FLIP ROBO

A

Data science

Project

on

“Flight Price Prediction”

Submitted by:

Santosh Arvind Dharam

ACKNOWLEDGMENT

I feel great pleasure to present the Project entitled “Flight Price Prediction”. But it would be unfair on our part if I do not acknowledge efforts of some of the people without the support of whom, this Project would not have been a success. First and for most I am very much thankful to my respected SME ‘Swati Mahaseth’ for his leading guidance in this Project. Also he has been persistent source of inspiration to me. I would like to express my sincere thanks and appreciation to ‘flip robo’ for their valuable support. Most importantly I would like to express our sincere gratitude towards my Friend & Family for always being there when I needed them most.

Mr. Santosh Arvind Dharam

INTRODUCTION

Flight ticket prices can be something hard to guess, today we might see a price, check out the price of the same flight tomorrow, it will be a different story. We might have often heard travellers saying that flight ticket prices are so unpredictable. Anyone who has booked a flight ticket knows how unexpectedly the prices vary. The cheapest available ticket on a given flight gets more and less expensive over time. This usually happens as an attempt to maximize revenue

PROBLEM STATEMENT

We have to work on a project where we collect data of flight fares with other features and work to make a model to predict fares of flights.

Analytical Problem Framing

EDA steps:

1) import necessary libraries:

first we will import all the necessary libraries which will be usefull for analysis of data

In [1]:
```
#import all libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
from sklearn.metrics import r2_score,mean_absolute_error,mean_squared_error
from sklearn.linear_model import LogisticRegression,Lasso,LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import AdaBoostRegressor,GradientBoostingRegressor
from sklearn.preprocessing import LabelEncoder,StandardScaler
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.decomposition import PCA
from scipy.stats import zscore
from sklearn.model_selection import cross_val_score
```

in this case we have to import all the necessary library that are usefull for data analysis in jupyter notebook

2)extract the dataset in jupyter notebook:

In [2]:
```
data=pd.read_excel("C:\\Users\\SAI BABA\\Desktop\\flight.xlsx")
data.head()
```

Out[2]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	non-stop	No info	3897
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	2 stops	No info	7662
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	19h	2 stops	No info	13882
3	IndiGo	12/05/2019	Kolkata	Banglore	CCU → NAG → BLR	18:05	23:30	5h 25m	1 stop	No info	6218
4	IndiGo	01/03/2019	Banglore	New Delhi	BLR → NAG → DEL	16:50	21:35	4h 45m	1 stop	No info	13302

In [3]:
```
data.shape
```

Out[3]: (10683, 11)

Data is extracted for further analysis in jupyter notebook.data contains a 10683 rows and 11 columns some columns contains categorical data and some contains numerical .

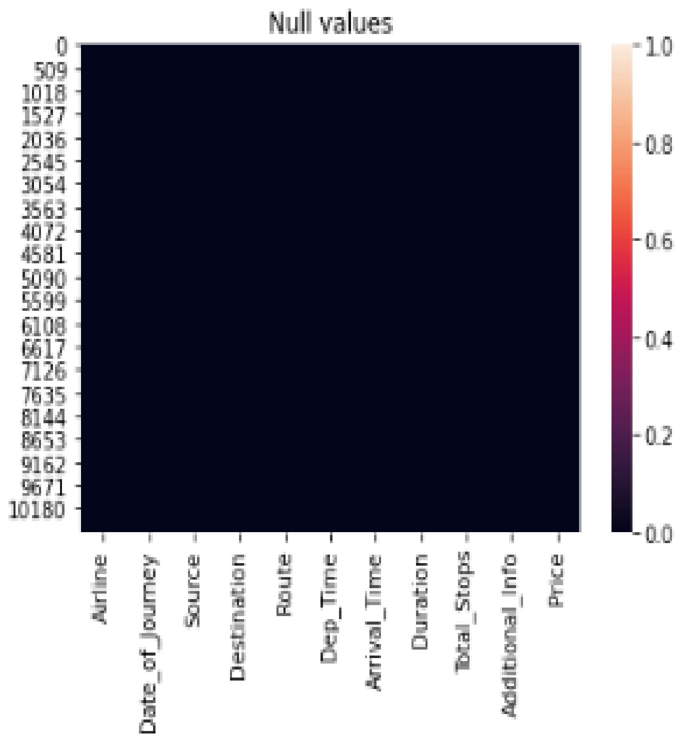
3)checking null values:

In this case we have to find out the null values present in our data set. if it is there it is required to remove it. in our data set has some null values it is also shown by heat map

```
In [7]: data.isnull().sum()

Out[7]: Airline            0
        Date_of_Journey    0
        Source             0
        Destination        0
        Route              1
        Dep_Time           0
        Arrival_Time       0
        Duration           0
        Total_Stops        1
        Additional_Info    0
        Price              0
        dtype: int64
```

```
In [8]: sns.heatmap(data.isnull())
        plt.title("Null values")
        plt.show()
```



4)Making dataframe for categorical data:

as the data contains some categorical columns and some numerical data lets we will form the dataframe for this two categories

```
In [9]: data.columns

Out[9]: Index(['Airline', 'Date_of_Journey', 'Source', 'Destination', 'Route',
               'Dep_Time', 'Arrival_Time', 'Duration', 'Total_Stops',
               'Additional_Info', 'Price'],
              dtype='object')

In [10]: data_visualization_categorical=data[['Airline','Source','Destination','Additional_Info']].copy()

In [11]: data_visualization_categorical.columns

Out[11]: Index(['Airline', 'Source', 'Destination', 'Additional_Info'], dtype='object')

In [12]: len(data_visualization_categorical.columns)

Out[12]: 4

In [13]: data['Airline'].unique()

Out[13]: array(['IndiGo', 'Air India', 'Jet Airways', 'SpiceJet',
                'Multiple carriers', 'GoAir', 'Vistara', 'Air Asia',
                'Vistara Premium economy', 'Jet Airways Business',
                'Multiple carriers Premium economy', 'Trujet'], dtype=object)

In [14]: data['Source'].unique()

Out[14]: array(['Banglore', 'Kolkata', 'Delhi', 'Chennai', 'Mumbai'], dtype=object)

In [15]: data['Destination'].unique()

Out[15]: array(['New Delhi', 'Banglore', 'Cochin', 'Kolkata', 'Delhi', 'Hyderabad'],
               dtype=object)

In [16]: data['Additional_Info'].unique()

Out[16]: array(['No info', 'In-flight meal not included',
                'No check-in baggage included', '1 Short layover', 'No Info',
                '1 Long layover', 'Change airports', 'Business class',
                'Red-eye flight', '2 Long layover'], dtype=object)
```

We have formed dataframe of categorical column and and also we found a unique values present in the particular column for further analysis

a)split the Date column to extract the 'Date', 'Month' and 'Year' values, and store them in new columns in our dataframe:

now we will do some further analysis we will separate the date,month and year from the date column so that task is more easy for us

```
In [17]: data.Date_of_Journey=data.Date_of_Journey.str.split('/')

In [18]: data.Date_of_Journey

Out[18]: 0        [24, 03, 2019]
         1         [1, 05, 2019]
         2         [9, 06, 2019]
         3        [12, 05, 2019]
         4        [01, 03, 2019]
                       ...
         10678     [9, 04, 2019]
         10679    [27, 04, 2019]
         10680    [27, 04, 2019]
         10681    [01, 03, 2019]
         10682     [9, 05, 2019]
         Name: Date_of_Journey, Length: 10683, dtype: object

In [19]: data['Date']=data.Date_of_Journey.str[0]
         data['Month']=data.Date_of_Journey.str[1]
         data['Year']=data.Date_of_Journey.str[2]
```

split the Route column to create multiple columns with cities that the flight travels through

```
In [20]: data.Route=data.Route.str.split('→')

In [21]: data.Route

Out[21]: 0                    [BLR ,  DEL]
         1        [CCU ,  IXR ,  BBI ,  BLR]
         2        [DEL ,  LKO ,  BOM ,  COK]
         3              [CCU ,  NAG ,  BLR]
         4              [BLR ,  NAG ,  DEL]
                             ...
         10678                [CCU ,  BLR]
         10679                [CCU ,  BLR]
         10680                [BLR ,  DEL]
```

b) split the Route column to create multiple columns with cities that the flight travels through:

lets split the route column for detail analysis

```
In [22]: data['City1']=data.Route.str[0]
         data['City2']=data.Route.str[1]
         data['City3']=data.Route.str[2]
         data['City4']=data.Route.str[3]
         data['City5']=data.Route.str[4]
         data['City6']=data.Route.str[5]
```

we split the Dep_time column

```
In [23]: data.Dep_Time=data.Dep_Time.str.split(':')
```

```
In [24]: data.Dep_Time
```

```
Out[24]: 0        [22, 20]
         1        [05, 50]
         2        [09, 25]
         3        [18, 05]
         4        [16, 50]
                    ...
         10678    [19, 55]
         10679    [20, 45]
         10680    [08, 20]
         10681    [11, 30]
         10682    [10, 55]
         Name: Dep_Time, Length: 10683, dtype: object
```

```
In [25]: data['Dep_Time_Hour']=data.Dep_Time.str[0]
         data['Dep_Time_Min']=data.Dep_Time.str[1]
```

```
In [26]: data.Arrival_Time=data.Arrival_Time.str.split(' ')
```

```
In [27]: data.Arrival_Time
```

```
Out[27]: 0        [01:10, 22, Mar]
         1                 [13:15]
         2        [04:25, 10, Jun]
         3                 [23:30]
```

c) we divide the 'Duration' column to 'Travel_ hours' and ' Travel_ mins' also treat the 'Total _stops' column, further to the 'Additional_ info' column, We replace 'No Info' by 'No info' to merge it into a single category:

```
In [30]: data.Duration=data.Duration.str.split(' ')
         data['Travel_hours']=data.Duration.str[0]
         data['Travel_hours']=data['Travel_hours'].str.split('h')
         data['Travel_hours']=data['Travel_hours'].str[0]
         data.Travel_hours=data.Travel_hours
```

```
In [31]: data['Travel_mins']=data.Duration.str[1]
```

```
In [32]: data.Travel_mins=data.Travel_mins.str.split('m')
         data.Travel_mins=data.Travel_mins.str[0]
```

We also treat the 'Total_stops' column

```
In [33]: data.Total_Stops.replace('non-stop','0',inplace=True)
         data.Total_Stops=data.Total_Stops.str.split(' ')
         data.Total_Stops=data.Total_Stops.str[0]
```

We proceed further to the 'Additional_info' column, We replace 'No Info' by 'No info' to merge it into a single categor

```
In [34]: data.Additional_Info.replace('No Info','No info',inplace=True)
```

So now our task is quite simple now so that we will able to found out deep insight present in our dataset

5)dropping the unwanted columns ,checking and removing the null values:

Now by seeing the data set it is found that some data has less values or more null values present to it also it has not that much importance on the output variable so lets we will drop it.

```
In [36]: data.drop(['Route','City5','City6'],axis=1,inplace=Tru
```

```
In [37]: data.isnull().sum()
```

```
Out[37]: Airline                0
         Date_of_Journey        0
         Source                 0
         Destination            0
         Dep_Time               0
         Arrival_Time           0
         Duration               0
         Total_Stops            1
         Additional_Info        0
         Price                  0
         Date                   0
         Month                  0
         Year                   0
         City1                  1
         City2                  1
         City3               3492
         City4               9117
         Dep_Time_Hour          0
         Dep_Time_Min           0
         Arrival_date        6348
         Time_of_arrival        0
         Arrival_Time_Hour      0
         Arrival_Time_Min       0
         Travel_hours           0
         Travel_mins         1032
         dtype: int64
```

```
In [38]: data.drop(['City4'],axis=1,inplace=True)
```

```
In [39]: #now will fill the missing values
         data['City3'].fillna('None',inplace=True)
         data['Arrival_date'].fillna(data['Date'],inplace=True)
         data['Travel_mins'].fillna(0,inplace=True)
```

Also we will replace the null values which are present in the dataset


```
In [38]: data.drop(['City4'],axis=1,inplace=True)
```

```
In [39]: #now will fill the missing values
         data['City3'].fillna('None',inplace=True)
         data['Arrival_date'].fillna(data['Date'],inplace=True)
         data['Travel_mins'].fillna(0,inplace=True)
```


```
In [40]: data['City1'].fillna('DEL',inplace=True)
         data['City2'].fillna('COK',inplace=True)
         data['Total_Stops'].fillna(0,inplace=True)
```

7) changing columns from object to integer:

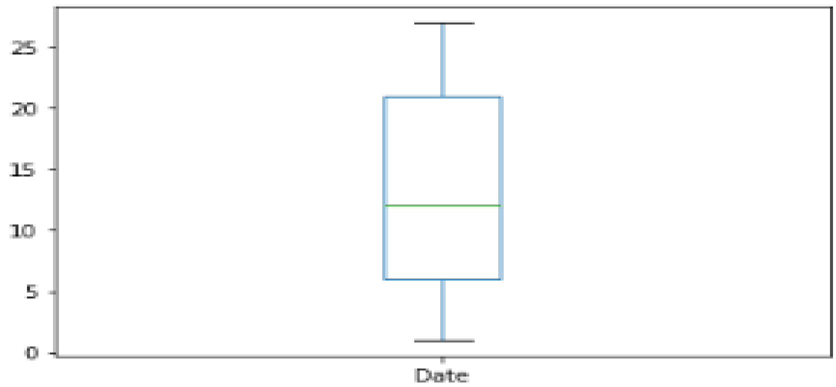
Now lets we will change object type columns into the integer one for further analysis

```
In [42]: #changing numerical columns from object to int
         data.Total_Stops=data.Total_Stops.astype('int64')
         data.Date=data.Date.astype('int64')
         data.Month=data.Month.astype('int64')
         data.Year=data.Year.astype('int64')
         data.Dep_Time_Hour=data.Dep_Time_Hour.astype('int64')
         data.Dep_Time_Min=data.Dep_Time_Min.astype('int64')
         data.Arrival_date=data.Arrival_date.astype('int64')
         data.Arrival_Time_Hour=data.Arrival_Time_Hour.astype('int64'
         data.Arrival_Time_Min=data.Arrival_Time_Min.astype('int64')
         data.Travel_mins=data.Travel_mins.astype('int64')
         data.drop(index=6474,inplace=True,axis=0)
```

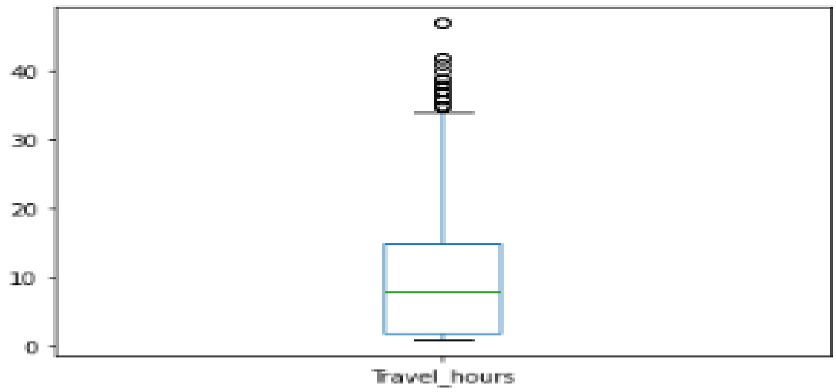
So we have converted the object type columns into the integer type

8) checking outliers:

Now we will check the outliers present in our dataset




```
In [47]: data['Travel_hours'].plot.box()
Out[47]: <AxesSubplot:>
```



8)ckecking and removing the skewness:

```
In [48]: data.skew()

Out[48]: Total_Stops           0.317345
         Price                 1.813248
         Date                  0.118174
         Month                -0.387708
         Year                  0.000000
         Dep_Time_Hour         0.113224
         Dep_Time_Min          0.167210
         Arrival_date          0.119667
         Arrival_Time_Hour    -0.369876
         Arrival_Time_Min      0.110928
         Travel_hours          0.850822
         Travel_mins          -0.091004
         dtype: float64

In [49]: data.Travel_hours=np.log(data.Travel_hours)
         data.Travel_hours.skew()

Out[49]: -0.26612233332369917
```

so we have sucessfully skewed data

Now we have checked the skewness present in the datset ,and we have successfully skewed the data by log method

10)encoding the dataset:

In this case as our data contains some categorical column having object type of data it is necessary to convert it into numerical form by Label Encoder

```
In [50]: from sklearn.preprocessing import LabelEncoder
```

```
In [51]: le=LabelEncoder()
         label=le.fit_transform(data["Airline"])
         label
         data=data.drop("Airline",axis='columns')
         data["Airline"]=label
```

```
In [52]: le=LabelEncoder()
         label=le.fit_transform(data["Source"])
         label
         data=data.drop("Source",axis='columns')
         data["Source"]=label
```

```
In [53]: le=LabelEncoder()
         label=le.fit_transform(data["Destination"])
         label
         data=data.drop("Destination",axis='columns')
         data["Destination"]=label
```

```
In [54]: le=LabelEncoder()
         label=le.fit_transform(data["Additional_Info"])
         label
         data=data.drop("Additional_Info",axis='columns')
         data["Additional_Info"]=label
```

```
In [55]: le=LabelEncoder()
         label=le.fit_transform(data["City1"])
         label
         data=data.drop("City1",axis='columns')
         data["City1"]=label
```

```
In [56]: le=LabelEncoder()
         label=le.fit_transform(data["City2"])
         label
         data=data.drop("City2",axis='columns')
         data["City2"]=label
```

```
In [57]: le=LabelEncoder()
         label=le.fit_transform(data["City3"])
         label
         data=data.drop("City3",axis='columns')
         data["City3"]=label
         data
```

Out[57]:

	Date_of_Journey	Dep_Time	Arrival_Time	Duration	Total_Stops	Price	Date	Month	Year	Dep_Time_Hour	...	Arrival_Time_Min	Travel_hours
0	[24, 03, 2019]	[22, 20]	[01:10, 22, Mar]	[2h, 50m]	0	3897	24	3	2019	22	...	10	0.693147
1	[1, 05, 2019]	[05, 50]	[13:15]	[7h, 25m]	2	7662	1	5	2019	5	...	15	1.945910
2	[9, 06, 2019]	[09, 25]	[04:25, 10, Jun]	[19h]	2	13882	9	6	2019	9	...	25	2.944439
3	[12, 05, 2019]	[18, 05]	[23:30]	[5h, 25m]	1	6218	12	5	2019	18	...	30	1.609438
4	[01, 03, 2019]	[16, 50]	[21:35]	[4h, 45m]	1	13302	1	3	2019	16	...	35	1.386294
...	...	...	...	...	...	...	...	...	...	...	...	...	...
10678	[9, 04, 2019]	[19, 55]	[22:25]	[2h, 30m]	0	4107	9	4	2019	19	...	25	0.693147

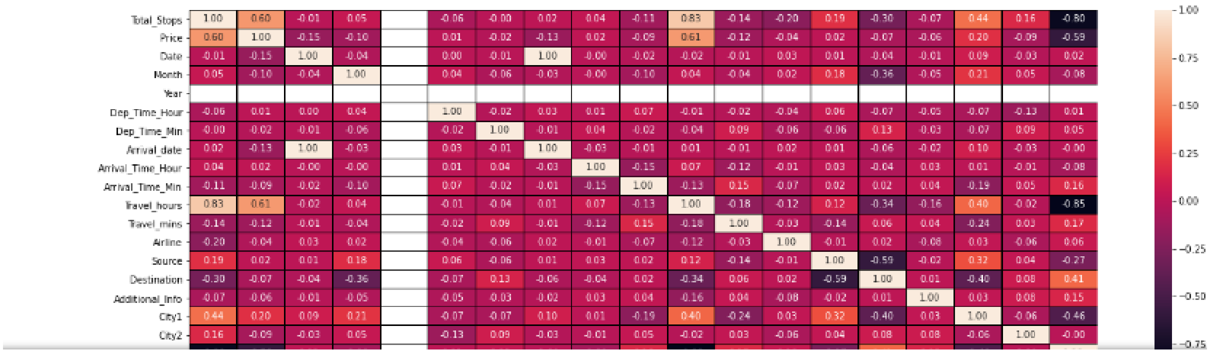
We have successfully encoded the dataset as seen in the above table with the help of label encoder

11) **heatmap:**

Now lets we will check the correlation of each column with the output variable and also with its own by heat map

```
In [61]: #heat map
         plt.figure(figsize=(22,7))
         sns.heatmap(data.corr(),annot=True,linewidths=0.1,linecolor='black',fmt='0.2f')
```

```
Out[61]: <AxesSubplot:>
```



Data is correlated with other column data and also with its own it also gives the positive negative correlation of data with respective one another. it shows that some column has maximum correlation with the price,also the column ‘year’has no data present in it so lets we will drop that column.

12) devide data set into feature and label:

Lets we will devide the dates into variable x,y for further analysis

```
In [65]: from sklearn.preprocessing import LabelEncoder,StandardScaler
```

```
In [66]: #devide data set into feature and label
         y=data['Price']
         x=data.drop(['Price'],axis=1)
```

```
In [67]: #Standardize the value of x so that mean will 0 and SD will become 1 , and make the data as normal distributed
         sc = StandardScaler()
         sc.fit_transform(x)
         x = pd.DataFrame(x,columns=x.columns)
```

So we have devided the dataset into variable x,y so thet we can form the model in the successive steps

13) Now by using multiple Algorithms we are calculating the best Algo which suit best for our data set:

```
from sklearn.ensemble import AdaBoostRegressor,GradientBoostingRegressor
from sklearn.preprocessing import LabelEncoder,StandardScaler
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.decomposition import PCA
from scipy.stats import zscore
from sklearn.model_selection import cross_val_score
```

```
In [71]: #Now by using multiple Algorithms we are calculating the best Algo which suit best for our data set

model = [DecisionTreeRegressor(),KNeighborsRegressor(),AdaBoostRegressor(),LinearRegression(),GradientBoostingRegressor()]
max_r2_score = 0
for r_state in range(0,100):
    train_x,test_x,train_y,test_y = train_test_split(x,y,random_state = r_state,test_size = 0.2)
    for i in model:
        i.fit(train_x,train_y)
        pre = i.predict(test_x)
        r2_sc = r2_score(test_y,pre)
        print("R2 score correspond to random state " ,r_state ,"is", r2_sc)
        if r2_sc> max_r2_score:
            max_r2_score=r2_sc
            final_state = r_state
            final_model = i

print()
print()
print()
print()
print("max R2 score correspond to random state " ,final_state , "is" , max_r2_score ,"and model is",final_model)
```

```
R2 score correspond to random state  0 is 0.7565201676441533
R2 score correspond to random state  0 is 0.5859495608097537
R2 score correspond to random state  0 is 0.017083517862649722
R2 score correspond to random state  0 is 0.48830824158733055
R2 score correspond to random state  0 is 0.7803888518105803
R2 score correspond to random state  1 is 0.8045801024850422
R2 score correspond to random state  1 is 0.626052838682385
R2 score correspond to random state  1 is 0.2623134774984702
R2 score correspond to random state  1 is 0.5245374549999047
```

```
max R2 score correspond to random state  77 is 0.8875102640018068 and model is DecisionTreeRegressor()
```

```
scalar=StandardScaler()
x_scaled=scalar.fit_transform(x)
```

```
In [73]: x_train,x_test,y_train,y_test=train_test_split(x_scaled,y,test_size=0.2,random_state=94)
```

```
In [74]: dt=DecisionTreeRegressor()
```

```
In [75]: dt.fit(x_train,y_train)
Out[75]: DecisionTreeRegressor()
```

```
In [76]: pred_test=dt.predict(x_test)
         print(r2_score(y_test,pred_test))

0.8967433411438777
```

So we have trained our dataset into various algorithm and we got a maximum accuracy for the random state of 77 is 88.75% for the DecisionTreeRegressor().we have also done prediction for test data which is 89.74%

14) cross validation:

```
In [77]: from sklearn.model_selection import cross_val_s
```

```
In [78]: for i in range(2,10):
             cv=cross_val_score(dt,x,y,cv=i)
             print(dt,cv.mean())
```

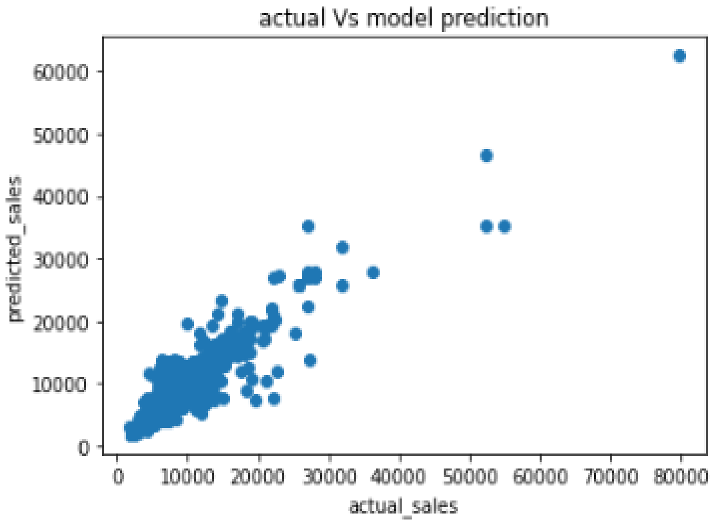
```
DecisionTreeRegressor() 0.7961669536526234
DecisionTreeRegressor() 0.7957039380177101
DecisionTreeRegressor() 0.8200252618394204
DecisionTreeRegressor() 0.8221751006631483
DecisionTreeRegressor() 0.8171382016528614
DecisionTreeRegressor() 0.8159273976953505
DecisionTreeRegressor() 0.8360975275781157
DecisionTreeRegressor() 0.8347744775967549
```

We have also done a cross validation of model it is also showing a better results

15)Scatter Plot:

Now lets we will plot a scatter plot to visualize the actual VS predicted price of data

```
In [80]: plt.scatter(y_test,y_pred)
         plt.xlabel('actual_price')
         plt.ylabel('predicted_price')
         plt.title('actual Vs model prediction')
         plt.show()
```



So it shows that the actual and predicted result are close to each other

16) model saving:

model saving

```
[81]: import joblib
```

```
[82]: joblib.dump(dt,'flight_price_prediction')
```

```
:[82]: ['flight_price_prediction']
```

So lets we have saved model which we are giving the best accuracy

17)conclusion:

```
In [130]: #load the saved model
flight_price=joblib.load('flight_price_prediction')
prices=flight_price.predict(testdata)
prices
```

```
Out[130]: array([5583., 5583., 5583., ..., 5583., 8452., 7804.])
```

We observed that data was filled with some outliers it is removed, also there some encoding is done now data is uniformly distributed in column. it is also observed from subplot, we have saved model and also predicted the result

with help of saved model .model is ready for the future data prediction.