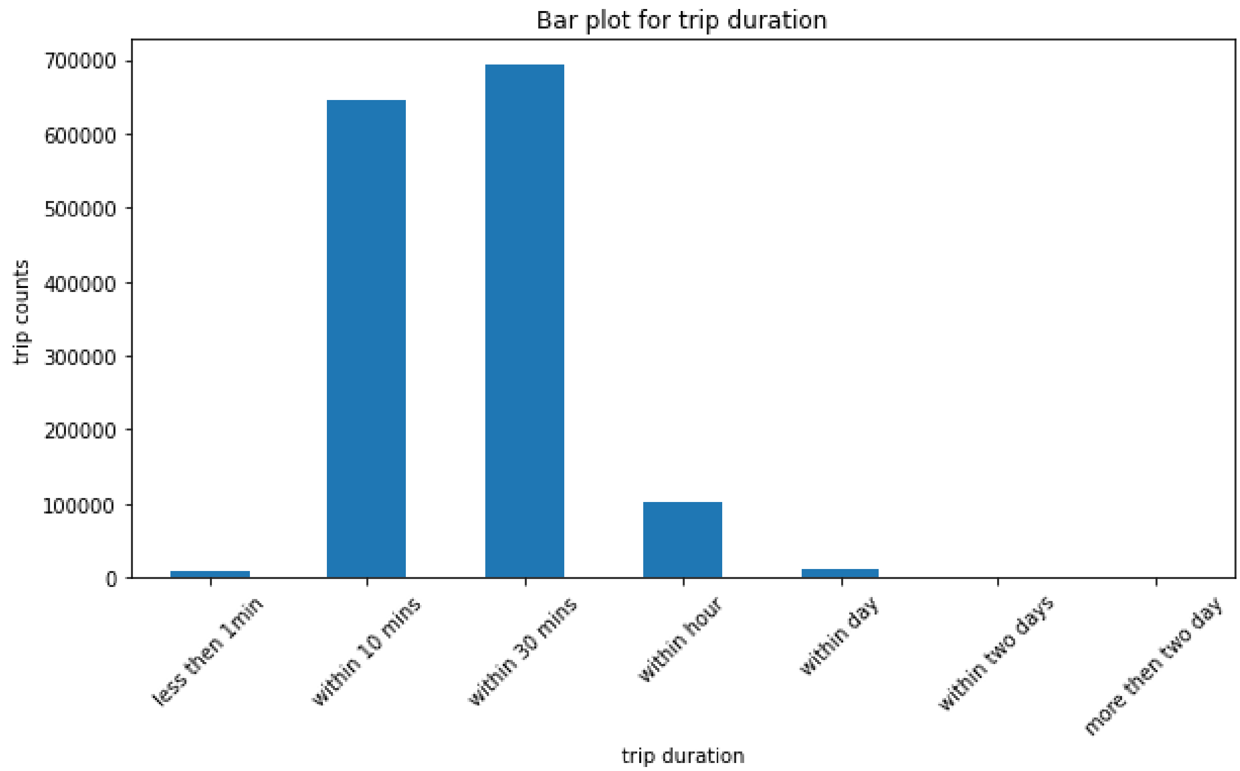
- We can notice that most of the bookings are made by solo traveler.which means less number of people prefer car pool or may be less number of groups book car...people prefer to ride solo

Chart-8

```
# divide trip duration in differnt bins
labels=['less then 1min','within 10 mins','within 30 mins','within hour','within day','within two days','more then two day']

plt.figure(figsize=[10,5])
df1=pd.cut(df['trip_duration_in_minute'],bins=[0,1,10,30,60,1440,1440*2,50000],labels=labels)
df.groupby(df1)['trip_duration_in_minute'].count().plot(kind='bar')
plt.title("Bar plot for trip duration")
plt.ylabel("trip counts")
plt.xlabel("trip duration")
plt.xticks(rotation=45)
plt.show()
```



- By above chart we can see that most of trip duration 10 to 30 minute. some trip also goes on hourly.long trip with in day very rare.

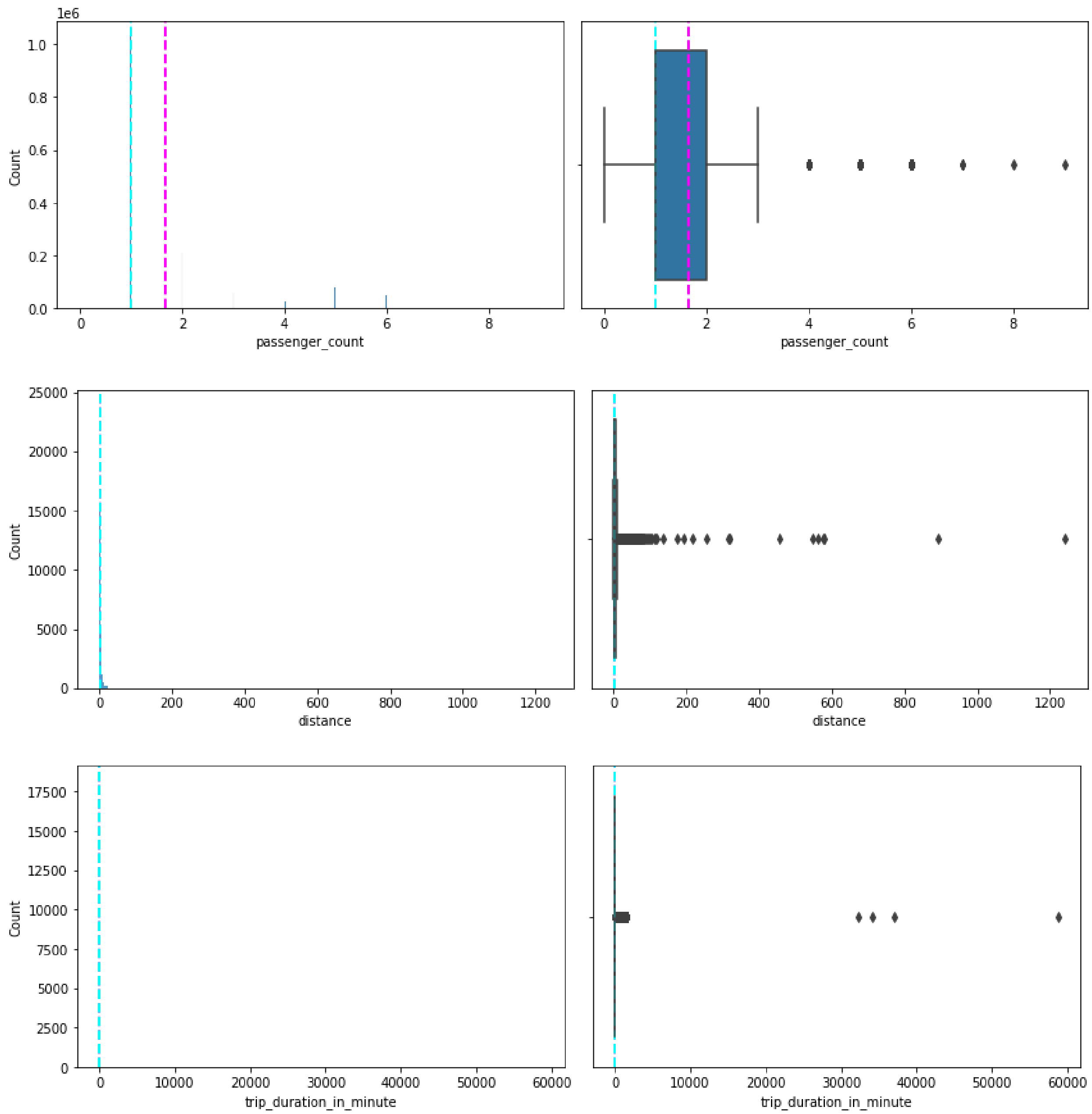


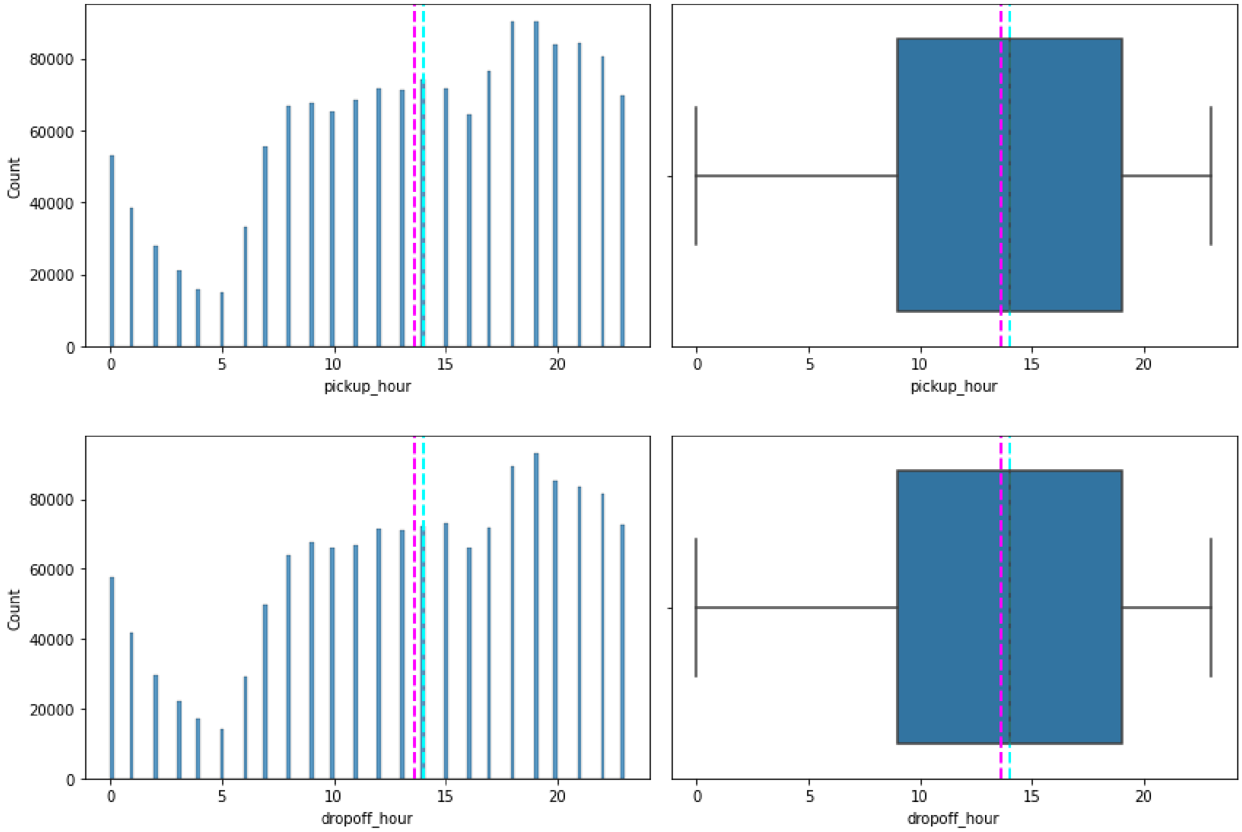
- By above chart we can see that most of trip duration 10 to 30 minute. some trip also goes on hourly.long trip with in day very rare.

#Distribution of differnt features

```
# Histplots and boxplots to determine distribution the data given
below
numeric_feature=['passenger_count','distance','trip_duration_in_minute
','pickup_hour', 'dropoff_hour']
numeric_feature

['passenger_count',
 'distance',
 'trip_duration_in_minute',
 'pickup_hour',
 'dropoff_hour']

for col in numeric_feature:
  fig,ax=plt.subplots(nrows=1,ncols=2,figsize=(12,4))
  sns.histplot(data=df,x=col,ax=ax[0])
  ax[0].axvline(df[col].mean(), color='magenta', linestyle='dashed',
linewidth=2)
  ax[0].axvline(df[col].median(), color='cyan', linestyle='dashed',
linewidth=2)
  sns.boxplot(data=df, x=col, ax=ax[1])
  ax[1].axvline(df[col].mean(), color='magenta', linestyle='dashed',
linewidth=2)
```

```
  ax[1].axvline(df[col].median(), color='cyan', linestyle='dashed', linewidth=2)
  plt.tight_layout()
```





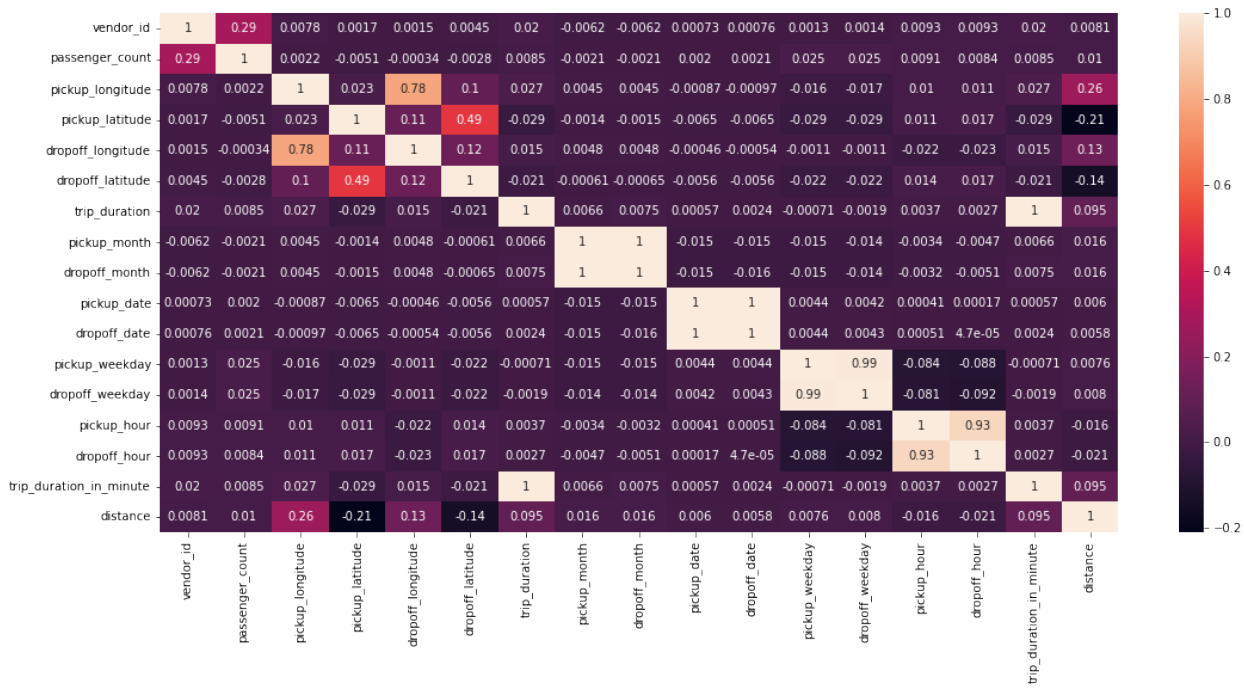
##(histplot) distance and trip_duration graphs are highly skewed.

##(boxplot) distance and trip_duration columns have a lot outliers as well

#Multicollinearity and correlation check

#**Heatmap**

```
plt.figure(figsize=(18,8))
correlation=df.corr()
sns.heatmap(correlation,annot=True)
plt.show()
```



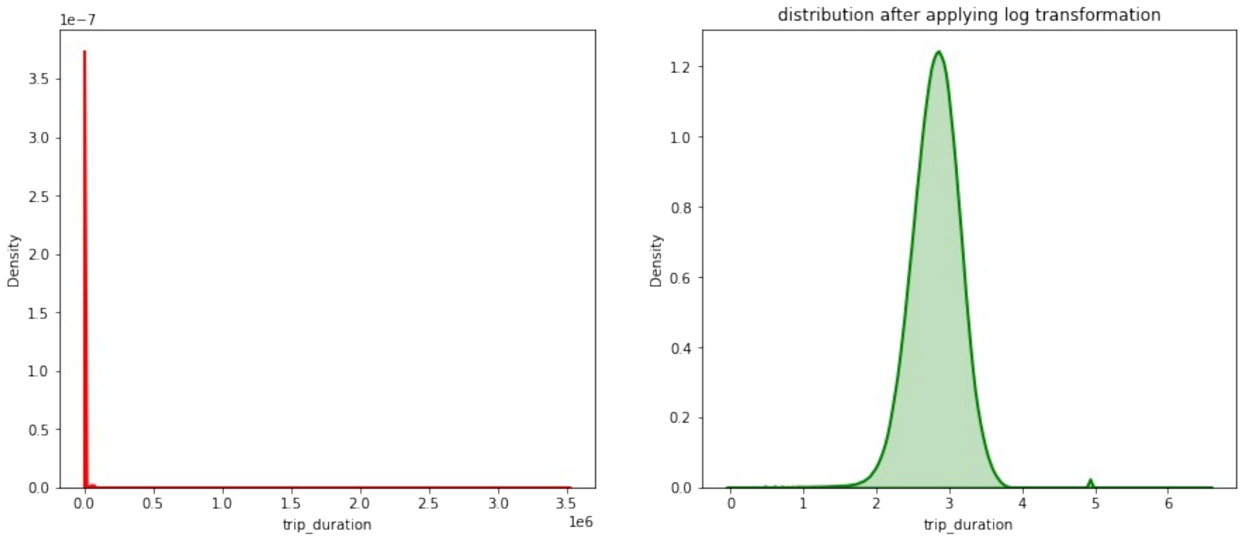
- By above haetmap it visulaize that pickup_month and dropp off month is 100% correlated.Along with pickup hour ,dropoff hour,pickup weekday and dropoff week day,trip duration and trip duration in minute are highly correlated.

```
def correlated (dataset,thresold):
  corr_column=set()    # all the highly corelated column
  for i in range(len(correlation.columns)):
      for j in range(i):
        if abs(correlation.iloc[i,j])>=thresold:  # we want absolute value
          column_name=correlation.columns[i]      # getting the name of columns
          corr_column.add(column_name)            # add he name column in empty set
  return corr_column

# Calling the function with threshold value 0.90
highly_correlated_features=correlated(df,0.90)
print('total highly correlated features:',len(set(highly_correlated_features)))

total highly correlated features: 5

highly_correlated_features

{'dropoff_date',
 'dropoff_hour',
 'dropoff_month',
 'dropoff_weekday',
 'trip_duration_in_minute'}
```

- by above evaluation we can say that there are four column they are highly correlated above 90%.
- it better to drop higly correlated features for better performance.

#checking skewness of target variable

```
# dist plot of trip duration.
fig,ax=plt.subplots(nrows=1,ncols=2,figsize=(15,6))
sns.distplot(df.trip_duration,color='red',ax=ax[0],hist=False,kde_kws=
{'shade':True, 'linewidth':2})
sns.distplot(np.log10(df["trip_duration"]),color='green',ax=ax[1],hist
= False,kde= True,kde_kws= {'shade':True, 'linewidth':2})
ax[1].set_title("distribution after applying log transformation")

Text(0.5, 1.0, 'distribution after applying log transformation')
```



- BY above distribution we can see that target variable is higly right skewed .to remove the skewness we apply log transformation.after transformation we found normal distribution of targer variable.

#Outlier Removal (Quartile Method)

Interquartile range measures the spread of the middle half of our data.

Formula: Q3 - Q1

where Q1- quartile 1 and Q3- quartile 3

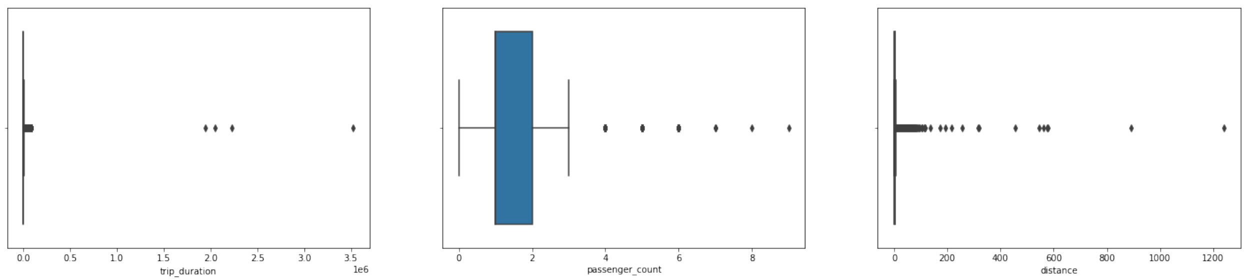
lower limit of the data is given by Q1-1.5*IQR

upper limit of the data is given by Q3+1.5*IQR

```
#boxplot for visualizing for outliers
fig,ax=plt.subplots(nrows=1,ncols=3,figsize=(25,5))
```

```
sns.boxplot(df["trip_duration"],ax=ax[0])
sns.boxplot(df['passenger_count'],ax=ax[1])
sns.boxplot(df['distance'],ax=ax[2])
```

```
<AxesSubplot:xlabel='distance'>
```




```
#finding differnt quarters of trip_duration column
trip_duration_Q1=df['trip_duration'].quantile(0.25)
print('first quartile value ie 25th percentile of trip duration:',trip_duration_Q1)
trip_duration_Q2=df['trip_duration'].quantile(0.50)
print('second quartile value ie 50th percentile of trip duration:',trip_duration_Q2)
trip_duration_Q3=df['trip_duration'].quantile(0.75)
print('third quartile value ie 75th percentile of trip duration:',trip_duration_Q3)
```

```
first quartile value ie 25th percentile of trip duration: 397.0
second quartile value ie 50th percentile of trip duration: 662.0
third quartile value ie 75th percentile of trip duration: 1075.0
```

```
# calculate interquartile range
IQR=trip_duration_Q3-trip_duration_Q1
print('IQR:',IQR)
trip_duration_lower_limit=trip_duration_Q1-1.5*IQR
trip_duration_upper_limit=trip_duration_Q3+1.5*IQR
print('The lower limit of trip duration:',trip_duration_lower_limit)
print('The upper limit of trip duration:',trip_duration_upper_limit)
```

```
IQR: 678.0
The lower limit of trip duration: -620.0
The upper limit of trip duration: 2092.0
```

```
#removing outliers in trip_duration features
df=df[df['trip_duration']>0]
df=df[df['trip_duration']<trip_duration_upper_limit]
```

```
df.shape
```

```
(1384320, 23)
```

```
#finding differnt quarters of passenger_count column
passenger_count_Q1=df['passenger_count'].quantile(0.25)
print('first quartile value ie 25th percentile of passenger count:',passenger_count_Q1)
passenger_count_Q2=df['passenger_count'].quantile(0.50)
print('second quartile value ie 50th percentile of passenger count:',passenger_count_Q2)
passenger_count_Q3=df['passenger_count'].quantile(0.75)
print('third quartile value ie 75th percentile of passenger count:',passenger_count_Q3)
```

```
first quartile value ie 25th percentile of passenger count: 1.0
second quartile value ie 50th percentile of passenger count: 1.0
third quartile value ie 75th percentile of passenger count: 2.0
```

```
# Calculating IQR
IQR= passenger_count_Q3 - passenger_count_Q1
passenger_count_lower_limit=passenger_count_Q1 - 1.5*IQR
passenger_count_upper_limit=passenger_count_Q3 + 1.5*IQR
print("The lower limit of passenger count:", passenger_count_lower_limit)
print("The upper limit of passenger count:", passenger_count_upper_limit)
```

```
The lower limit of passenger count: -0.5
The upper limit of passenger count: 3.5
```

```
# Removing outliers
df=df[df['passenger_count']>0]
df=df[df['passenger_count']<passenger_count_upper_limit]
```

```
df.shape
```

```
(1237987, 23)
```

```
#finding differnt quarters of distance column
distance_Q1=df['distance'].quantile(0.25)
print('first quartile value ie 25th percentile of distance:',distance_Q1)
distance_Q2=df['distance'].quantile(0.50)
print('second quartile value ie 50th percentile of distance:',distance_Q2)
distance_Q3=df['distance'].quantile(0.75)
print('third quartile value ie 75th percentile of distance:',distance_Q3)
```

```
first quartile value ie 25th percentile of distance: 1.197497120120381
second quartile value ie 50th percentile of distance: 1.9919619004442215
third quartile value ie 75th percentile of distance: 3.4835674136716936
```

```
# Calculating IQR
IQR= distance_Q3 - distance_Q1
distance_lower_limit=distance_Q1 - 1.5*IQR
distance_upper_limit=distance_Q3 + 1.5*IQR
print("The lower limit of distance:", distance_lower_limit)
print("The upper limit of distance:", distance_upper_limit)
```

```
The lower limit of distance: -2.231608320206588
The upper limit of distance: 6.912672853998663
```

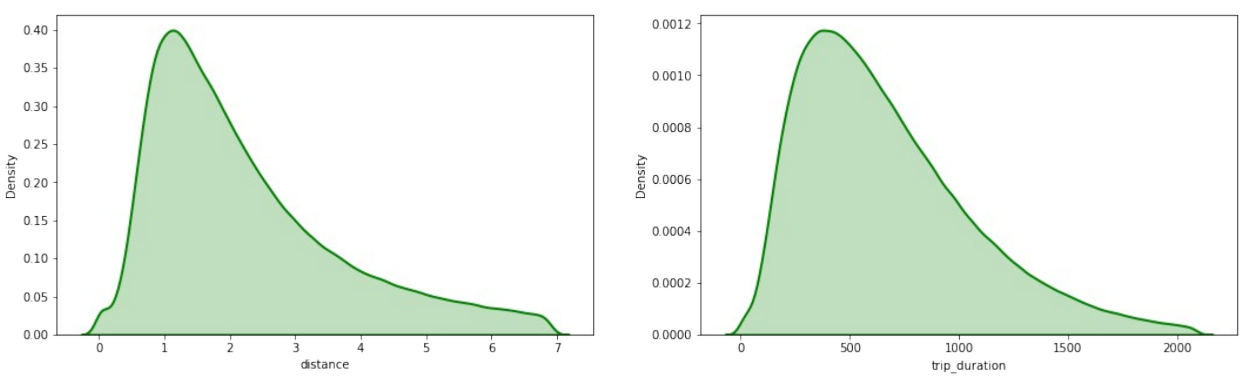
```
# Removing outliers
df=df[df['distance']>0]
df=df[df['distance']<distance_upper_limit]
```

```
df.shape
```

```
(1136749, 23)
```

```
# Earlier we saw that distance and tripduration had highly skewed graph... lets check the distribution again
figure, ax = plt.subplots(nrows = 1, ncols = 2, figsize = (18,5))
sns.distplot(df['distance'], hist=False, kde=True, kde_kws={'shade':True, 'linewidth':2}, color="green", ax=ax[0])
sns.distplot(df['trip_duration'], hist=False, kde=True, kde_kws={'shade':True, 'linewidth':2}, color="green", ax=ax[1])
```

```
<AxesSubplot:xlabel='trip_duration', ylabel='Density'>
```



##It seems both the columns now follow near to normal distribution

#**Textual Data Preprocessing**

#ONE HOT ENCODING

###Since we have textual data in our dataset which might create problems during model prediction, therefore we need to convert this textual data into dummy variables

```
#add dummy variable to convert textual data to numerical data through one hot encoding
```

```
df=pd.get_dummies(df,columns=['store_and_fwd_flag', 'pickup_weekday',
'dropoff_weekday'],drop_first=True)

df.shape

(1136749, 33)
```

#Instead of dropping irrelevant or collinear columns we will be creating a separate list containg only those variables that are important and are not collinear

##(dropoff_date', 'dropoff_hour', 'dropoff_month', 'dropoff_weekday', 'trip_duration_minute are highly correlated according to the heatmap.)

```
features=['vendor_id', 'passenger_count', 'distance',
'pickup_longitude', 'pickup_latitude',
       'dropoff_longitude', 'dropoff_latitude',
'store_and_fwd_flag_Y','pickup_weekday_1',
          'pickup_weekday_2', 'pickup_weekday_3', 'pickup_weekday_4',
'pickup_weekday_5', 'pickup_weekday_6']

final_df=df[features]
final_df.shape

(1136749, 14)
```

#STEP 5- Data Modelling

#Evaluating which model is better. Therefore we will be calculating evaluation metrics for different models

```
# define a  function to calculate evaluation metrics
def evaluation_metrics (x_train,y_train,y_predicted):
  MSE=round(mean_squared_error(y_true=y_train, y_pred=y_predicted),4)
  RMSE=math.sqrt(MSE)
  R2_score=r2_score(y_true=y_train, y_pred=y_predicted)
  Adjusted_R2_score=1-((1-( R2_score))*(x_train.shape[0]-
1)/(x_train.shape[0]-x_train.shape[1]-1))

  print("Mean Squared Error:",MSE,"Root Mean Squared Error:", RMSE)
  print("R2 Score :",R2_score,"Adjusted R2 Score :",Adjusted_R2_score)
  # plotting actual and predicted values
  #Plotting Actual and Predicted Values
  plt.figure(figsize=(18,6))
  plt.plot((y_predicted)[:100], color='red')
  plt.plot(np.array(y_train)[:100], color='green')
  plt.legend(["Predicted","Actual"])
  plt.title('Actual and Predicted Time Duration')

  #return(MSE,RMSE,R2_score,Adjusted_R2_score)
```

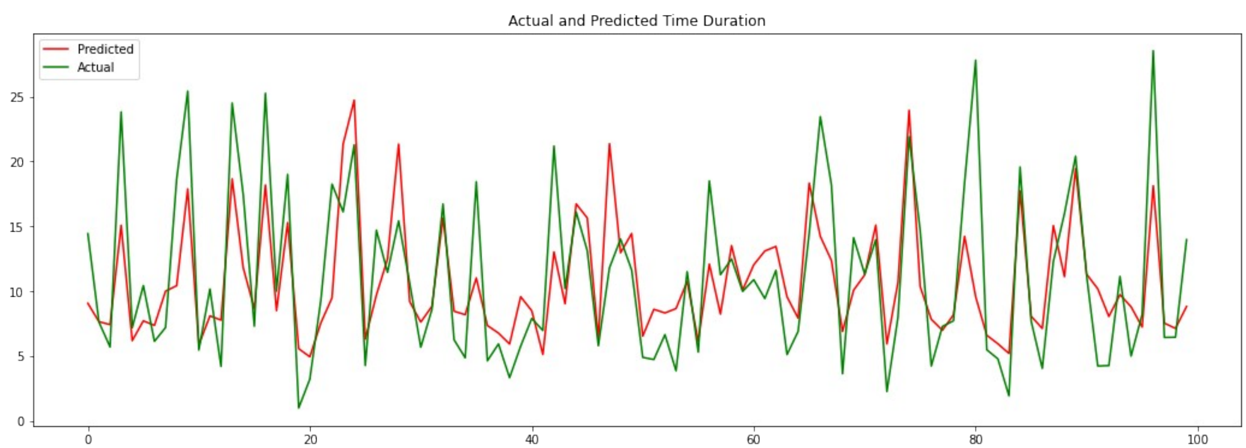
```
x=final_df[features]
y=df["trip_duration_in_minute"]
```

#**Train and Test Model**

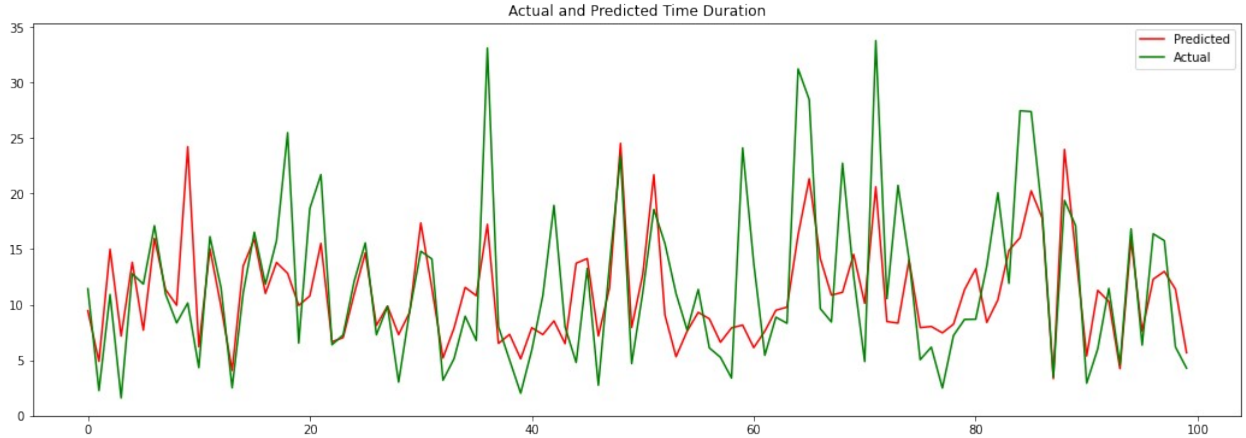
```
# Importing train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,rando
m_state=42)
```

#**Model 1 - Linear Regression**

```
lr=LinearRegression()
lr.fit(x_train,y_train)
a=lr.score(x_train, y_train)
y_pred_train = lr.predict(x_train)
y_pred_test = lr.predict(x_test)

# evaluation metrics for train data set
evaluation_metrics(x_train,y_train,y_pred_train)

Mean Squared Error: 22.5249 Root Mean Squared Error: 4.746040454947682
R2 Score : 0.48710009447510005 Adjusted R2 Score : 0.4870921983622618
```




```
# evaluation metrics for test data set
evaluation_metrics(x_test,y_test,y_pred_test)

Mean Squared Error: 22.4453 Root Mean Squared Error: 4.737647095341737
R2 Score : 0.48885358400792234 Adjusted R2 Score : 0.4888221060136677
```



#Inference - As we can see that R2 score is very less and MSE is pretty high which means this algorithm is not suitable for our model

#**Model 2 - Decision Tree**

```
# Maximum depth of trees
max_depth = [4,6,8,10,12]

# Minimum number of samples required to split a node
min_samples_split = [10,20,30]

# Minimum number of samples required at each leaf node
min_samples_leaf = [6,10,16,20]

# Hyperparameter Grid
param_decision_tree = {
              'max_depth' : max_depth,
              'min_samples_split' : min_samples_split,
              'min_samples_leaf' : min_samples_leaf}
DTR = DecisionTreeRegressor()

# Grid search
decision_tree_grid = GridSearchCV(estimator=DTR,
                         param_grid = param_decision_tree,
                         cv = 5, verbose=2, scoring='r2')
decision_tree_grid.fit(x_train,y_train)

Fitting 5 folds for each of 60 candidates, totalling 300 fits
[CV] END max_depth=4, min_samples_leaf=6, min_samples_split=10; total
time=   3.1s
[CV] END max_depth=4, min_samples_leaf=6, min_samples_split=10; total
time=   2.7s
[CV] END max_depth=4, min_samples_leaf=6, min_samples_split=10; total
time=   2.7s
[CV] END max_depth=4, min_samples_leaf=6, min_samples_split=10; total
time=   2.7s
[CV] END max_depth=4, min_samples_leaf=6, min_samples_split=10; total
```

```
time=   3.7s
[CV] END max_depth=4, min_samples_leaf=6, min_samples_split=20; total
time=   2.7s
[CV] END max_depth=4, min_samples_leaf=6, min_samples_split=20; total
time=   2.7s
[CV] END max_depth=4, min_samples_leaf=6, min_samples_split=20; total
time=   2.7s
[CV] END max_depth=4, min_samples_leaf=6, min_samples_split=20; total
time=   2.8s
[CV] END max_depth=4, min_samples_leaf=6, min_samples_split=20; total
time=   3.7s
[CV] END max_depth=4, min_samples_leaf=6, min_samples_split=30; total
time=   2.6s
[CV] END max_depth=4, min_samples_leaf=6, min_samples_split=30; total
time=   2.7s
[CV] END max_depth=4, min_samples_leaf=6, min_samples_split=30; total
time=   2.7s
[CV] END max_depth=4, min_samples_leaf=6, min_samples_split=30; total
time=   3.1s
[CV] END max_depth=4, min_samples_leaf=6, min_samples_split=30; total
time=   3.5s
[CV] END max_depth=4, min_samples_leaf=10, min_samples_split=10; total
time=   2.7s
[CV] END max_depth=4, min_samples_leaf=10, min_samples_split=10; total
time=   2.7s
[CV] END max_depth=4, min_samples_leaf=10, min_samples_split=10; total
time=   2.7s
[CV] END max_depth=4, min_samples_leaf=10, min_samples_split=10; total
time=   3.5s
[CV] END max_depth=4, min_samples_leaf=10, min_samples_split=10; total
time=   3.1s
[CV] END max_depth=4, min_samples_leaf=10, min_samples_split=20; total
time=   2.7s
[CV] END max_depth=4, min_samples_leaf=10, min_samples_split=20; total
time=   2.7s
[CV] END max_depth=4, min_samples_leaf=10, min_samples_split=20; total
time=   2.7s
[CV] END max_depth=4, min_samples_leaf=10, min_samples_split=20; total
time=   3.9s
[CV] END max_depth=4, min_samples_leaf=10, min_samples_split=20; total
time=   2.7s
[CV] END max_depth=4, min_samples_leaf=10, min_samples_split=30; total
time=   2.6s
[CV] END max_depth=4, min_samples_leaf=10, min_samples_split=30; total
time=   2.6s
[CV] END max_depth=4, min_samples_leaf=10, min_samples_split=30; total
time=   2.9s
[CV] END max_depth=4, min_samples_leaf=10, min_samples_split=30; total
time=   3.6s
```

```
[CV] END max_depth=4, min_samples_leaf=10, min_samples_split=30; total
time=   2.6s
[CV] END max_depth=4, min_samples_leaf=16, min_samples_split=10; total
time=   2.6s
[CV] END max_depth=4, min_samples_leaf=16, min_samples_split=10; total
time=   2.7s
[CV] END max_depth=4, min_samples_leaf=16, min_samples_split=10; total
time=   3.2s
[CV] END max_depth=4, min_samples_leaf=16, min_samples_split=10; total
time=   3.3s
[CV] END max_depth=4, min_samples_leaf=16, min_samples_split=10; total
time=   2.7s
[CV] END max_depth=4, min_samples_leaf=16, min_samples_split=20; total
time=   2.7s
[CV] END max_depth=4, min_samples_leaf=16, min_samples_split=20; total
time=   2.7s
[CV] END max_depth=4, min_samples_leaf=16, min_samples_split=20; total
time=   3.6s
[CV] END max_depth=4, min_samples_leaf=16, min_samples_split=20; total
time=   3.0s
[CV] END max_depth=4, min_samples_leaf=16, min_samples_split=20; total
time=   2.7s
[CV] END max_depth=4, min_samples_leaf=16, min_samples_split=30; total
time=   2.7s
[CV] END max_depth=4, min_samples_leaf=16, min_samples_split=30; total
time=   2.7s
[CV] END max_depth=4, min_samples_leaf=16, min_samples_split=30; total
time=   3.8s
[CV] END max_depth=4, min_samples_leaf=16, min_samples_split=30; total
time=   2.7s
[CV] END max_depth=4, min_samples_leaf=16, min_samples_split=30; total
time=   2.7s
[CV] END max_depth=4, min_samples_leaf=20, min_samples_split=10; total
time=   2.7s
[CV] END max_depth=4, min_samples_leaf=20, min_samples_split=10; total
time=   3.1s
[CV] END max_depth=4, min_samples_leaf=20, min_samples_split=10; total
time=   3.4s
[CV] END max_depth=4, min_samples_leaf=20, min_samples_split=10; total
time=   2.7s
[CV] END max_depth=4, min_samples_leaf=20, min_samples_split=10; total
time=   2.7s
[CV] END max_depth=4, min_samples_leaf=20, min_samples_split=20; total
time=   2.6s
[CV] END max_depth=4, min_samples_leaf=20, min_samples_split=20; total
time=   3.3s
[CV] END max_depth=4, min_samples_leaf=20, min_samples_split=20; total
time=   3.2s
[CV] END max_depth=4, min_samples_leaf=20, min_samples_split=20; total
```

```
time=   2.7s
[CV] END max_depth=4, min_samples_leaf=20, min_samples_split=20; total
time=   2.7s
[CV] END max_depth=4, min_samples_leaf=20, min_samples_split=30; total
time=   2.7s
[CV] END max_depth=4, min_samples_leaf=20, min_samples_split=30; total
time=   3.7s
[CV] END max_depth=4, min_samples_leaf=20, min_samples_split=30; total
time=   2.9s
[CV] END max_depth=4, min_samples_leaf=20, min_samples_split=30; total
time=   2.7s
[CV] END max_depth=4, min_samples_leaf=20, min_samples_split=30; total
time=   2.7s
[CV] END max_depth=6, min_samples_leaf=6, min_samples_split=10; total
time=   4.8s
[CV] END max_depth=6, min_samples_leaf=6, min_samples_split=10; total
time=   4.4s
[CV] END max_depth=6, min_samples_leaf=6, min_samples_split=10; total
time=   4.0s
[CV] END max_depth=6, min_samples_leaf=6, min_samples_split=10; total
time=   4.5s
[CV] END max_depth=6, min_samples_leaf=6, min_samples_split=10; total
time=   4.7s
[CV] END max_depth=6, min_samples_leaf=6, min_samples_split=20; total
time=   4.0s
[CV] END max_depth=6, min_samples_leaf=6, min_samples_split=20; total
time=   4.2s
[CV] END max_depth=6, min_samples_leaf=6, min_samples_split=20; total
time=   5.0s
[CV] END max_depth=6, min_samples_leaf=6, min_samples_split=20; total
time=   4.0s
[CV] END max_depth=6, min_samples_leaf=6, min_samples_split=20; total
time=   4.0s
[CV] END max_depth=6, min_samples_leaf=6, min_samples_split=30; total
time=   5.2s
[CV] END max_depth=6, min_samples_leaf=6, min_samples_split=30; total
time=   4.0s
[CV] END max_depth=6, min_samples_leaf=6, min_samples_split=30; total
time=   4.1s
[CV] END max_depth=6, min_samples_leaf=6, min_samples_split=30; total
time=   5.2s
[CV] END max_depth=6, min_samples_leaf=6, min_samples_split=30; total
time=   4.0s
[CV] END max_depth=6, min_samples_leaf=10, min_samples_split=10; total
time=   4.0s
[CV] END max_depth=6, min_samples_leaf=10, min_samples_split=10; total
time=   5.1s
[CV] END max_depth=6, min_samples_leaf=10, min_samples_split=10; total
time=   4.0s
```

```
[CV] END max_depth=6, min_samples_leaf=10, min_samples_split=10; total
time=   4.0s
[CV] END max_depth=6, min_samples_leaf=10, min_samples_split=10; total
time=   4.8s
[CV] END max_depth=6, min_samples_leaf=10, min_samples_split=20; total
time=   4.3s
[CV] END max_depth=6, min_samples_leaf=10, min_samples_split=20; total
time=   3.9s
[CV] END max_depth=6, min_samples_leaf=10, min_samples_split=20; total
time=   4.5s
[CV] END max_depth=6, min_samples_leaf=10, min_samples_split=20; total
time=   4.6s
[CV] END max_depth=6, min_samples_leaf=10, min_samples_split=20; total
time=   4.0s
[CV] END max_depth=6, min_samples_leaf=10, min_samples_split=30; total
time=   4.2s
[CV] END max_depth=6, min_samples_leaf=10, min_samples_split=30; total
time=   5.0s
[CV] END max_depth=6, min_samples_leaf=10, min_samples_split=30; total
time=   4.0s
[CV] END max_depth=6, min_samples_leaf=10, min_samples_split=30; total
time=   4.0s
[CV] END max_depth=6, min_samples_leaf=10, min_samples_split=30; total
time=   5.2s
[CV] END max_depth=6, min_samples_leaf=16, min_samples_split=10; total
time=   4.0s
[CV] END max_depth=6, min_samples_leaf=16, min_samples_split=10; total
time=   4.0s
[CV] END max_depth=6, min_samples_leaf=16, min_samples_split=10; total
time=   5.1s
[CV] END max_depth=6, min_samples_leaf=16, min_samples_split=10; total
time=   4.0s
[CV] END max_depth=6, min_samples_leaf=16, min_samples_split=10; total
time=   4.0s
[CV] END max_depth=6, min_samples_leaf=16, min_samples_split=20; total
time=   5.0s
[CV] END max_depth=6, min_samples_leaf=16, min_samples_split=20; total
time=   4.1s
[CV] END max_depth=6, min_samples_leaf=16, min_samples_split=20; total
time=   4.0s
[CV] END max_depth=6, min_samples_leaf=16, min_samples_split=20; total
time=   4.7s
[CV] END max_depth=6, min_samples_leaf=16, min_samples_split=20; total
time=   4.4s
[CV] END max_depth=6, min_samples_leaf=16, min_samples_split=30; total
time=   4.0s
[CV] END max_depth=6, min_samples_leaf=16, min_samples_split=30; total
time=   4.4s
[CV] END max_depth=6, min_samples_leaf=16, min_samples_split=30; total
```

```
time=   4.7s
[CV] END max_depth=6, min_samples_leaf=16, min_samples_split=30; total
time=   4.0s
[CV] END max_depth=6, min_samples_leaf=16, min_samples_split=30; total
time=   4.1s
[CV] END max_depth=6, min_samples_leaf=20, min_samples_split=10; total
time=   5.1s
[CV] END max_depth=6, min_samples_leaf=20, min_samples_split=10; total
time=   3.9s
[CV] END max_depth=6, min_samples_leaf=20, min_samples_split=10; total
time=   4.0s
[CV] END max_depth=6, min_samples_leaf=20, min_samples_split=10; total
time=   5.1s
[CV] END max_depth=6, min_samples_leaf=20, min_samples_split=10; total
time=   4.0s
[CV] END max_depth=6, min_samples_leaf=20, min_samples_split=20; total
time=   4.0s
[CV] END max_depth=6, min_samples_leaf=20, min_samples_split=20; total
time=   5.1s
[CV] END max_depth=6, min_samples_leaf=20, min_samples_split=20; total
time=   4.0s
[CV] END max_depth=6, min_samples_leaf=20, min_samples_split=20; total
time=   4.0s
[CV] END max_depth=6, min_samples_leaf=20, min_samples_split=20; total
time=   5.0s
[CV] END max_depth=6, min_samples_leaf=20, min_samples_split=30; total
time=   4.2s
[CV] END max_depth=6, min_samples_leaf=20, min_samples_split=30; total
time=   4.0s
[CV] END max_depth=6, min_samples_leaf=20, min_samples_split=30; total
time=   4.6s
[CV] END max_depth=6, min_samples_leaf=20, min_samples_split=30; total
time=   4.4s
[CV] END max_depth=6, min_samples_leaf=20, min_samples_split=30; total
time=   3.9s
[CV] END max_depth=8, min_samples_leaf=6, min_samples_split=10; total
time=   6.1s
[CV] END max_depth=8, min_samples_leaf=6, min_samples_split=10; total
time=   5.6s
[CV] END max_depth=8, min_samples_leaf=6, min_samples_split=10; total
time=   5.2s
[CV] END max_depth=8, min_samples_leaf=6, min_samples_split=10; total
time=   6.5s
[CV] END max_depth=8, min_samples_leaf=6, min_samples_split=10; total
time=   5.3s
[CV] END max_depth=8, min_samples_leaf=6, min_samples_split=20; total
time=   6.4s
[CV] END max_depth=8, min_samples_leaf=6, min_samples_split=20; total
time=   5.3s
```

```
[CV] END max_depth=8, min_samples_leaf=6, min_samples_split=20; total
time=   5.7s
[CV] END max_depth=8, min_samples_leaf=6, min_samples_split=20; total
time=   6.0s
[CV] END max_depth=8, min_samples_leaf=6, min_samples_split=20; total
time=   5.2s
[CV] END max_depth=8, min_samples_leaf=6, min_samples_split=30; total
time=   6.3s
[CV] END max_depth=8, min_samples_leaf=6, min_samples_split=30; total
time=   5.2s
[CV] END max_depth=8, min_samples_leaf=6, min_samples_split=30; total
time=   6.0s
[CV] END max_depth=8, min_samples_leaf=6, min_samples_split=30; total
time=   5.6s
[CV] END max_depth=8, min_samples_leaf=6, min_samples_split=30; total
time=   5.2s
[CV] END max_depth=8, min_samples_leaf=10, min_samples_split=10; total
time=   6.4s
[CV] END max_depth=8, min_samples_leaf=10, min_samples_split=10; total
time=   5.2s
[CV] END max_depth=8, min_samples_leaf=10, min_samples_split=10; total
time=   6.4s
[CV] END max_depth=8, min_samples_leaf=10, min_samples_split=10; total
time=   5.2s
[CV] END max_depth=8, min_samples_leaf=10, min_samples_split=10; total
time=   5.4s
[CV] END max_depth=8, min_samples_leaf=10, min_samples_split=20; total
time=   6.2s
[CV] END max_depth=8, min_samples_leaf=10, min_samples_split=20; total
time=   5.3s
[CV] END max_depth=8, min_samples_leaf=10, min_samples_split=20; total
time=   6.4s
[CV] END max_depth=8, min_samples_leaf=10, min_samples_split=20; total
time=   5.2s
[CV] END max_depth=8, min_samples_leaf=10, min_samples_split=20; total
time=   5.8s
[CV] END max_depth=8, min_samples_leaf=10, min_samples_split=30; total
time=   5.9s
[CV] END max_depth=8, min_samples_leaf=10, min_samples_split=30; total
time=   5.3s
[CV] END max_depth=8, min_samples_leaf=10, min_samples_split=30; total
time=   6.3s
[CV] END max_depth=8, min_samples_leaf=10, min_samples_split=30; total
time=   5.1s
[CV] END max_depth=8, min_samples_leaf=10, min_samples_split=30; total
time=   6.1s
[CV] END max_depth=8, min_samples_leaf=16, min_samples_split=10; total
time=   5.4s
[CV] END max_depth=8, min_samples_leaf=16, min_samples_split=10; total
```

```
time=   5.2s
[CV] END max_depth=8, min_samples_leaf=16, min_samples_split=10; total
time=   6.4s
[CV] END max_depth=8, min_samples_leaf=16, min_samples_split=10; total
time=   5.2s
[CV] END max_depth=8, min_samples_leaf=16, min_samples_split=10; total
time=   6.3s
[CV] END max_depth=8, min_samples_leaf=16, min_samples_split=20; total
time=   5.2s
[CV] END max_depth=8, min_samples_leaf=16, min_samples_split=20; total
time=   5.7s
[CV] END max_depth=8, min_samples_leaf=16, min_samples_split=20; total
time=   6.1s
[CV] END max_depth=8, min_samples_leaf=16, min_samples_split=20; total
time=   5.3s
[CV] END max_depth=8, min_samples_leaf=16, min_samples_split=20; total
time=   6.4s
[CV] END max_depth=8, min_samples_leaf=16, min_samples_split=30; total
time=   5.2s
[CV] END max_depth=8, min_samples_leaf=16, min_samples_split=30; total
time=   6.6s
[CV] END max_depth=8, min_samples_leaf=16, min_samples_split=30; total
time=   7.6s
[CV] END max_depth=8, min_samples_leaf=16, min_samples_split=30; total
time=   6.1s
[CV] END max_depth=8, min_samples_leaf=16, min_samples_split=30; total
time=   5.5s
[CV] END max_depth=8, min_samples_leaf=20, min_samples_split=10; total
time=   5.2s
[CV] END max_depth=8, min_samples_leaf=20, min_samples_split=10; total
time=   6.4s
[CV] END max_depth=8, min_samples_leaf=20, min_samples_split=10; total
time=   5.2s
[CV] END max_depth=8, min_samples_leaf=20, min_samples_split=10; total
time=   6.3s
[CV] END max_depth=8, min_samples_leaf=20, min_samples_split=10; total
time=   5.3s
[CV] END max_depth=8, min_samples_leaf=20, min_samples_split=20; total
time=   5.3s
[CV] END max_depth=8, min_samples_leaf=20, min_samples_split=20; total
time=   6.2s
[CV] END max_depth=8, min_samples_leaf=20, min_samples_split=20; total
time=   5.2s
[CV] END max_depth=8, min_samples_leaf=20, min_samples_split=20; total
time=   6.3s
[CV] END max_depth=8, min_samples_leaf=20, min_samples_split=20; total
time=   5.1s
[CV] END max_depth=8, min_samples_leaf=20, min_samples_split=30; total
time=   5.6s
```

```
[CV] END max_depth=8, min_samples_leaf=20, min_samples_split=30; total
time=   6.0s
[CV] END max_depth=8, min_samples_leaf=20, min_samples_split=30; total
time=   5.2s
[CV] END max_depth=8, min_samples_leaf=20, min_samples_split=30; total
time=   6.4s
[CV] END max_depth=8, min_samples_leaf=20, min_samples_split=30; total
time=   5.3s
[CV] END max_depth=10, min_samples_leaf=6, min_samples_split=10; total
time=   7.6s
[CV] END max_depth=10, min_samples_leaf=6, min_samples_split=10; total
time=   6.3s
[CV] END max_depth=10, min_samples_leaf=6, min_samples_split=10; total
time=   7.4s
[CV] END max_depth=10, min_samples_leaf=6, min_samples_split=10; total
time=   6.3s
[CV] END max_depth=10, min_samples_leaf=6, min_samples_split=10; total
time=   7.5s
[CV] END max_depth=10, min_samples_leaf=6, min_samples_split=20; total
time=   6.4s
[CV] END max_depth=10, min_samples_leaf=6, min_samples_split=20; total
time=   7.5s
[CV] END max_depth=10, min_samples_leaf=6, min_samples_split=20; total
time=   6.4s
[CV] END max_depth=10, min_samples_leaf=6, min_samples_split=20; total
time=   7.4s
[CV] END max_depth=10, min_samples_leaf=6, min_samples_split=20; total
time=   6.3s
[CV] END max_depth=10, min_samples_leaf=6, min_samples_split=30; total
time=   7.4s
[CV] END max_depth=10, min_samples_leaf=6, min_samples_split=30; total
time=   6.8s
[CV] END max_depth=10, min_samples_leaf=6, min_samples_split=30; total
time=   7.5s
[CV] END max_depth=10, min_samples_leaf=6, min_samples_split=30; total
time=   6.7s
[CV] END max_depth=10, min_samples_leaf=6, min_samples_split=30; total
time=   7.5s
[CV] END max_depth=10, min_samples_leaf=10, min_samples_split=10;
total time=   6.4s
[CV] END max_depth=10, min_samples_leaf=10, min_samples_split=10;
total time=   7.5s
[CV] END max_depth=10, min_samples_leaf=10, min_samples_split=10;
total time=   6.3s
[CV] END max_depth=10, min_samples_leaf=10, min_samples_split=10;
total time=   7.4s
[CV] END max_depth=10, min_samples_leaf=10, min_samples_split=10;
total time=   6.3s
[CV] END max_depth=10, min_samples_leaf=10, min_samples_split=20;
```

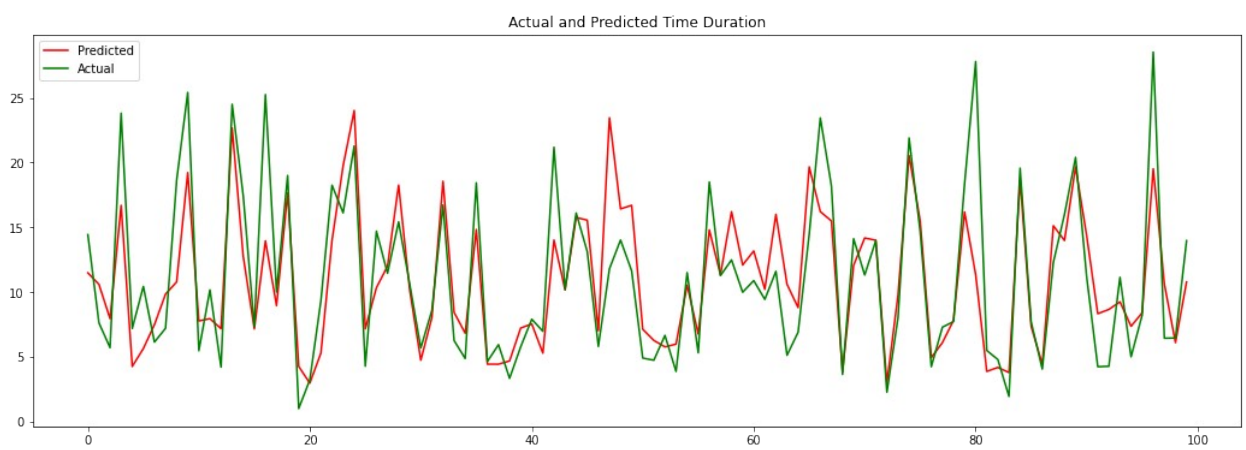
```
total time=   7.4s
[CV] END max_depth=10, min_samples_leaf=10, min_samples_split=20;
total time=   6.2s
[CV] END max_depth=10, min_samples_leaf=10, min_samples_split=20;
total time=   7.4s
[CV] END max_depth=10, min_samples_leaf=10, min_samples_split=20;
total time=   6.3s
[CV] END max_depth=10, min_samples_leaf=10, min_samples_split=20;
total time=   7.5s
[CV] END max_depth=10, min_samples_leaf=10, min_samples_split=30;
total time=   6.3s
[CV] END max_depth=10, min_samples_leaf=10, min_samples_split=30;
total time=   7.5s
[CV] END max_depth=10, min_samples_leaf=10, min_samples_split=30;
total time=   6.3s
[CV] END max_depth=10, min_samples_leaf=10, min_samples_split=30;
total time=   7.5s
[CV] END max_depth=10, min_samples_leaf=10, min_samples_split=30;
total time=   6.4s
[CV] END max_depth=10, min_samples_leaf=16, min_samples_split=10;
total time=   7.5s
[CV] END max_depth=10, min_samples_leaf=16, min_samples_split=10;
total time=   6.3s
[CV] END max_depth=10, min_samples_leaf=16, min_samples_split=10;
total time=   7.5s
[CV] END max_depth=10, min_samples_leaf=16, min_samples_split=10;
total time=   6.3s
[CV] END max_depth=10, min_samples_leaf=16, min_samples_split=10;
total time=   7.5s
[CV] END max_depth=10, min_samples_leaf=16, min_samples_split=20;
total time=   6.3s
[CV] END max_depth=10, min_samples_leaf=16, min_samples_split=20;
total time=   7.5s
[CV] END max_depth=10, min_samples_leaf=16, min_samples_split=20;
total time=   6.3s
[CV] END max_depth=10, min_samples_leaf=16, min_samples_split=20;
total time=   7.4s
[CV] END max_depth=10, min_samples_leaf=16, min_samples_split=20;
total time=   6.3s
[CV] END max_depth=10, min_samples_leaf=16, min_samples_split=30;
total time=   7.4s
[CV] END max_depth=10, min_samples_leaf=16, min_samples_split=30;
total time=   6.3s
[CV] END max_depth=10, min_samples_leaf=16, min_samples_split=30;
total time=   7.5s
[CV] END max_depth=10, min_samples_leaf=16, min_samples_split=30;
total time=   6.3s
[CV] END max_depth=10, min_samples_leaf=16, min_samples_split=30;
total time=   7.6s
```

```
[CV] END max_depth=10, min_samples_leaf=20, min_samples_split=10;
total time=   6.3s
[CV] END max_depth=10, min_samples_leaf=20, min_samples_split=10;
total time=   7.6s
[CV] END max_depth=10, min_samples_leaf=20, min_samples_split=10;
total time=   6.4s
[CV] END max_depth=10, min_samples_leaf=20, min_samples_split=10;
total time=   7.5s
[CV] END max_depth=10, min_samples_leaf=20, min_samples_split=10;
total time=   6.3s
[CV] END max_depth=10, min_samples_leaf=20, min_samples_split=20;
total time=   7.6s
[CV] END max_depth=10, min_samples_leaf=20, min_samples_split=20;
total time=   6.3s
[CV] END max_depth=10, min_samples_leaf=20, min_samples_split=20;
total time=   7.6s
[CV] END max_depth=10, min_samples_leaf=20, min_samples_split=20;
total time=   6.3s
[CV] END max_depth=10, min_samples_leaf=20, min_samples_split=20;
total time=   7.4s
[CV] END max_depth=10, min_samples_leaf=20, min_samples_split=30;
total time=   6.4s
[CV] END max_depth=10, min_samples_leaf=20, min_samples_split=30;
total time=   7.6s
[CV] END max_depth=10, min_samples_leaf=20, min_samples_split=30;
total time=   6.3s
[CV] END max_depth=10, min_samples_leaf=20, min_samples_split=30;
total time=   7.6s
[CV] END max_depth=10, min_samples_leaf=20, min_samples_split=30;
total time=   6.3s
[CV] END max_depth=12, min_samples_leaf=6, min_samples_split=10; total
time=   8.5s
[CV] END max_depth=12, min_samples_leaf=6, min_samples_split=10; total
time=   7.4s
[CV] END max_depth=12, min_samples_leaf=6, min_samples_split=10; total
time=   8.5s
[CV] END max_depth=12, min_samples_leaf=6, min_samples_split=10; total
time=   8.1s
[CV] END max_depth=12, min_samples_leaf=6, min_samples_split=10; total
time=   8.0s
[CV] END max_depth=12, min_samples_leaf=6, min_samples_split=20; total
time=  10.3s
[CV] END max_depth=12, min_samples_leaf=6, min_samples_split=20; total
time=   7.4s
[CV] END max_depth=12, min_samples_leaf=6, min_samples_split=20; total
time=   8.5s
[CV] END max_depth=12, min_samples_leaf=6, min_samples_split=20; total
time=   8.3s
[CV] END max_depth=12, min_samples_leaf=6, min_samples_split=20; total
time=   7.6s
```

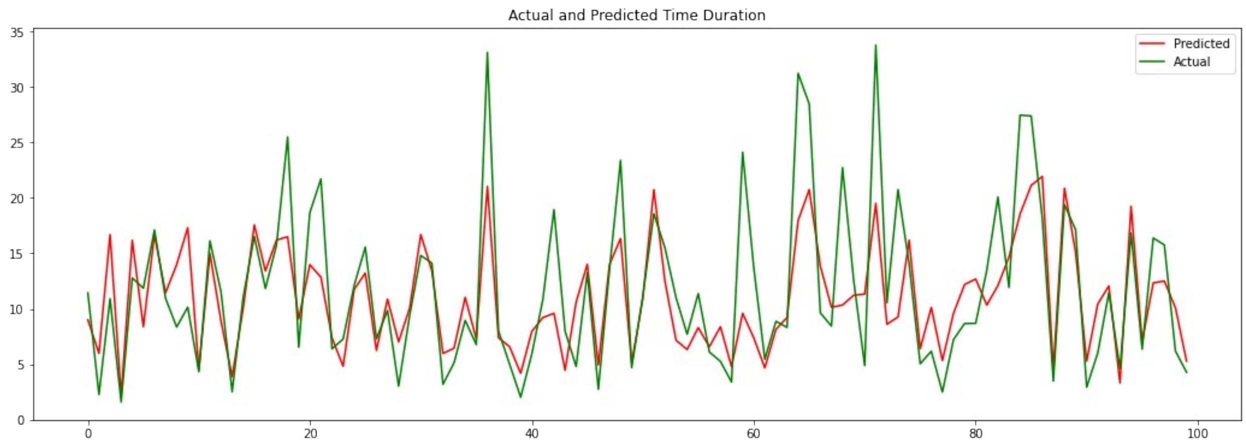
```
[CV] END max_depth=12, min_samples_leaf=6, min_samples_split=30; total
time=   8.5s
[CV] END max_depth=12, min_samples_leaf=6, min_samples_split=30; total
time=   7.4s
[CV] END max_depth=12, min_samples_leaf=6, min_samples_split=30; total
time=   8.5s
[CV] END max_depth=12, min_samples_leaf=6, min_samples_split=30; total
time=   7.8s
[CV] END max_depth=12, min_samples_leaf=6, min_samples_split=30; total
time=   8.1s
[CV] END max_depth=12, min_samples_leaf=10, min_samples_split=10;
total time=   8.4s
[CV] END max_depth=12, min_samples_leaf=10, min_samples_split=10;
total time=   7.3s
[CV] END max_depth=12, min_samples_leaf=10, min_samples_split=10;
total time=   8.5s
[CV] END max_depth=12, min_samples_leaf=10, min_samples_split=10;
total time=   7.3s
[CV] END max_depth=12, min_samples_leaf=10, min_samples_split=10;
total time=   8.5s
[CV] END max_depth=12, min_samples_leaf=10, min_samples_split=20;
total time=   8.0s
[CV] END max_depth=12, min_samples_leaf=10, min_samples_split=20;
total time=   7.8s
[CV] END max_depth=12, min_samples_leaf=10, min_samples_split=20;
total time=   8.6s
[CV] END max_depth=12, min_samples_leaf=10, min_samples_split=20;
total time=   7.4s
[CV] END max_depth=12, min_samples_leaf=10, min_samples_split=20;
total time=   8.6s
[CV] END max_depth=12, min_samples_leaf=10, min_samples_split=30;
total time=   7.5s
[CV] END max_depth=12, min_samples_leaf=10, min_samples_split=30;
total time=   8.4s
[CV] END max_depth=12, min_samples_leaf=10, min_samples_split=30;
total time=   8.4s
[CV] END max_depth=12, min_samples_leaf=10, min_samples_split=30;
total time=   7.4s
[CV] END max_depth=12, min_samples_leaf=10, min_samples_split=30;
total time=   8.5s
[CV] END max_depth=12, min_samples_leaf=16, min_samples_split=10;
total time=   7.3s
[CV] END max_depth=12, min_samples_leaf=16, min_samples_split=10;
total time=   8.5s
[CV] END max_depth=12, min_samples_leaf=16, min_samples_split=10;
total time=   7.8s
[CV] END max_depth=12, min_samples_leaf=16, min_samples_split=10;
total time=  11.2s
[CV] END max_depth=12, min_samples_leaf=16, min_samples_split=10;
```

```
total time=   8.5s
[CV] END max_depth=12, min_samples_leaf=16, min_samples_split=20;
total time=   7.7s
[CV] END max_depth=12, min_samples_leaf=16, min_samples_split=20;
total time=   8.2s
[CV] END max_depth=12, min_samples_leaf=16, min_samples_split=20;
total time=   8.6s
[CV] END max_depth=12, min_samples_leaf=16, min_samples_split=20;
total time=   7.3s
[CV] END max_depth=12, min_samples_leaf=16, min_samples_split=20;
total time=   8.4s
[CV] END max_depth=12, min_samples_leaf=16, min_samples_split=30;
total time=   7.3s
[CV] END max_depth=12, min_samples_leaf=16, min_samples_split=30;
total time=   8.5s
[CV] END max_depth=12, min_samples_leaf=16, min_samples_split=30;
total time=   7.8s
[CV] END max_depth=12, min_samples_leaf=16, min_samples_split=30;
total time=   7.9s
[CV] END max_depth=12, min_samples_leaf=16, min_samples_split=30;
total time=   8.4s
[CV] END max_depth=12, min_samples_leaf=20, min_samples_split=10;
total time=   7.3s
[CV] END max_depth=12, min_samples_leaf=20, min_samples_split=10;
total time=   8.4s
[CV] END max_depth=12, min_samples_leaf=20, min_samples_split=10;
total time=   7.3s
[CV] END max_depth=12, min_samples_leaf=20, min_samples_split=10;
total time=   8.5s
[CV] END max_depth=12, min_samples_leaf=20, min_samples_split=10;
total time=   7.9s
[CV] END max_depth=12, min_samples_leaf=20, min_samples_split=20;
total time=   7.8s
[CV] END max_depth=12, min_samples_leaf=20, min_samples_split=20;
total time=   8.5s
[CV] END max_depth=12, min_samples_leaf=20, min_samples_split=20;
total time=   7.3s
[CV] END max_depth=12, min_samples_leaf=20, min_samples_split=20;
total time=   8.5s
[CV] END max_depth=12, min_samples_leaf=20, min_samples_split=20;
total time=   7.3s
[CV] END max_depth=12, min_samples_leaf=20, min_samples_split=30;
total time=   8.4s
[CV] END max_depth=12, min_samples_leaf=20, min_samples_split=30;
total time=   8.1s
[CV] END max_depth=12, min_samples_leaf=20, min_samples_split=30;
total time=   7.6s
[CV] END max_depth=12, min_samples_leaf=20, min_samples_split=30;
total time=   8.4s
```

```
[CV] END max_depth=12, min_samples_leaf=20, min_samples_split=30;
total time=   7.3s

GridSearchCV(cv=5, estimator=DecisionTreeRegressor(),
             param_grid={'max_depth': [4, 6, 8, 10, 12],
                         'min_samples_leaf': [6, 10, 16, 20],
                         'min_samples_split': [10, 20, 30]},
             scoring='r2', verbose=2)

decision_tree_grid.best_estimator_

DecisionTreeRegressor(max_depth=12, min_samples_leaf=20,
min_samples_split=10)

decision_tree_grid.best_score_

0.561668620153759

decision_tree_optimal_model =decision_tree_grid.best_estimator_
y_predict_train_decision_tree=decision_tree_optimal_model.predict(x_tr
ain)
y_predict_test_decision_tree=decision_tree_optimal_model.predict(x_tes
t)

# evaluation metrics for train data set
evaluation_metrics(x_train,y_train,y_predict_train_decision_tree)

Mean Squared Error: 18.5693 Root Mean Squared Error: 4.309211064684578
R2 Score : 0.5771718360098075 Adjusted R2 Score : 0.5771653265547303
```




```
# evaluation metrics for test data set
evaluation_metrics(x_test,y_test,y_predict_test_decision_tree)

Mean Squared Error: 19.0283 Root Mean Squared Error: 4.362143968279819
R2 Score : 0.5666694419637506 Adjusted R2 Score : 0.5666427561132985
```



##Inference - This algorithm is better than the previous one (linear regression) but still the accuracy score is low.

#**Model 3 - Random Forest**

```
RFR=RandomForestRegressor()

# number of trees in random forest
n_estimators=[20,22,24]
#number of feature to consider at every split
max_features=[0.6]
# maximum number of level in trees
max_depth=[10,16]
#number of samples
max_samples=[0.75,1.0]

# Hyperparameter Grid
param_grid={'n_estimators':n_estimators,
            'max_features':max_features,
            'max_depth':max_depth,
            'max_samples':max_samples,
            }
print(param_grid)

{'n_estimators': [20, 22, 24], 'max_features': [0.6], 'max_depth':
[10, 16], 'max_samples': [0.75, 1.0]}

RF_grid=GridSearchCV(estimator=RFR,param_grid=param_grid,cv=2,verbose=
2)

RF_grid.fit(x_train,y_train)

Fitting 2 folds for each of 12 candidates, totalling 24 fits
[CV] END max_depth=10, max_features=0.6, max_samples=0.75,
n_estimators=20; total time=  24.0s
[CV] END max_depth=10, max_features=0.6, max_samples=0.75,
n_estimators=20; total time=  26.4s
```

```
[CV] END max_depth=10, max_features=0.6, max_samples=0.75,
n_estimators=22; total time=  28.0s
[CV] END max_depth=10, max_features=0.6, max_samples=0.75,
n_estimators=22; total time=  28.8s
[CV] END max_depth=10, max_features=0.6, max_samples=0.75,
n_estimators=24; total time=  31.1s
[CV] END max_depth=10, max_features=0.6, max_samples=0.75,
n_estimators=24; total time=  30.5s
[CV] END max_depth=10, max_features=0.6, max_samples=1.0,
n_estimators=20; total time=  30.4s
[CV] END max_depth=10, max_features=0.6, max_samples=1.0,
n_estimators=20; total time=  30.0s
[CV] END max_depth=10, max_features=0.6, max_samples=1.0,
n_estimators=22; total time=  32.1s
[CV] END max_depth=10, max_features=0.6, max_samples=1.0,
n_estimators=22; total time=  33.3s
[CV] END max_depth=10, max_features=0.6, max_samples=1.0,
n_estimators=24; total time=  37.2s
[CV] END max_depth=10, max_features=0.6, max_samples=1.0,
n_estimators=24; total time=  36.9s
[CV] END max_depth=16, max_features=0.6, max_samples=0.75,
n_estimators=20; total time=  37.1s
[CV] END max_depth=16, max_features=0.6, max_samples=0.75,
n_estimators=20; total time=  37.3s
[CV] END max_depth=16, max_features=0.6, max_samples=0.75,
n_estimators=22; total time=  40.3s
[CV] END max_depth=16, max_features=0.6, max_samples=0.75,
n_estimators=22; total time=  41.1s
[CV] END max_depth=16, max_features=0.6, max_samples=0.75,
n_estimators=24; total time=  45.9s
[CV] END max_depth=16, max_features=0.6, max_samples=0.75,
n_estimators=24; total time=  44.5s
[CV] END max_depth=16, max_features=0.6, max_samples=1.0,
n_estimators=20; total time=  44.4s
[CV] END max_depth=16, max_features=0.6, max_samples=1.0,
n_estimators=20; total time=  45.0s
[CV] END max_depth=16, max_features=0.6, max_samples=1.0,
n_estimators=22; total time=  49.7s
[CV] END max_depth=16, max_features=0.6, max_samples=1.0,
n_estimators=22; total time=  50.2s
[CV] END max_depth=16, max_features=0.6, max_samples=1.0,
n_estimators=24; total time=  53.2s
[CV] END max_depth=16, max_features=0.6, max_samples=1.0,
n_estimators=24; total time=  53.7s

GridSearchCV(cv=2, estimator=RandomForestRegressor(),
             param_grid={'max_depth': [10, 16], 'max_features': [0.6],
                         'max_samples': [0.75, 1.0],
                         'n_estimators': [20, 22, 24]},
             verbose=2)
```

```
RF_grid.best_params_
```

```
{'max_depth': 16, 'max_features': 0.6, 'max_samples': 1.0,
'n_estimators': 20}
```

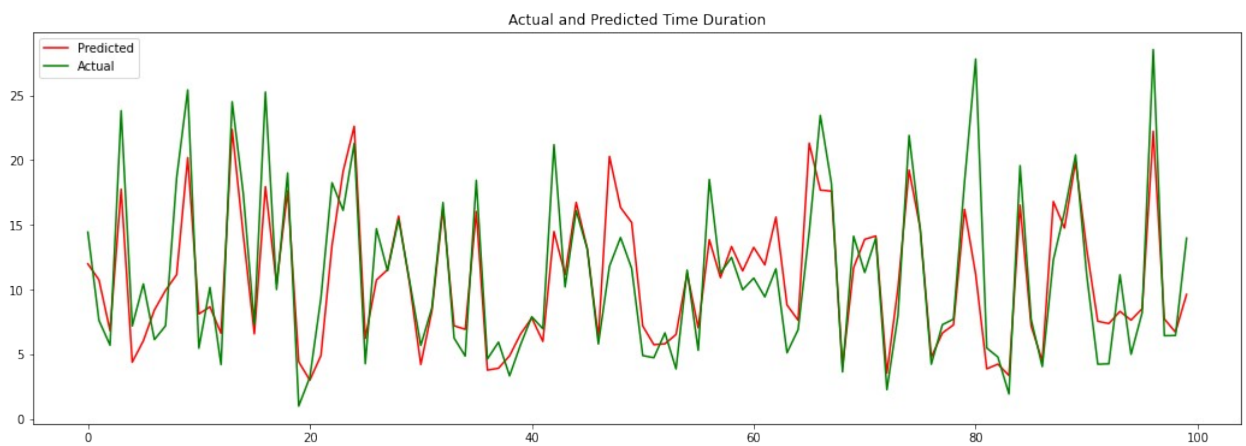
```
RF_grid.best_score_
```

```
0.6011542361879783
```

```
Random_Forest_optimal_model =RF_grid.best_estimator_
y_predict_train_Random_Forest=Random_Forest_optimal_model.predict(x_train)
y_predict_test_Random_Forest=Random_Forest_optimal_model.predict(x_test)
```

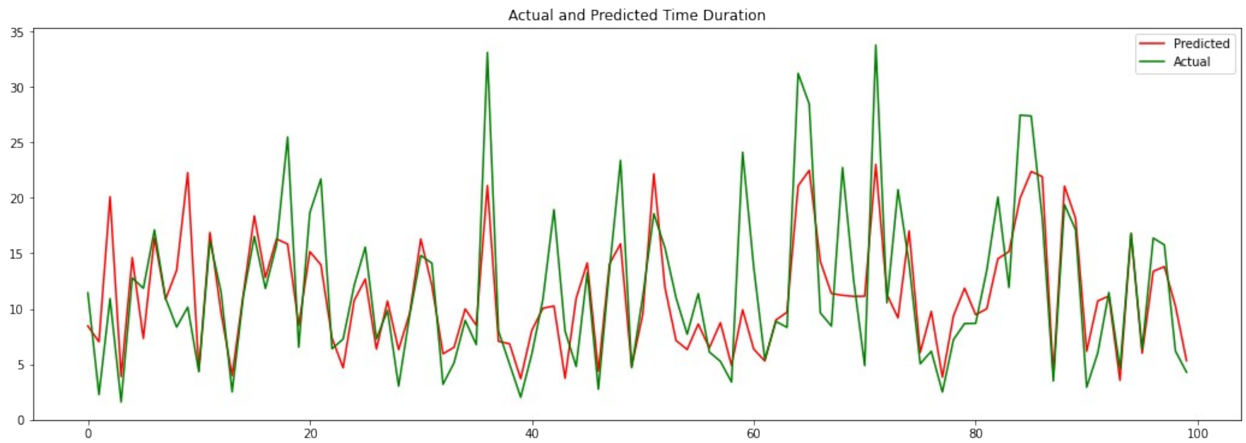
```
# evaluation metrics for train data set
evaluation_metrics(x_train,y_train,y_predict_train_Random_Forest)
```

```
Mean Squared Error: 14.3663 Root Mean Squared Error: 3.790290226354705
R2 Score : 0.672873788148407 Adjusted R2 Score : 0.6728687520283896
```




```
# evaluation metrics for test data set
evaluation_metrics(x_test,y_test,y_predict_test_Random_Forest)
```

```
Mean Squared Error: 17.1459 Root Mean Squared Error: 4.140760799659889
R2 Score : 0.6095358417115371 Adjusted R2 Score : 0.6095117957079914
```



##This algorithm has performed a little better that the previous one (accuracy score:67% train, 60% test).

#**Model 4 - XG Boost**

```
# Number of trees
total_estimators = [50]

# Maximum depth of trees
max_depth_of_trees = [7,9]
min_samples_split = [50]
#learning_rate=[0.1,0.3,0.5]

# Hyperparameter Grid
param_xgboost = {'total_estimators' : total_estimators,
              'max_depth' : max_depth_of_trees,
             'min_samples_split':min_samples_split
              }

# Instantiate  XGBRegressor
import xgboost as xgb
xgboost_model = xgb.XGBRegressor()

# Grid search
xgboost_grid = GridSearchCV(estimator=xgboost_model,param_grid =
param_xgboost,cv = 5, verbose=2,scoring="r2")

xgboost_grid.fit(x_train,y_train)

Fitting 5 folds for each of 2 candidates, totalling 10 fits
[06:43:15] WARNING: ../src/learner.cc:767:
Parameters: { "min_samples_split", "total_estimators" } are not used.

[CV] END max_depth=7, min_samples_split=50, total_estimators=50; total
time= 1.6min
[06:44:50] WARNING: ../src/learner.cc:767:
Parameters: { "min_samples_split", "total_estimators" } are not used.
```

```
[CV] END max_depth=7, min_samples_split=50, total_estimators=50; total
time= 1.8min
[06:46:37] WARNING: ../src/learner.cc:767:
Parameters: { "min_samples_split", "total_estimators" } are not used.

[CV] END max_depth=7, min_samples_split=50, total_estimators=50; total
time= 1.8min
[06:48:22] WARNING: ../src/learner.cc:767:
Parameters: { "min_samples_split", "total_estimators" } are not used.

[CV] END max_depth=7, min_samples_split=50, total_estimators=50; total
time= 1.8min
[06:50:09] WARNING: ../src/learner.cc:767:
Parameters: { "min_samples_split", "total_estimators" } are not used.

[CV] END max_depth=7, min_samples_split=50, total_estimators=50; total
time= 1.9min
[06:52:02] WARNING: ../src/learner.cc:767:
Parameters: { "min_samples_split", "total_estimators" } are not used.

[CV] END max_depth=9, min_samples_split=50, total_estimators=50; total
time= 2.5min
[06:54:29] WARNING: ../src/learner.cc:767:
Parameters: { "min_samples_split", "total_estimators" } are not used.

[CV] END max_depth=9, min_samples_split=50, total_estimators=50; total
time= 2.4min
[06:56:53] WARNING: ../src/learner.cc:767:
Parameters: { "min_samples_split", "total_estimators" } are not used.

[CV] END max_depth=9, min_samples_split=50, total_estimators=50; total
time= 2.0min
[06:58:53] WARNING: ../src/learner.cc:767:
Parameters: { "min_samples_split", "total_estimators" } are not used.

[CV] END max_depth=9, min_samples_split=50, total_estimators=50; total
time= 2.0min
[07:00:54] WARNING: ../src/learner.cc:767:
Parameters: { "min_samples_split", "total_estimators" } are not used.

[CV] END max_depth=9, min_samples_split=50, total_estimators=50; total
time= 2.1min
[07:02:59] WARNING: ../src/learner.cc:767:
Parameters: { "min_samples_split", "total_estimators" } are not used.


GridSearchCV(cv=5,
             estimator=XGBRegressor(base_score=None, booster=None,
                                    callbacks=None,
```

```
colsample_bylevel=None,
                                    colsample_bynode=None,
                                    colsample_bytree=None,
                                    early_stopping_rounds=None,
                                    enable_categorical=False,
eval_metric=None,
                                    feature_types=None, gamma=None,
gpu_id=None,
                                    grow_policy=None,
importance_type=None,
                                    interaction_constraints=None,
                                    learning_rate=None, m...
                                    max_cat_threshold=None,
                                    max_cat_to_onehot=None,
max_delta_step=None,
                                    max_depth=None, max_leaves=None,
                                    min_child_weight=None,
missing=nan,
                                    monotone_constraints=None,
n_estimators=100,
                                    n_jobs=None,
num_parallel_tree=None,
                                    predictor=None, random_state=None,
...),
             param_grid={'max_depth': [7, 9], 'min_samples_split':
[50],
                         'total_estimators': [50]},
             scoring='r2', verbose=2)
```

```
xgboost_grid.best_score_
```

```
0.6177382916991666
```

```
xgboost_grid.best_params_
```

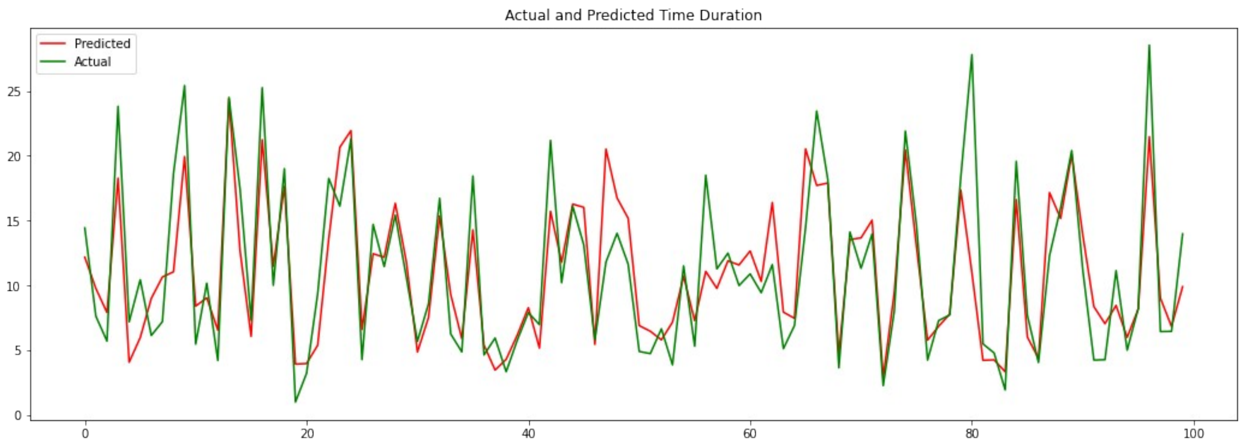
```
{'max_depth': 9, 'min_samples_split': 50, 'total_estimators': 50}
```

```
xgboost_optimal_model =xgboost_grid.best_estimator_
```

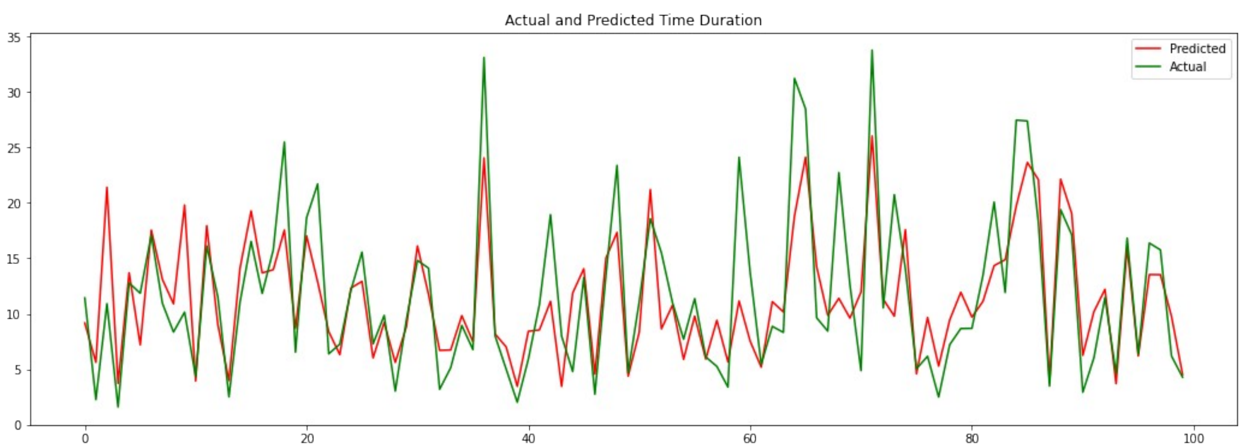
```
y_pred_xgboost_test=xgboost_optimal_model.predict(x_test)
y_pred_xgboost_train=xgboost_optimal_model.predict(x_train)
```

```
#Evaluation metrics for Train set
evaluation_metrics(x_train,y_train,y_pred_xgboost_train)
```

```
Mean Squared Error: 14.7339 Root Mean Squared Error: 3.838476260184502
R2 Score : 0.6645034691246143 Adjusted R2 Score : 0.6644983041432289
```




```
# Evaluation metrics for Test set
evaluation_metrics(x_test,y_test,y_pred_xgboost_test)

Mean Squared Error: 16.5368 Root Mean Squared Error: 4.066546446310432
R2 Score : 0.6234086818752311 Adjusted R2 Score : 0.6233854902045524
```



##This algorithm has given the best accuracy score till now (66% train, 62% test) with low MSE

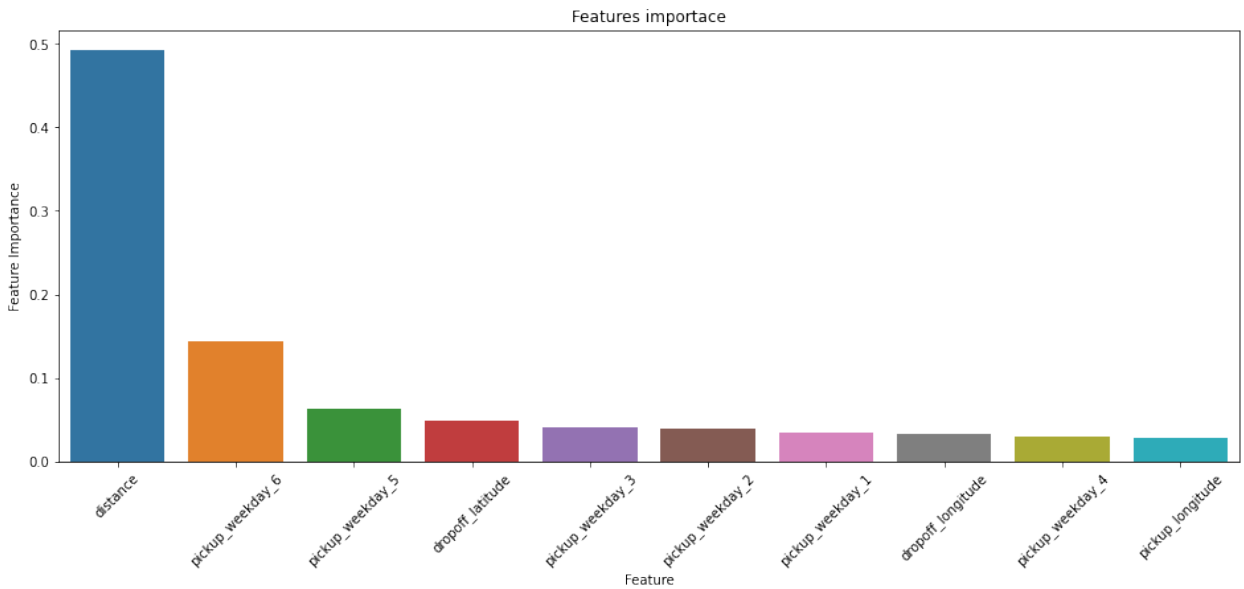
```
x.columns

Index(['vendor_id', 'passenger_count', 'distance', 'pickup_longitude',
       'pickup_latitude', 'dropoff_longitude', 'dropoff_latitude',
       'store_and_fwd_flag_Y', 'pickup_weekday_1', 'pickup_weekday_2',
       'pickup_weekday_3', 'pickup_weekday_4', 'pickup_weekday_5',
       'pickup_weekday_6'],
      dtype='object')

importance=xgboost_optimal_model.feature_importances_
importance
```

```
array([0.0054059 , 0.00540721, 0.49205348, 0.02869008, 0.02682842,
       0.03292176, 0.04935059, 0.0064651 , 0.03490359, 0.03953377,
       0.04098299, 0.0302284 , 0.06294437, 0.14428443], dtype=float32)
```

```python
imp_dict = {'Feature' : list(x.columns),
                   'Feature Importance' : importance}

importance_df = pd.DataFrame(imp_dict)

importance_df.sort_values(by=['Feature
Importance'],ascending=False,inplace=True)
importance_df
```

	Feature	Feature Importance
2	distance	0.492053
13	pickup_weekday_6	0.144284
12	pickup_weekday_5	0.062944
6	dropoff_latitude	0.049351
10	pickup_weekday_3	0.040983
9	pickup_weekday_2	0.039534
8	pickup_weekday_1	0.034904
5	dropoff_longitude	0.032922
11	pickup_weekday_4	0.030228
3	pickup_longitude	0.028690
4	pickup_latitude	0.026828
7	store_and_fwd_flag_Y	0.006465
1	passenger_count	0.005407
0	vendor_id	0.005406

```python
# Feature importance plot
plt.figure(figsize=(16,6))
plt.title('Features importace')
sns.barplot(x='Feature',y="Feature
Importance",data=importance_df[:10])
plt.xticks(rotation=45)
plt.show()
```



As we can see that most important feature is our distance column which affect our dependent variable the most

#**Model 5 - Gradient Boost**

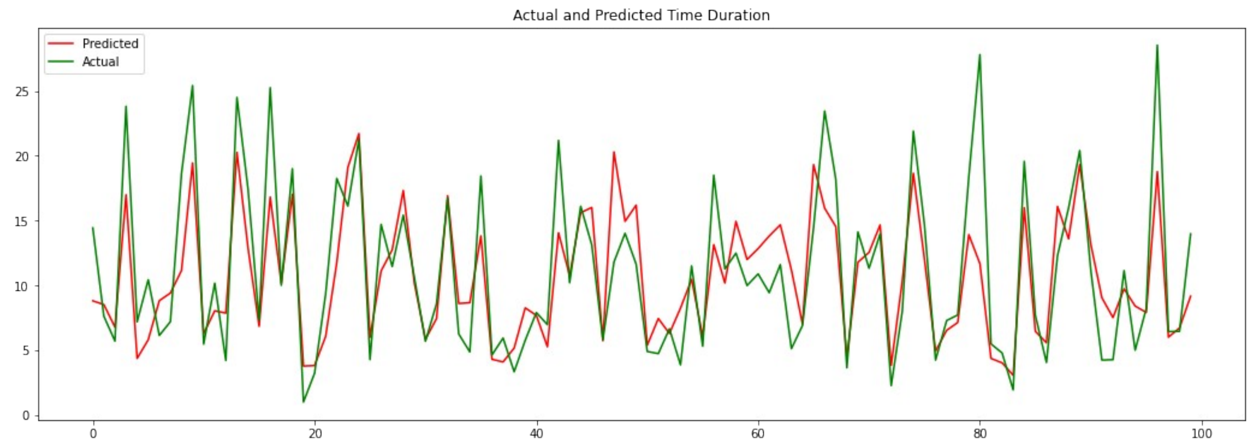
```
# Create an instance of the  GradientBoostingRegressor
from sklearn.ensemble import GradientBoostingRegressor
gradient_boost_model=GradientBoostingRegressor()
gradient_boost_model.fit(x_train,y_train)
```

```
GradientBoostingRegressor()
```

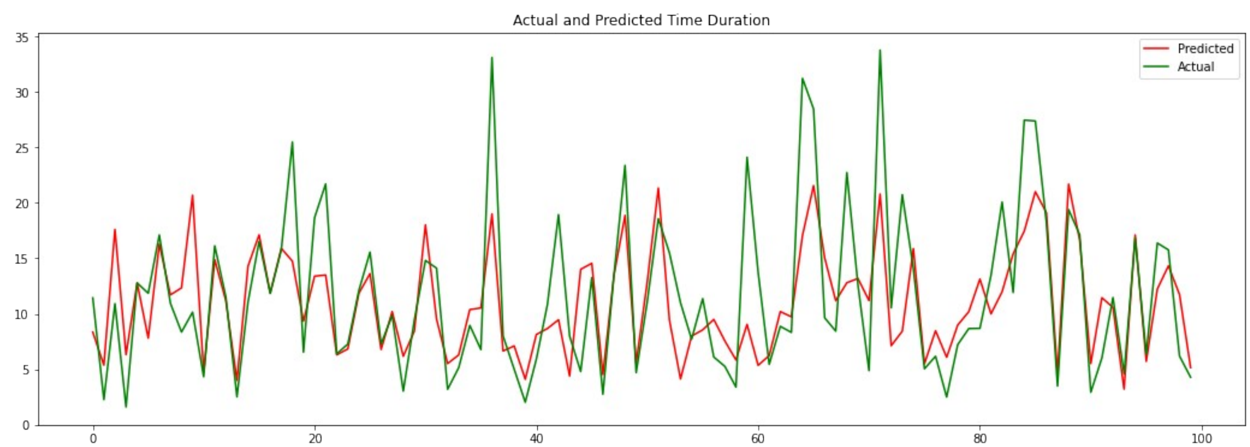
```
y_preds_gradient_boost_test = gradient_boost_model.predict(x_test)
y_pred_gradient_boost_train=gradient_boost_model.predict(x_train)
```

```
#Evaluation metrics for Train set
evaluation_metrics(x_train,y_train,y_pred_gradient_boost_train)
```

```
Mean Squared Error: 19.6792 Root Mean Squared Error: 4.436124434683951
R2 Score : 0.5518977478192577 Adjusted R2 Score : 0.5518908492686668
```




```
#Evaluation metrics for Test set
evaluation_metrics(x_test,y_test,y_preds_gradient_boost_test)

Mean Squared Error: 19.612 Root Mean Squared Error: 4.428543778715527
R2 Score : 0.5533771167133508 Adjusted R2 Score : 0.553349612279955
```



##Above algorithm has an accuracy score of 55% which is lower that our previous algorithm (XG Boost)

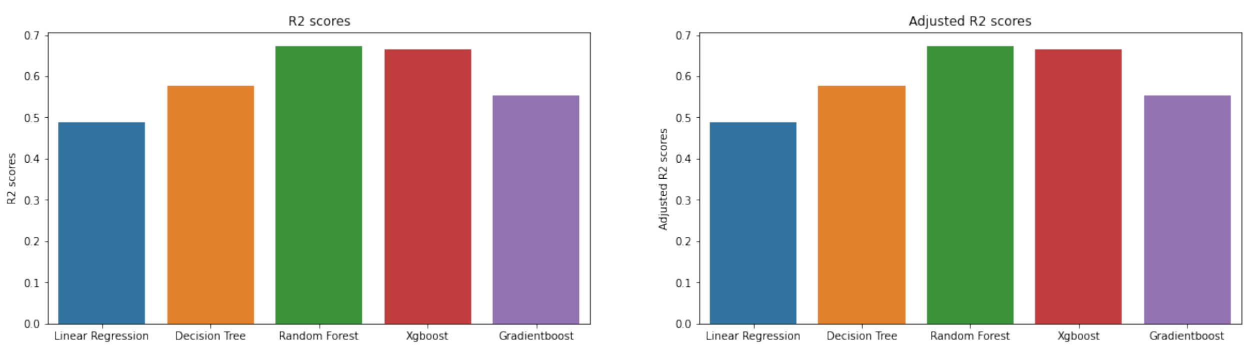
#**STEP 9 - Comparing evaluation metrics of different models**

```
# list of all evaluation matrics values
score_values=[[22.5249,4.746040454947682,0.48710009447510005,0.4870921983622618],

[18.5693,4.309211064684578,0.5771718360098075,0.5771653265547303],

[14.3663,3.790290226354705,0.672873788148407,0.6728687520283896],
              [ 14.7339,3.838476260184502,0.6645034691246143 ,
0.6644983041432289],

[19.6792,4.436124434683951,0.5518977478192577,0.5518908492686668]]
```

```
# Create the pandas DataFrame
df_score = pd.DataFrame(score_values,columns=['MSE', 'RMSE', 'R2',
'AdjustedR2'], index=['Linear Regression', 'Decision Tree', 'Random
Forest', 'Xgboost', 'Gradientboost'])
df_score
```

	MSE	RMSE	R2	AdjustedR2
Linear Regression	22.5249	4.746040	0.487100	0.487092
Decision Tree	18.5693	4.309211	0.577172	0.577165
Random Forest	14.3663	3.790290	0.672874	0.672869
Xgboost	14.7339	3.838476	0.664503	0.664498
Gradientboost	19.6792	4.436124	0.551898	0.551891

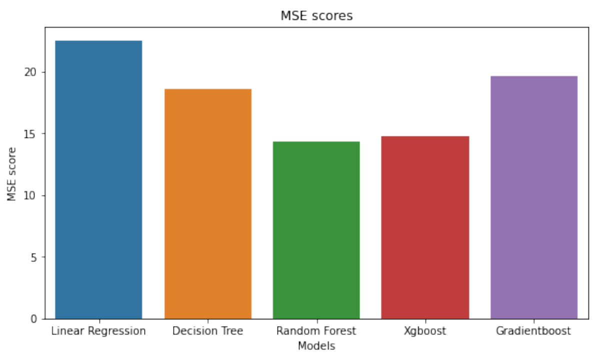
```
#bar plot for R2 score
fig, (ax1, ax2) = plt.subplots(ncols = 2, figsize = (20, 5))
x_= ['Linear Regression', 'Decision Tree', 'Random Forest', 'Xgboost',
'Gradientboost']
ax1.set_title('R2 scores')
ax = sns.barplot(x = x_, y='R2', data =df_score , ax = ax1)
ax.set_xlabel('Models')
ax.set_ylabel('R2 scores')

# barplot for adjustedR2
ax = sns.barplot(x = x_, y='AdjustedR2',  data = df_score, ax = ax2)
ax2.set_title('Adjusted R2 scores')
ax.set_xlabel('Models')
ax.set_ylabel('Adjusted R2 scores')
plt.show()
```

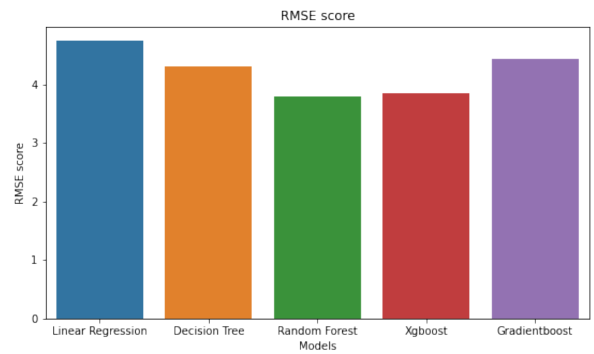


##The above graph clearly shows that Random forest has highest R2 scores and adjusted R2 score which suggests that it has better efficiency than other models.

```
#barplot of MSE score
fig, (ax1, ax2) = plt.subplots(ncols = 2, figsize = (20, 5))
x_= ['Linear Regression', 'Decision Tree', 'Random Forest', 'Xgboost',
'Gradientboost']
ax1.set_title('MSE scores')
```

```python
ax = sns.barplot(x = x_, y='MSE', data =df_score , ax = ax1)
ax.set_xlabel('Models')
ax.set_ylabel('MSE score')

# barplot for RMSE score
ax = sns.barplot(x = x_, y='RMSE',  data = df_score, ax = ax2)
ax2.set_title('RMSE score')
ax.set_xlabel('Models')
ax.set_ylabel('RMSE score')
plt.show()
```





##Only Random Forest has least errors, therefore it can be considered as good algorithm for training our model.

```
# This is formatted as code
```

#**Step 6-Conclusion for EDA:**

- Vendor id distribution shows Vendor 2 receives more number of bookings
- Store_and_ fwd_flag Count shows that majority of the time the taxi driver hasn't logged onto the vendor's systems.
- Distribution of pickups and dropoffs on daily basis interprets that we can see that compared to other days, taxi booking rates are higher on the weekends (4- Friday and 5-Saturday).This suggests that individuals used to go out on weekends for their celebrations, parties, or even other personnel work.
- Distribution of pickups and dropoffs on monthly basis shows that taxi reservations were more in the month of March and April.
- Monthly trend for vendors tells us that both vendors' trips are at their maximum in the month of March and their lowest in the month of January, February, and after June.
- Distribution of pickups and dropoffs on hourly basis gives us the insight that people often use taxi services to get to their workplaces in the mornings after 10:00. Additionally, the demand for taxis tends to surge in the late evening after six o'clock.

- Passenger count distribution shows that most of the bookings are made by solo travelers, which means less number of people prefer car pool or amy be less number of groups book car...people prefer to ride solo

Conclusion for Model Training:

- There were a lot of outliers in our variables some values were near to zero, we tried to remove those values but we found that we were losing a lot of data. we trained our model using various algorithms and we got an accuracy of 67%.
- we were curious whether the model was overfit or not, hopefully it was not, as it gave pretty much similar results for train and test data in all the algorithms tried.
- In all the above model's graph we saw that actual and predicted values are almost near to each other (lines coinciding) in only 2 models namely: **XG Boost** and **Random Forest**. **R2 scores** were also **high** for the above two models and **MSE scores** were also **low** in these models which satisafies the requirements of a good model.
- So we came to a conclusion that removing data removes a lot of information, new column if highly collinear can give pseudo good results,also we got our best R2 score from **Random Forest** model,we tried taking an optimum parameter so that our model doesnt overfit.