Blog on:

Predicting the Flight Price

Submitted by:

Archana kumari

Batch : 1833

Problem Framing

Predicting the flight ticket's price is a difficult task as there is continuous change in these prices based on various conditions as well as on availability of passengers. Most of the people who are purchasing the flight tickets think that these prices are so unpredictable.

In this blog-post, I will go through the whole process of building machine learning model using different algorithms on Flight_Ticket_Participant_Datasets

This dataset is having two different excel files for training and testing data. We need to build our model using training dataset and have to predict flight prices for test dataset.

Size of training set: 10683 records

Size of test set: 2671 records

Importing Libraries

```
# data processing
import pandas as pd
# linear algebra
import numpy as np
# data visualization
import matplotlib.pyplot as plt
import seaborn as sns

#scaling
from sklearn.preprocessing import StandardScaler

#model selection
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.model_selection import KFold, cross_val_score

#model evaluation
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

Loading the Data

```
#lets import the dataset
train = pd.read_excel("Data_Train.xlsx")
test = pd.read_excel("Test_set.xlsx")

print('Shape of train dataset:',train.shape)
print('Shape of test dataset:',test.shape)
```

```
Shape of train dataset: (10683, 11)
Shape of test dataset: (2671, 10)
```

I have loaded both data sets from Excel files as 'train' and 'test'. Looking at the shapes of both datasets, train has 10683 rows and test has 2671 rows. 'test' dataset is not having column for target variable, which we need to predict hence it is having one less column than 'train' dataset.

I have loaded both data sets from Excel files as 'train' and 'test'. Looking at the shapes of both datasets, train has 10683 rows and test has 2671 rows. 'test' dataset is not having column for

target variable, which we need to predict hence it is having one less column than 'train' dataset.

```
#lets add source column to train and test dataset
train["source"] = "train"
test["source"] = "test"

#lets combine both the datasets
df = pd.concat([train,test],ignore_index=True)
df
```

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price	source
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	non-stop	No info	3897.0	train
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	2 stops	No info	7662.0	train
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	19h	2 stops	No info	13882.0	train
3	IndiGo	12/05/2019	Kolkata	Banglore	CCU → NAG → BLR	18:05	23:30	5h 25m	1 stop	No info	6218.0	train
4	IndiGo	01/03/2019	Banglore	New Delhi	BLR → NAG → DEL	16:50	21:35	4h 45m	1 stop	No info	13302.0	train
...	...	...	...	...	...	...	...	...	...	...	...	...
13349	Air India	6/06/2019	Kolkata	Banglore	CCU → DEL → BLR	20:30	20:25 07 Jun	23h 55m	1 stop	No info	NaN	test
13350	IndiGo	27/03/2019	Kolkata	Banglore	CCU → BLR	14:20	16:55	2h 35m	non-stop	No info	NaN	test
13351	Jet Airways	6/03/2019	Delhi	Cochin	DEL → BOM → COK	21:50	04:25 07 Mar	6h 35m	1 stop	No info	NaN	test
13352	Air India	6/03/2019	Delhi	Cochin	DEL → BOM → COK	04:00	19:15	15h 15m	1 stop	No info	NaN	test
13353	Multiple carriers	15/06/2019	Delhi	Cochin	DEL → BOM → COK	04:55	19:15	14h 20m	1 stop	No info	NaN	test

13354 rows × 12 columns

For applying all processing and analyzing to whole data we have combined both training and testing datasets. And to recognize them I am adding source column to both data sets as shown in above codes.

FEATURES:

Airline: The name of the airline.

Date_of_Journey: The date of the journey

Source: The source from which the service begins.

Destination: The destination where the service ends.

Route: The route taken by the flight to reach the destination.

Dep_Time: The time when the journey starts from the source.

Arrival_Time: Time of arrival at the destination.

Duration: Total duration of the flight.

Total_Stops: Total stops between the source and destination.

Additional_Info: Additional information about the flight

Price: The price of the ticket (this is our target variable)

Our target variable is Price and as it contains continuous data this is a regression problem. Rest all columns will act as features here.

```
#Let's check the null values in the dataset
df.isnull().sum()
```

```
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              2671
source                0
dtype: int64
```

We are having only single null values in the Route and Total_Stops columns, the null values presents in price column are because test data set is not having column for Price as we need to predict that.

```
#Lets chcek the datatypes of the columns
df.dtypes
```

```
Airline            object
Date_of_Journey    object
Source             object
Destination        object
Route              object
Dep_Time           object
Arrival_Time       object
Duration           object
Total_Stops         object
Additional_Info     object
Price              float64
source              object
dtype: object
```

I observe that Date_of_Journy , De_Time and Arrival_Time contains dates and time but their data type is object, lets convert it to datetime

2. Data Processing

```
#lets convert data type to datetime
df['Date_of_Journey'] =  pd.to_datetime(df['Date_of_Journey'])
df['Dep_Time'] =  pd.to_datetime(df['Dep_Time'])
df['Arrival_Time'] =  pd.to_datetime(df['Date_of_Journey'])
```

In data types I observe that columns like Date_of_Journy , Dep_Time and Arrival_Time contains dates and time but their data type is object, so I am converting these columns to datetime type.

```
#Check the data types again
df.dtypes
```

```
Airline                    object
Date_of_Journey    datetime64[ns]
Source                     object
Destination                object
Route                      object
Dep_Time           datetime64[ns]
Arrival_Time       datetime64[ns]
Duration                   object
Total_Stops                object
Additional_Info            object
Price                     float64
source                     object
dtype: object
```

Great we have successfully converted the data types of columns **Date_of_Journey , Dep_Time and Arrival_Time** into datetime type.

Filling the missing values

```
df['Route'].fillna(df['Route'].mode().iloc[0], inplace = True)
df['Total_Stops'].fillna(df['Total_Stops'].mode().iloc[0], inplace =
True)
```

As these are categorical columns I am replacing null values from these columns with the mode of that particular column. After filling all the null values I have done some data engineering here.

Date_of_Journey

```
df["Journey_day"] = pd.to_datetime(df.Date_of_Journey,
format="%d/%m/%Y").dt.day
df["Journey_month"] = pd.to_datetime(df["Date_of_Journey"], format =
"%d/%m/%Y").dt.month
df["Journey_year"] = pd.to_datetime(df["Date_of_Journey"], format =
"%d/%m/%Y").dt.year
df.drop(columns = "Date_of_Journey", inplace = True)
```

creating new columns using "Date_of_Journey " column as "Journey_day", "Journey_month" and "Journey_year" separately for better results in our model. And as I have derived new columns for "Date_of_Journey"; I will drop this column from the dataset.

Duration

```
#Getting Duration column using Arrival_Time and Dep_Time
x = (df["Arrival_Time"]-df["Dep_Time"])
duration_list = list()
for i in range(len(x)):
    dur = x.iloc[i].seconds/3600
    duration_list.append(dur)
df["Duration"] = duration_list
```

We already have a column for flight duration; that is "Duration", but it is given in hours and minutes separately as a object type. So I have decided to calculate the duration of flight by making use of "Arrival_Time" and "Dep_Time" (Flight duration = Arrival time-Departure time). And will update the data in the "Duration" column with these new numerical values.

Dep_Time

```
# Extracting Hours
df["Dep_hour"] = pd.to_datetime(df["Dep_Time"]).dt.hour

# Extracting Minutes
df["Dep_min"] = pd.to_datetime(df["Dep_Time"]).dt.minute

df["Dep_time"] = df["Dep_hour"] + df["Dep_min"]/60

# Now we can drop Dep_Time as it is of no use
df.drop(["Dep_Time"], axis = 1, inplace = True)
```

Similar to column "Duration"; "Dep_Time" is also in datetime type. Departure time is the time when a plane leaves the station. I am fetching hours and minutes separately from column "Dep_Time" and using these two columns calculate time of departure in numerical value in column "Dep_time" and will drop earlier column for departure time

Arrival_Time

Now we Know journey day and duration of flight, and we know time of arrival depends on these two factors so we can drop Arrival_time column as those two features will carry the information related "Arrival_time".

```
#lets drop Arrival_Time column
df.drop(columns = "Arrival_Time", inplace = True)

#Lets check the dataset after updating
df.head()
```

	Airline	Source	Destination	Route	Duration	Total_Stops	Additional_Info	Price	source	Journey_day	Journey_month	Journey_year	Dep_hour	D
0	IndiGo	Banglore	New Delhi	BLR → DEL	1.666667	non-stop	No info	3897.0	train	24	3	2019	22	2
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	18.166667	2 stops	No info	7662.0	train	5	1	2019	5	5
2	Jet Airways	Delhi	Cochin	DEL → LKO → BOM → COK	14.583333	2 stops	No info	13882.0	train	6	9	2019	9	2
3	IndiGo	Kolkata	Banglore	CCU → NAG → BLR	5.916667	1 stop	No info	6218.0	train	5	12	2019	18	5
4	IndiGo	Banglore	New Delhi	BLR → NAG → DEL	7.166667	1 stop	No info	13302.0	train	3	1	2019	16	5

By using Departure time column we created two seperates columns with Hours and minutes data, and using these two columns we created **Dep_time** column, which contains float value, so we can drop Dep_hour & Dep_min

```
df.drop(columns = ['Dep_hour','Dep_min'], inplace = True)
```

Journey_year

The column "Journey_year" is derived from column "Date_of_Journey". Let's check the count of this column.

```
df['Journey_year'].value_counts()
```

```
2019    13354
Name: Journey_year, dtype: int64
```

I can say that the column "Journey_year" has single value throughout the data. It means all the data collected in this dataset is from the year 2019. So this column will not have any impact on our target variable. I will delete this column.

```
df.drop(columns = 'Journey_year', inplace = True)
```

Let’s have a look at the data now

```
#let’s check the data after updation
df.head()
```

	Airline	Source	Destination	Route	Duration	Total_Stops	Additional_Info	Price	source	Journey_day	Journey_month	Dep_time
0	IndiGo	Banglore	New Delhi	BLR → DEL	1.666667	non-stop	No info	3897.0	train	24	3	22.333333
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	18.166667	2 stops	No info	7662.0	train	5	1	5.833333
2	Jet Airways	Delhi	Cochin	DEL → LKO → BOM → COK	14.583333	2 stops	No info	13882.0	train	6	9	9.416667
3	IndiGo	Kolkata	Banglore	CCU → NAG → BLR	5.916667	1 stop	No info	6218.0	train	5	12	18.083333
4	IndiGo	Banglore	New Delhi	BLR → NAG → DEL	7.166667	1 stop	No info	13302.0	train	3	1	16.833333

```
df.dtypes
```

```
Airline             object
Source              object
Destination         object
Route               object
Duration           float64
Total_Stops         object
Additional_Info     object
Price              float64
source              object
Journey_day          int64
Journey_month        int64
Dep_time           float64
dtype: object
```

Great we have left with 7 columns with object type of data. Let's check counts from these columns.

By checking the value counts of every categorical columns I observe that some modification is needed.

Additional_Info

By checking the value counts of every categorical column we will do some modifications
In column Additional_Info We will combine **1 Long layover** and **2 Long layover** with **Long layover** And **No Info** with **No info**

```
df["Additional_Info"].replace("1 Long layover","Long
layover",inplace=True)

df["Additional_Info"].replace("2 Long layover","Long
layover",inplace=True)
df["Additional_Info"].replace("No Info","No info",inplace=True)
```

Airline

In column Airline we will combine **Jet Airways Business** with **Jet Airways**. **Multiple carriers Premium economy** with **Multiple carriers**. **Vistara Premium economy** with **Vistara**

```
df["Airline"].replace("Jet Airways Business","Jet Airways",inplace=True)

df["Airline"].replace("Multiple carriers Premium economy","Multiple
carriers",inplace=True)

df["Airline"].replace("Vistara Premium economy","Vistara",inplace=True)
```

Destination

In "Destination" column 'New Delhi' can be replaced with 'Delhi'.

```
df["Destination"].replace("New Delhi","Delhi",inplace=True)
```

Total_Stops

The column "Total_Stops" have data of number of stops in the journey of particular flight. checking for value count

```
#lets check total stops
df["Total_Stops"].value_counts()
```

```
1 stop      7057
non-stop    4340
2 stops     1899
3 stops       56
4 stops        2
```

I can say the class 'non-stop' means there is no stop (0 stop) in the journey of flight. And it is the case of ordinal categorical type. I will replace these class with numerical values

```
df.replace({"non-stop": 0,"1 stop": 1,"2 stops": 2,"3 stops": 3,"4
stops": 4},inplace = True)
```

Let's check the data again after updation

```
# Check data
df.head()
```

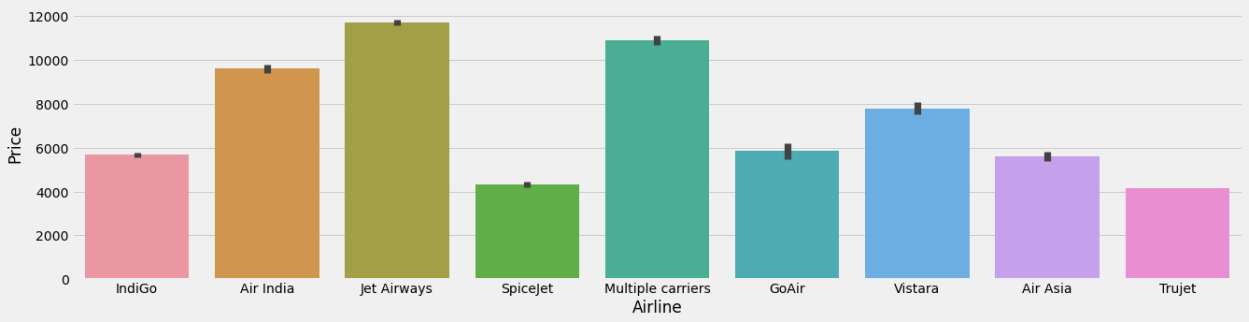
	Airline	Source	Destination	Route	Duration	Total_Stops	Additional_Info	Price	source	Journey_day	Journey_month	Dep_time
0	IndiGo	Banglore	Delhi	BLR → DEL	1.666667	0	No info	3897.0	train	24	3	22.333333
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	18.166667	2	No info	7662.0	train	5	1	5.833333
2	Jet Airways	Delhi	Cochin	DEL → LKO → BOM → COK	14.583333	2	No info	13882.0	train	6	9	9.416667
3	IndiGo	Kolkata	Banglore	CCU → NAG → BLR	5.916667	1	No info	6218.0	train	5	12	18.083333
4	IndiGo	Banglore	Delhi	BLR → NAG → DEL	7.166667	1	No info	13302.0	train	3	1	16.833333

Great after doing data processing and data cleaning we are now with better form of our data for our model. I will do encoding for remaining object type of features.

3. Visualization / EDA

Airline

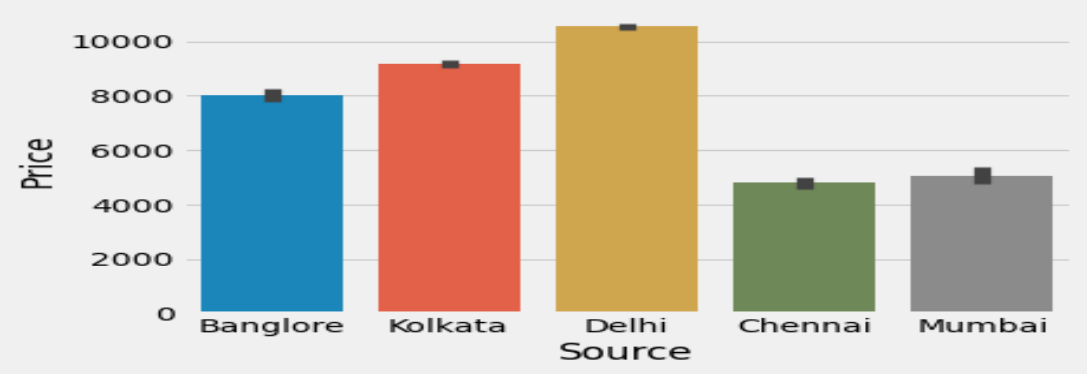
```
plt.style.use('fivethirtyeight')
plt.figure(figsize=(20,5))
sns.barplot(x = "Airline", y = "Price", data=df)
plt.show()
```



- By above plot we can say that the Jet Airways is most expensive airline, followed by Multiple carriers Airline and Air India.
- Spicejet and Trujet Airlines are cheaper compared to others.

Source

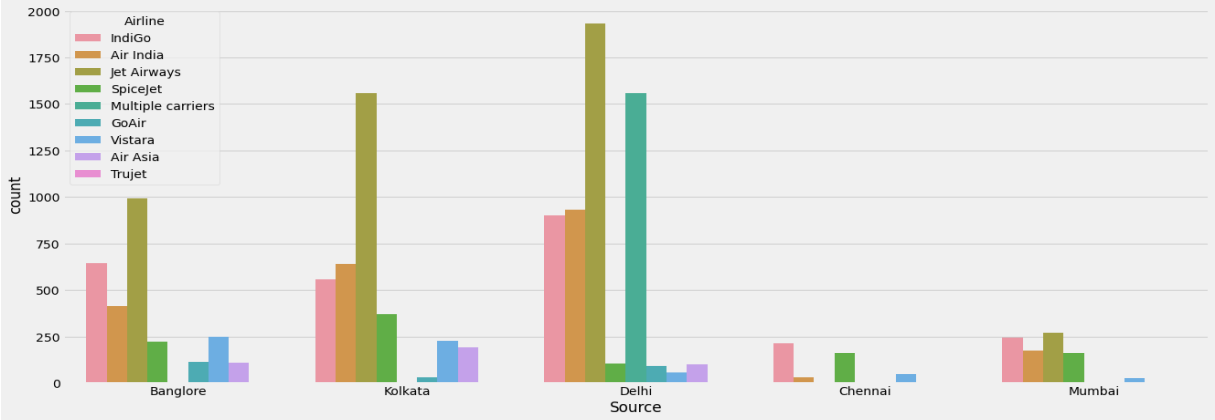
```
sns.barplot(x = "Source", y= "Price", data = df)
plt.show()
```



Average prices are higher at Delhi region compared to others and cheaper at Chennai and Mumbai. Kolkata and Banglore also having prices more than around 8000.

Airline

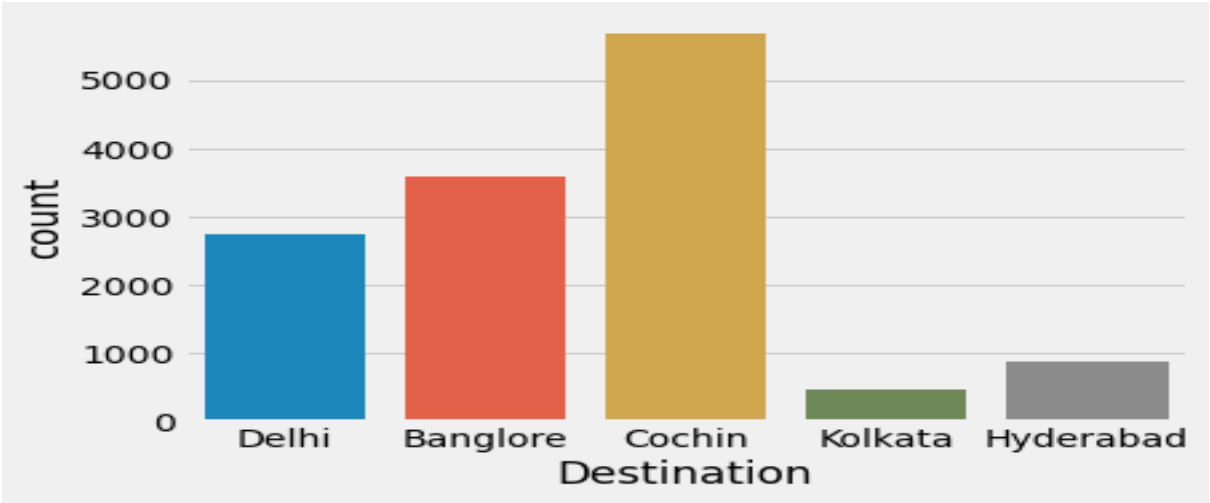
```
plt.figure(figsize=(20,8))
sns.countplot(x = "Source", hue = "Airline", data = df)
plt.show()
```



We can say that the Jet Airways airline is much popular than others in every region except in Chennai. And Multiple carrier airlines is only associates with Delhi region.

Destination

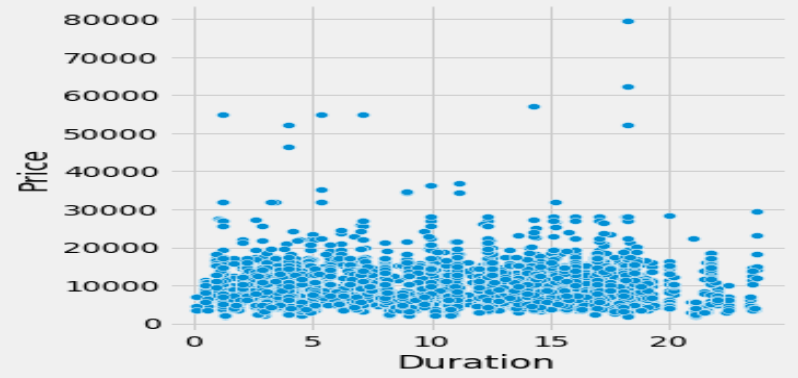
```
#lets see the counts of destinations
sns.countplot(df['Destination'])
plt.show()
```



As shown in this plot large number of flight's destination is cochin, and very few flight are going to Kolkata and Hyderabad.

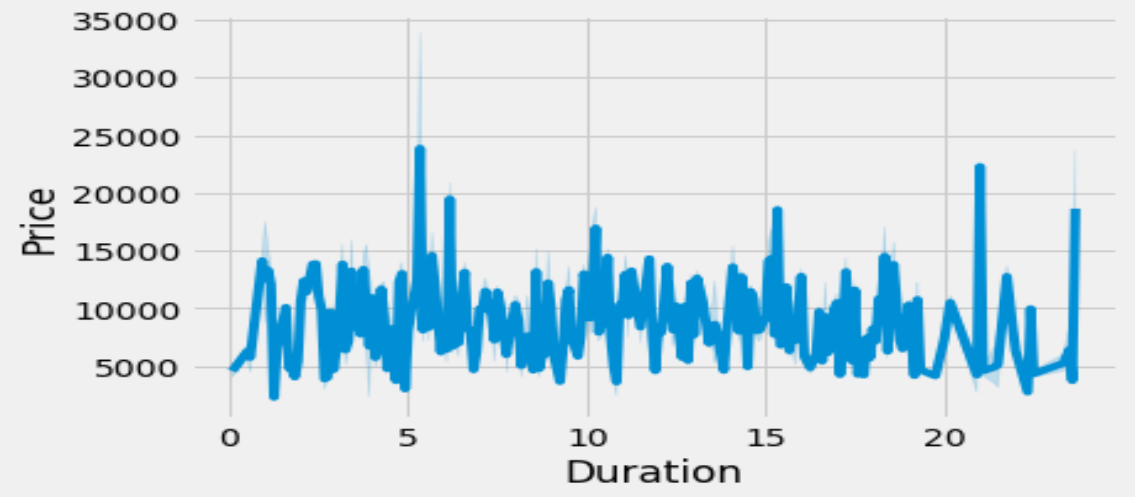
Duration

```
#lets check the relation between Duration and price
sns.relplot(x = 'Duration', y = 'Price', data = df)
plt.show()
```



Looking at above plot it is difficult to conclude anything.

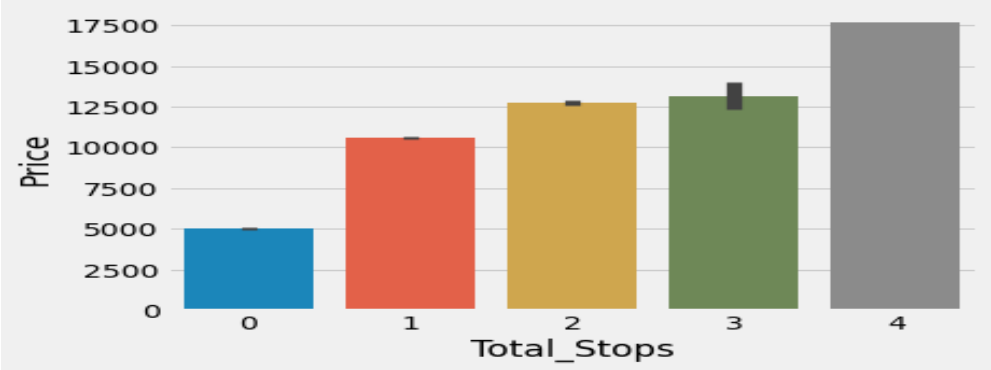
```
#lets plot a line plot for duration and price
sns.lineplot(x = "Duration", y = "Price", data = df)
plt.show()
```



As we see that Duration has not much impact on price of flight.

Total_Stops

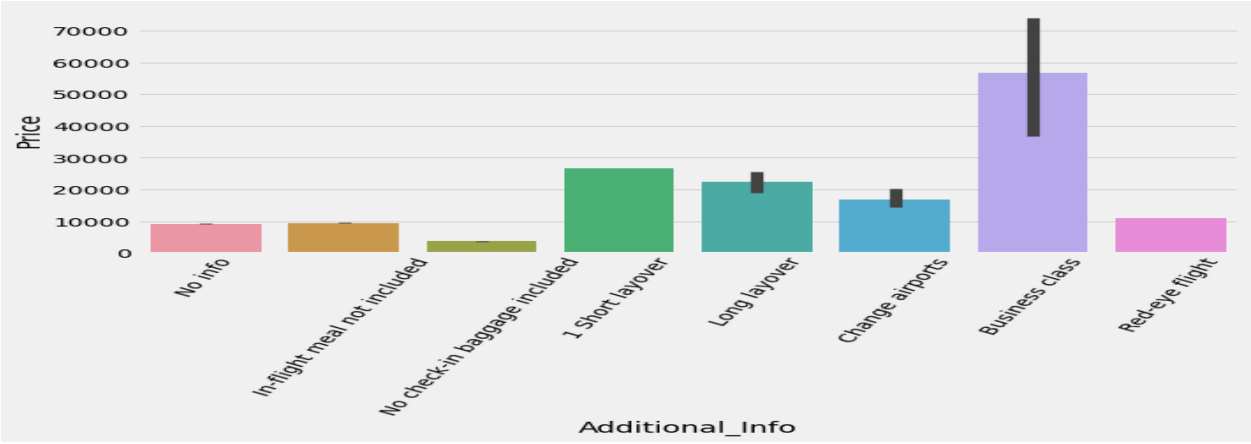
```
# lets plot barplot for total stops vs price
sns.barplot(x = 'Total_Stops', y = 'Price', data = df)
plt.show()
```



Flights which are with more stops are expensive compared to flights having less stops. we can see flight with 4 stops has higher price compared to others, and flights with 0 stops(that is non-stop) is having less price compared to others.

Additional_Info

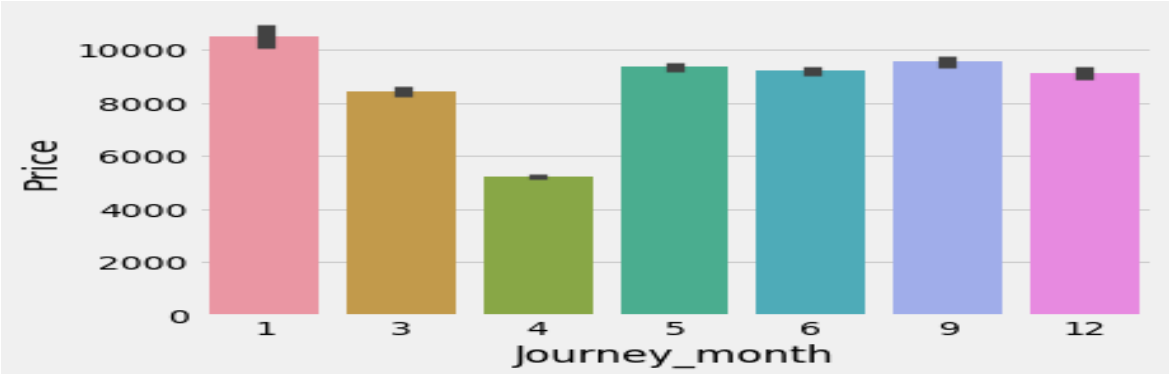
```
plt.figure(figsize = (10,5))
sns.barplot(x = 'Additional_Info', y = 'Price', data = df)
plt.xticks(rotation = 60)
plt.show()
```



- This will tell us that the Business class flights are much expencive than others, and the flight with No check-in baggage included class has least price.

Journey_month

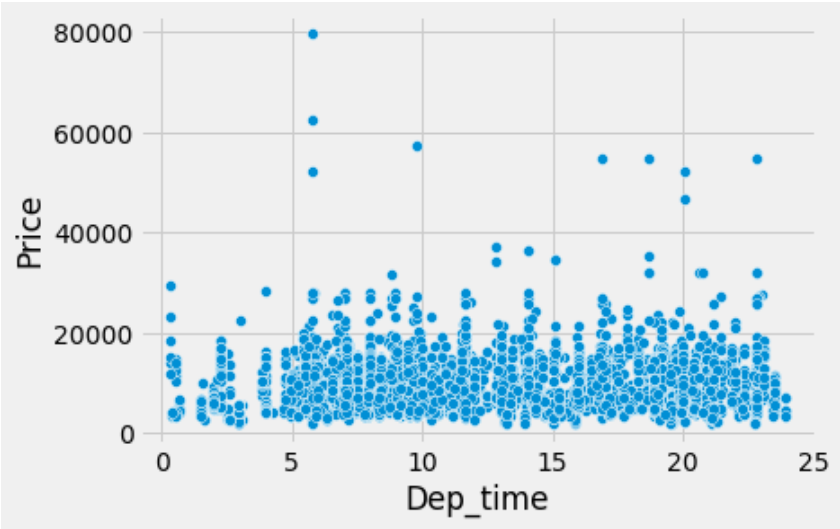
```
sns.barplot(x = 'Journey_month', y = 'Price', data = df)
plt.show()
```



Looking at above plot we can conclude that the flights from the month of april has less price compared to other months. and flights in the month of January are much expensive than others.

Dep_time

```
#Lets check the relation between Dep_time and price
sns.scatterplot(x = 'Dep_time', y = 'Price', data = df)
plt.show()
```



- By seeing above plot we can say there is no any fix relation between Dep_time and flight price.

```
#lets check the description
df.describe()
```

	Duration	Total_Stops	Price	Journey_day	Journey_month	Dep_time
count	13354.000000	13354.000000	10683.000000	13354.000000	13354.000000	13354.000000
mean	11.078291	0.826045	9087.064121	12.551146	5.549274	12.921709
std	5.736488	0.674608	4611.359167	8.759967	2.998690	5.736488
min	0.083333	0.000000	1759.000000	3.000000	1.000000	0.333333
25%	5.916667	0.000000	5277.000000	5.000000	3.000000	8.000000
50%	12.083333	1.000000	8372.000000	6.000000	5.000000	11.916667
75%	16.000000	1.000000	12373.000000	21.000000	6.000000	18.083333
max	23.666667	4.000000	79512.000000	27.000000	12.000000	23.916667

Looking at the data count in description it is ensured that there is no any null values in data. And as features are derived from object type data, we don't see any outlier in any feature.

Count is less only in price column because these values are only from train dataset, test dataset is not having column for Price.

4. EDA Concluding Remarks

We have done EDA after doing data processing. After analyzing the data we found that we are having many categorical features for those we need to do encoding.

Some features which are not relatable to our target variable are already dropped during data processing.

And there are outliers present in our dataset, we will remove them by z-score method

All features are derived from categorical features hence we don’t need to scale down our data.

5. Pre-processing Pipeline

Encoding

To check correlation among every feature and further requirement for model building; I am encoding the categorical features using OrdinalEncoder.

```
#lets convert categorical data into numeric values, using
OrdinalEncoder
from sklearn.preprocessing import OrdinalEncoder
enc = OrdinalEncoder()
for i in df.columns:
    if df[i].dtypes == "object" :
        df[i] = enc.fit_transform(df[i].values.reshape(-1,1))
```

Lets have a look on data after encoding

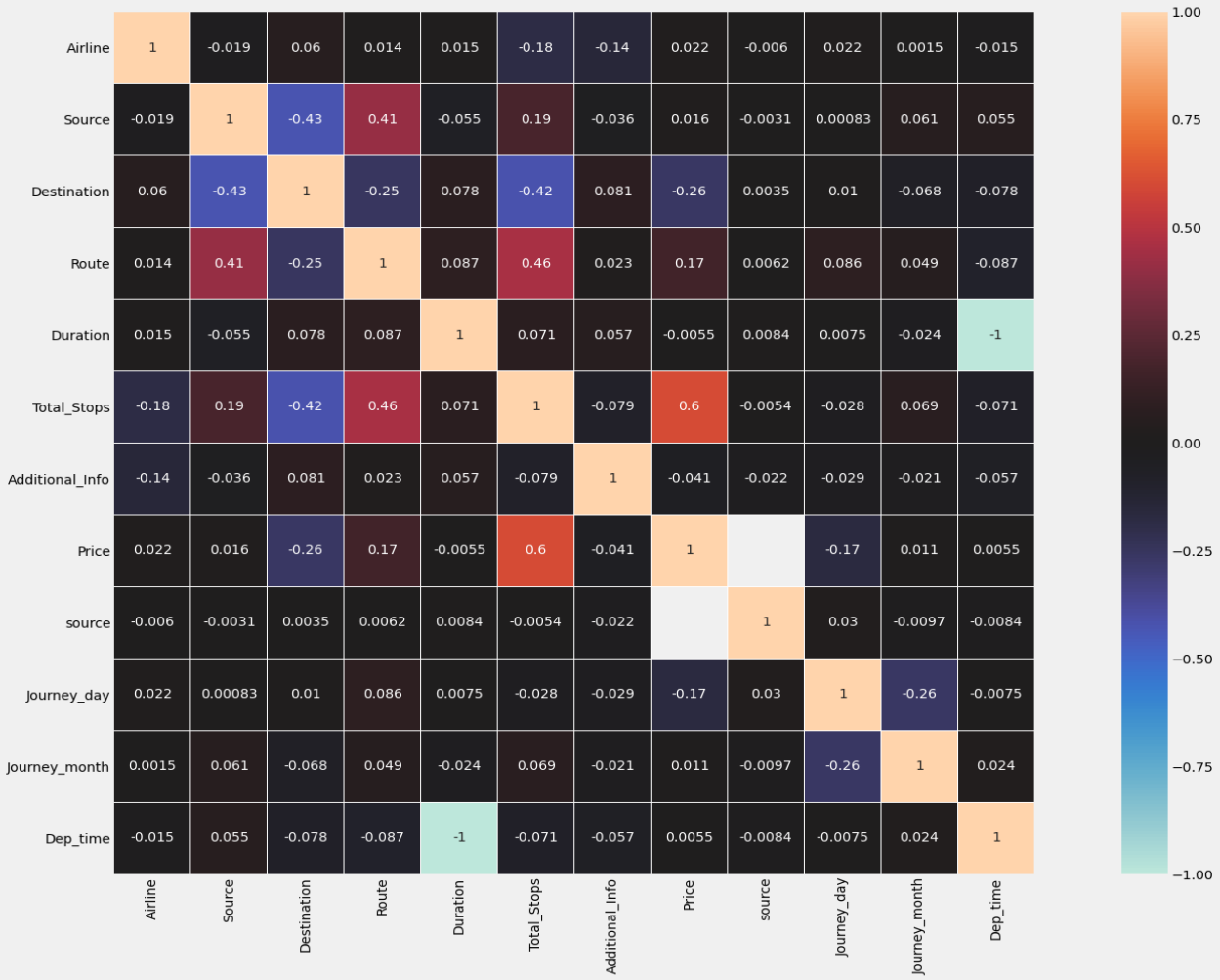
```
df.head()
```

	Airline	Source	Destination	Route	Duration	Total_Stops	Additional_Info	Price	source	Journey_day	Journey_month	Dep_time
0	3.0	0.0	2.0	18.0	1.666667	0	6.0	3897.0	1.0	24	3	22.333333
1	1.0	3.0	0.0	87.0	18.166667	2	6.0	7662.0	1.0	5	1	5.833333
2	4.0	2.0	1.0	122.0	14.583333	2	6.0	13882.0	1.0	6	9	9.416667
3	3.0	3.0	0.0	95.0	5.916667	1	6.0	6218.0	1.0	5	12	18.083333
4	3.0	0.0	2.0	29.0	7.166667	1	6.0	13302.0	1.0	3	1	16.833333

Great the categorical data is successfully encoded into numerical data.

Heat map for checking correlation

```
#Lets plot heatmap to check correlation among differnt features and
label
df_corr = df.corr()
plt.figure(figsize = (25,15))
sns.heatmap(df_corr,vmin=-
1,vmax=1,annot=True,square=True,center=0,fmt='.2g',linewidths=0.1)
plt.tight_layout()
```

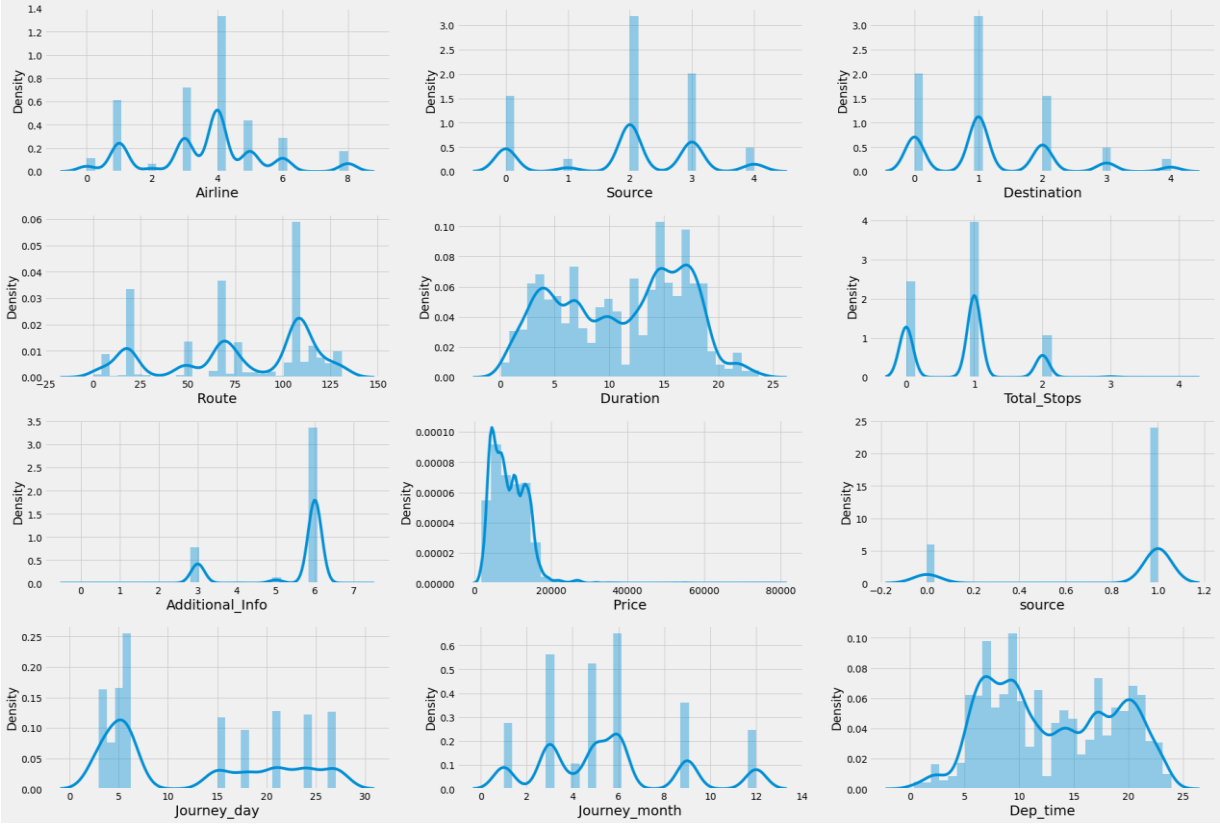


By looking at the heat map we can say Total_stops has maximum correlation with price.

Duration and Dep_time are having very less correlation with price, but I don't want to lose this information so I decided to not to drop these columns, and these two columns are perfectly in negative correlation with each other.

Check for Outliers

```
#Lets have a look on distribution of our data
plt.figure(figsize = (25,20))
plotnumber = 1
for column in df.columns:
    if plotnumber <= 12:
        ax = plt.subplot(5,3,plotnumber)
        sns.distplot(df[column], bins=30)
        plt.xlabel(column,fontsize = 20)
    plotnumber+=1
plt.tight_layout()
```



By looking at the above distribution plots we can see skewness in many columns, but these are categorical data, we can remove outliers from Duration and Dep_time as these are derived from datetime types.

Outliers removing

```
from scipy import stats
from scipy.stats import zscore
z_score = zscore(df[["Duration","Dep_time"]])
abs_z_score = np.abs(z_score)
filtering_entry = (abs_z_score < 3).all(axis = 1)
df = df[filtering_entry]
df.reset_index(inplace = True)

#lets check the shape after removal of outliers
df.shape
```

```
(13354, 13)
```

Great after applying z-score for removing outliers we get the data as it is as earlier. It means there is no any outlier present in our data set.

As our data is categorical so I am not treating the skewness.

Now I will divide the dataset into train and test again to build machine learning model using **train** dataset

Devide train and test Data sets

I will separate our train and test datasets by using **'source'** column

```
#Divide into test and train:
df_train = df.loc[df['source']== 1]
df_test = df.loc[df['source']== 0]

#lets have a look at our training data set
df_train
```

	Airline	Source	Destination	Route	Duration	Total_Stops	Additional_Info	Price	source	Journey_day	Journey_month	Dep_time
0	3.0	0.0	2.0	18.0	1.666667	0	6.0	3897.0	1.0	24	3	22.333333
1	1.0	3.0	0.0	87.0	18.166667	2	6.0	7662.0	1.0	5	1	5.833333
2	4.0	2.0	1.0	122.0	14.583333	2	6.0	13882.0	1.0	6	9	9.416667
3	3.0	3.0	0.0	95.0	5.916667	1	6.0	6218.0	1.0	5	12	18.083333
4	3.0	0.0	2.0	29.0	7.166667	1	6.0	13302.0	1.0	3	1	16.833333
...	...	...	...	...	...	...	...	...	...	...	...	...
10678	0.0	3.0	0.0	67.0	4.083333	0	6.0	4107.0	1.0	4	9	19.916667
10679	1.0	3.0	0.0	67.0	3.250000	0	6.0	4145.0	1.0	27	4	20.750000
10680	4.0	0.0	2.0	18.0	15.666667	0	6.0	7229.0	1.0	27	4	8.333333
10681	8.0	0.0	2.0	18.0	12.500000	0	6.0	12648.0	1.0	3	1	11.500000
10682	1.0	2.0	1.0	112.0	13.083333	2	6.0	11753.0	1.0	5	9	10.916667

10683 rows × 12 columns

After dividing these two datasets I will drop 'Price' column from test dataset as that is going to be our prediction. And will also drop 'source' column from both the datasets as we don't want that column now.

```
df_test.drop(columns=["Price"],inplace=True)

#drop source column from train and test
df_train.drop(columns=["source"],inplace=True)
df_test.drop(columns=["source"],inplace=True)
```

Separate features and label

```
#lets saperate data into label and features
x = df_train.drop(columns = 'Price')
y = df_train["Price"]

#check features
x.head()
```

	Airline	Source	Destination	Route	Duration	Total_Stops	Additional_Info	Journey_day	Journey_month	Dep_time
0	3.0	0.0	2.0	18.0	1.666667	0	6.0	24	3	22.333333
1	1.0	3.0	0.0	87.0	18.166667	2	6.0	5	1	5.833333
2	4.0	2.0	1.0	122.0	14.583333	2	6.0	6	9	9.416667
3	3.0	3.0	0.0	95.0	5.916667	1	6.0	5	12	18.083333
4	3.0	0.0	2.0	29.0	7.166667	1	6.0	3	1	16.833333

As we know all features are derived from object type; our data is already in good range, I think we don't need to do scaling. So I am not scaling the features here.

Find Best randomstate

Random state ensures that the splits that you generate are reproducible. Scikit-learn use random permutations to generate the splits. For finding the best random_state for our model I am making use of LinearRegression model. First we will find best random_state for LinearRegression and use same for different models.

```
#to find random stat which gives maximum r2_score
from sklearn.linear_model import LinearRegression
max_r_score=0
r_state = 0
for i in range(1,200):
    x_train, x_test, y_train, y_test = train_test_split(x, y,test_size
= 0.25,random_state = r_state)
    reg = LinearRegression()
    reg.fit(x_train,y_train)
    y_pred = reg.predict(x_test)
    r2_scr=r2_score(y_test,y_pred)
    if r2_scr > max_r_score:
        max_r_score = r2_scr
        r_state = i
print("max r2 score is",max_r_score,"on Random State",r_state)
```

```
max r2 score is 0.43676576401498246 on Random State 1
```

Great I got random_state 1 which is giving maximum accuracy score for LogisticRegression model. I will split our data into train and test based on this random_state.

```
#lets split our train data into train and test part
x_train, x_test, y_train, y_test = train_test_split(x, y,test_size =
0.25,random_state = 1)
```

6. Model Building with Evaluation

Evaluation metrics used

Cross-Validation

We are using K-Fold Cross-validation as a model evaluation metric. It is a procedure of resampling for limited data sample. The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into, and the model is trained and tested for every fold separately. We can take the mean of these scores and compared with our model's accuracy.

MAE : In machine learning Mean Absolute Error is an absolute average difference between prediction of data and actual data.

RMSE : Root Mean Square Error is the measure of how good a regression line fits the data. And it is one of the most commonly used evaluation metrics.

LinearRegression Model

```
#Model with LinearRegression
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(x_train, y_train)
y_pred = lr.predict(x_train)
pred_lr = lr.predict(x_test)

r2score = r2_score(y_test,pred_lr)*100

#evaluation
mse = mean_squared_error(y_test,pred_lr)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test,pred_lr)
print("MAE :", mae)
print("RMSE :", rmse)
print('------------------------------')

# r2 score
print("Training r2 Score :", r2_score(y_train,y_pred)*100,'%')
print(f"Testing r2 Score:", r2score,"%")
print('------------------------------')

#cross validation score
scores = cross_val_score(lr, x, y, cv = 10).mean()*100
print("\nCross validation score :", scores)

#result of accuracy minus cv score
result = r2score - scores
print("\nAccuracy Score - Cross Validation Score :", result)
```

```
MAE : 2499.8196481043337
RMSE : 3509.241319218368
------------------------------
Training r2 Score : 43.2699582512223 %
Testing r2 Score: 43.381670837215225 %
------------------------------

Cross validation score : 43.500461252993745

Accuracy Score - Cross Validation Score : -0.11879041577851979
```

DecisionTreeRegressor Model

```
from sklearn.tree import DecisionTreeRegressor
dt = DecisionTreeRegressor()
dt.fit(x_train, y_train)
y_pred = dt.predict(x_train)
pred_dt = dt.predict(x_test)

r2score = r2_score(y_test,pred_dt)*100

#evaluation
mse = mean_squared_error(y_test,pred_dt)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test,pred_dt)
print("MAE :", mae)
print("RMSE :", rmse)
print('------------------------------')

# r2 score
print("Training r2 Score :", r2_score(y_train,y_pred)*100,'%')
print(f"Testing r2 Score:", r2score,"%")
print('------------------------------')

#cross validation score
scores = cross_val_score(dt, x, y, cv = 10).mean()*100
print("\nCross validation score :", scores)

#result of accuracy minus cv score
result = r2score - scores
print("\nAccuracy Score - Cross Validation Score :", result)
```

```
 MAE : 824.0362397710862
RMSE : 2081.1960063670317
------------------------------
Training r2 Score : 97.28145099376324 %
Testing r2 Score: 80.0860896291779 %
------------------------------

Cross validation score : 82.9588279029864

Accuracy Score - Cross Validation Score : -2.872738273808494
```

RandomForestRegressor Model

```
from sklearn.ensemble import RandomForestRegressor

rf = RandomForestRegressor()
rf.fit(x_train, y_train)
y_pred = rf.predict(x_train)
pred_rf = rf.predict(x_test)

r2score = r2_score(y_test,pred_rf)*100

#evaluation
mse = mean_squared_error(y_test,pred_rf)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test,pred_rf)
print("MAE :", mae)
print("RMSE :", rmse)
print('------------------------------')

# r2 score
print("Training r2 Score :", r2_score(y_train,y_pred)*100,'%')
print(f"Testing r2 Score:", r2score,"%")
print('------------------------------')

#cross validation score
scores = cross_val_score(rf, x, y, cv = 10).mean()*100
print("\nCross validation score :", scores)

#result of accuracy minus cv score
result = r2score - scores
print("\nAccuracy Score - Cross Validation Score :", result)
```

```
 MAE : 752.8571762756441
RMSE : 1664.6101712088453
------------------------------
Training r2 Score : 96.05034664316213 %
Testing r2 Score: 87.26040363274366 %
------------------------------

Cross validation score : 87.82127262776822

Accuracy Score - Cross Validation Score : -0.5608689950245633
```

KNeighborsRegressor Model

```
from sklearn.neighbors import KNeighborsRegressor
knr =  KNeighborsRegressor()
knr.fit(x_train, y_train)
y_pred = knr.predict(x_train)
pred_knr = knr.predict(x_test)

r2score = r2_score(y_test,pred_knr)*100

#evaluation
mse = mean_squared_error(y_test,pred_knr)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test,pred_knr)
print("MAE :", mae)
print("RMSE :", rmse)
print('------------------------------')

# r2 score
print("Training r2 Score :", r2_score(y_train,y_pred)*100,'%')
print(f"Testing r2 Score:", r2score,"%")
print('------------------------------')

#cross validation score
scores = cross_val_score(knr, x, y, cv = 10).mean()*100
print("\nCross validation score :", scores)

#result of accuracy minus cv score
result = r2score - scores
print("\nAccuracy Score - Cross Validation Score :", result)
```

```
 MAE : 1695.7488581055784
RMSE : 2722.7742611066687
------------------------------
Training r2 Score : 77.92080276449548 %
Testing r2 Score: 65.91574126634276 %
------------------------------

Cross validation score : 67.13089506978675

Accuracy Score - Cross Validation Score : -1.2151538034439824
```

XGBRegressor Model

```
from xgboost import XGBRegressor
xgb = XGBRegressor(verbosity = 0)
xgb.fit(x_train,y_train)
y_pred = xgb.predict(x_train)
pred_xgb = xgb.predict(x_test)

r2score = r2_score(y_test,pred_xgb)*100

#evaluation
mse = mean_squared_error(y_test,pred_xgb)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test,pred_xgb)
print("MAE :", mae)
print("RMSE :", rmse)
print('------------------------------')

# r2 score
print("Training r2 Score :", r2_score(y_train,y_pred)*100,'%')
print(f"Testing r2 Score:", r2score,"%")
print('------------------------------')

#cross validation score
scores = cross_val_score(xgb, x, y, cv = 10).mean()*100
print("\nCross validation score :", scores)

#result of accuracy minus cv score
result = r2score - scores
print("\nAccuracy Score - Cross Validation Score :", result)
```

```
MAE : 784.5907380487178
RMSE : 1600.3481711168859
------------------------------
Training r2 Score : 94.92217403640676 %
Testing r2 Score: 88.22503740633914 %
------------------------------

Cross validation score : 88.50136859943431

Accuracy Score - Cross Validation Score : -0.2763311930951744
```

Looking at the results of each algorithm I am selecting RandomForestRegressor as a best suitable algorithm for our final model as it is giving least difference between accuracy score and CV-score and least mean absolute error as well.

What is RandomForest?

Random forest is method of ensembling for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time. For classification tasks, the output of the random forest is the class selected by most trees. For regression tasks, the mean or average prediction of the individual trees is returned. But the accuracy of random forests is less than that of __gradient boosted trees.

Hyperparameter Tuning

Below you can see the code of the hyperparameter tuning for the parameters like max_depth, n_estimators, min_samples_split. As random forest takes large amount of time for hyperparameter tuning I am using less parameters here.

```
#lets selects different parameters for tuning
grid_params = {
                'max_depth': [12,15,20,22],
                'n_estimators':[800,900,1000,1200],
                'min_samples_split': [2]
                }

#train the model with given parameters using GridSearchCV
GCV =  GridSearchCV(RandomForestRegressor(), grid_params, cv = 5)
GCV.fit(x_train,y_train)
```

```
GridSearchCV(cv=5, estimator=RandomForestRegressor(),
             param_grid={'max_depth': [12, 15, 20, 22],
                         'min_samples_split': [2],
                         'n_estimators': [800, 900, 1000, 1200]})
```

```
GCV.best_params_       #printing the best parameters
```

```
{'max_depth': 15, 'min_samples_split': 2, 'n_estimators': 800}
```

Great we have got our best parameters for our final model.

Final model

Test new parameters

Now I will train and test RandomForestRegressor again with these best parameters

```
model = RandomForestRegressor(max_depth = 15, min_samples_split = 2,
n_estimators = 800)
model.fit(x_train,y_train)
pred = model.predict(x_test)

r2score = r2_score(y_test,pred)*100

#evaluation
mse = mean_squared_error(y_test,pred)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test,pred)
print("MAE :", mae)
print("RMSE :", rmse)
print('------------------------------')

# r2 score

print(f" \nr2 Score:", r2score,"%")
```

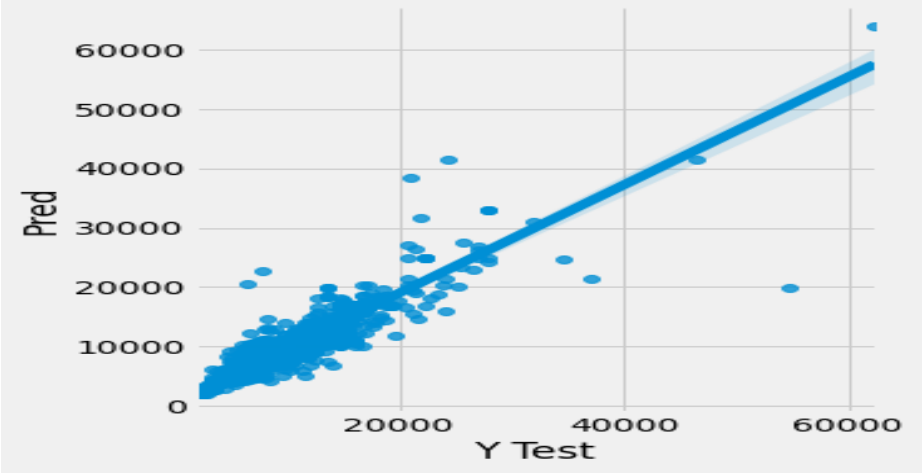
```
MAE : 737.6662172493312
RMSE : 1592.277530324997
------------------------------

r2 Score: 88.3435014639846 %
```

Nice! by doing hyperparameter tuning we have improved our final model r2-score from 87.66% to 88.33%. And also we got reduction in MAE and RMSE.

-Lets see final Actual Vs Predicted sample.

```
data = pd.DataFrame({'Y Test':y_test , 'Pred':pred},columns=['Y
Test','Pred'])
sns.lmplot(x='Y Test',y='Pred',data=data,palette='rainbow')
plt.show()
```



Predict flight price for test dataset using final model

Now using our best machine learning model I will predict the flight prices for test dataset and then will create a data frame with these predicted values. After that I will save this data frame into .csv file

```
#lets predict the price with our best model
prediction = model.predict(df_test)

#lets make the dataframe for prediction
flt_price = pd.DataFrame(pred, columns=["Price"])

#lets have a look at predicted prices
flt_price
```

	Price
0	2005.117083
1	5472.522363
2	14467.071026
3	19967.047846
4	10263.444478
...	...
2666	14527.450259
2667	10205.822587
2668	5652.360000
2669	4008.846517
2670	16893.486840

2671 rows × 1 columns

7. Conclusion

We started with data loading which includes loading train and test data files separately and combining these two in one dataset. After that we checked for missing values and filled that null values with suitable method. In data preprocessing we have changed the required feaure's data types and created new features using other columns also dropped unwanted columns from data set and replaced some of the entries with suitable element. After data processing we did visualization of the data using matplotlib and seaborn.

Then we removed outliers from our data and scaled down the numerical features and encoded categorical features using Ordinal Encoder. After that I have separated the train and test data to build our machine learning model using train dataset and using which we have to make prediction for test dataset. Then we build and tested for various machine learning models among which I have selected RandomForestRegressor as best suitable algorithm for final model and did hyperparameter tuning for getting best parameters. And with these best parameters we tested our final model and got improvement in our finale R2-score.

Lastly we have predicted flight prices for test dataset using our final model and saved in csv format as well. Below figure shows the dataframe for predicted values.

There is still scope for improvement, like taking real time data, doing a more feature engineering, by comparing and plotting the features against each other. And absolutely we can improve this r2 score more by doing more extensive hyperparameter tuning for different machine learning models.