



OPTIMIZING FLIGHT BOOKING DECISIONS THROUGH MACHINE LEARNING PRICE PREDICTIONS

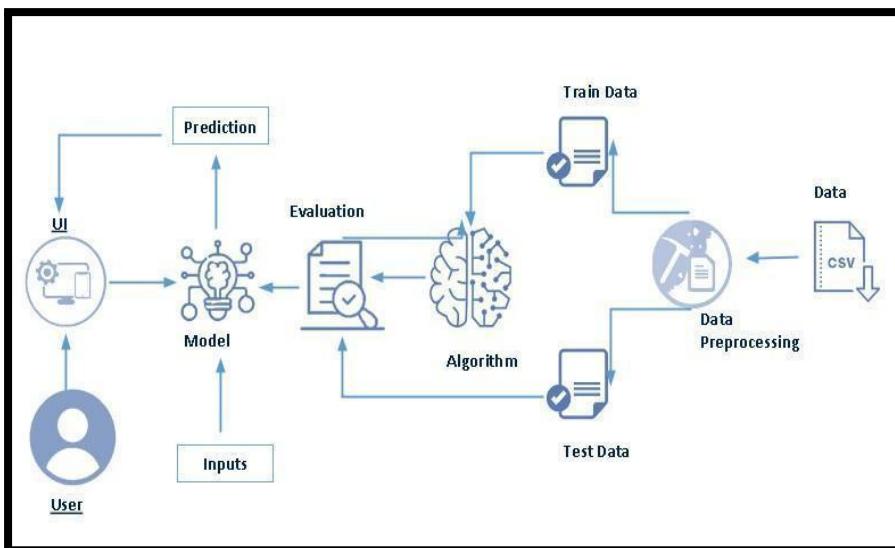
Project Based Experiential Learning Program

**Name:D.Balamurugan
Roll_NO:20S2CS06**

Optimizing Flight Booking Decisions through Machine Learning Price Predictions

People who work frequently travel through flight will have better knowledge on best discount and right time to buy the ticket. For the business purpose many airline companies change prices according to the seasons or time duration. They will increase the price when people travel more. Estimating the highest prices of the airlines data for the route is collected with features such as Duration, Source, Destination, Arrival and Departure. Features are taken from chosen dataset and in the price wherein the airline price ticket costs vary overtime. we have implemented flight price prediction for users by using KNN, decision tree and random forest algorithms. Random Forest shows the best accuracy of 80% for predicting the flight price. also, we have done correlation tests and metrics for the statistical analysis.

Technical Architecture:



Project Flow:

- User interacts with the UI to enter the input.

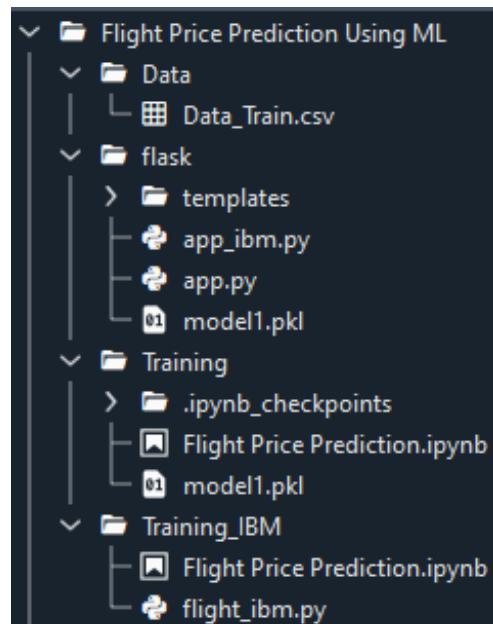
- Entered input is analysed by the model which is integrated.
- Once model analyses the input the prediction is showcased on the UI

To accomplish this, we have to complete all the activities listed below,

- Define Problem / Problem Understanding
 - Specify the business problem
 - Business requirements
 - Literature Survey
 - Social or Business Impact.
- Data Collection & Preparation
 - Collect the dataset
 - Data Preparation
- Exploratory Data Analysis
 - Descriptive statistical
 - Visual Analysis
- Model Building
 - Training the model in multiple algorithms
 - Testing the model
- Performance Testing & Hyperparameter Tuning
 - Testing model with multiple evaluation metrics
 - Comparing model accuracy before & after applying hyperparameter tuning
- Model Deployment
 - Save the best model
 - Integrate with Web Framework
- Project Demonstration & Documentation
 - Record explanation Video for project end to end solution
 - Project Documentation-Step by step project development procedure

Project Structure:

Create the Project folder which contains files as shown below



- We are building a flask application which needs HTML pages stored in the templates folder and a python script app.py for scripting.
- model1.pkl is our saved model. Further we will use this model for flask integration.
- Training folder contains model training files and training_ibm folder contains IBM deployment files.

Milestone 1: Define Problem / Problem Understanding

Activity 1: Specify the business problem

Refer Project Description

Activity 2: Business requirements

The business requirements for a machine learning model to predict personal loan approval include the ability to accurately predict loan approval based on applicant information, Minimise the number of false positives (approved loans that default) and false negatives (rejected loans that would have been successful). Provide an explanation for the model's decision, to comply with regulations and improve transparency.

Activity 3: Literature Survey (Student Will Write)

As the data is increasing daily due to digitization in the banking sector, people want to apply for loans through the internet. Machine Learning (ML), as a typical method for information investigation, has gotten more consideration increasingly. Individuals of various businesses are utilising ML calculations to take care of the issues dependent on their industry information. Banks are facing a significant problem in the approval of the loan. Daily there are so many applications that are challenging to manage by the bank employees, and also the chances of some mistakes are high. Most banks earn profit from the loan, but it is risky to choose deserving customers from the number of applications. There are various algorithms that have been used with varying levels of success. Logistic regression, decision tree, random forest, and neural networks have all been used and have been able to accurately predict loan defaults. Commonly used features in these studies include credit score, income, and employment history, sometimes also other features like age, occupation, and education level.

Activity 4: Social or Business Impact.

Social Impact: - Personal loans can stimulate economic growth by providing individuals with the funds they need to make major purchases, start businesses, or invest in their education.

Business Model/Impact: - Personal loan providers may charge fees for services such as loan origination, processing, and late payments. Advertising the brand awareness and marketing to reach out to potential borrowers to generate revenue.

Milestone 2: Data Collection & Preparation

ML depends heavily on data. It is the most crucial aspect that makes algorithm training possible. So this section allows you to download the required dataset.

Activity 1: Collect the dataset

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

In this project we have used .csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

Link: <https://www.kaggle.com/code/anshigupta01/flight-price-prediction/data>

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualisation techniques and some analysing techniques.

Note: There are a number of techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

Activity 1.1: Importing the libraries

Import the necessary libraries as shown in the image. (optional) Here we have used visualisation style as fivethirtyeight.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report, confusion_matrix
import warnings
import pickle
from scipy import stats
warnings.filterwarnings('ignore')
plt.style.use('fivethirtyeight')
```

Activity 1.2: Read the Dataset

Our dataset format might be in .csv, excel files,.txt,.json, etc. We can read the dataset with the help of pandas.

In pandas we have a function called `read_csv()` to read the dataset. As a parameter we have to give the directory of csv file.

```
data=pd.read_csv("Data_Train.csv")
```

```
data.head()
```

	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

Activity 2: Data Preparation

As we have understood how the data is let's pre-process the collected data.

The download data set is not suitable for training the machine learning model as it might have so much of randomness so we need to clean the dataset properly in order to fetch good results. This activity includes the following steps.

- Handling missing values
- Handling categorical data
- Handling outliers
- Scaling Techniques
- Splitting dataset into training and test set

Note: These are the general steps of pre-processing the data before using it for machine learning. Depending on the condition of your dataset, you may or may not have to go through all these steps.

We have 1 missing value in Route column, and 1 missing value in Total stops column. We will meaningfully replace the missing values going further.

We now start exploring the columns available in our dataset. The first thing we do is to create a list of categorical columns, and check the unique values present in these columns

```
for i in category:  
    print(i, data[i].unique())
```

```
Airline ['IndiGo' 'Air India' 'Jet Airways' 'SpiceJet' 'Multiple carriers' 'GoAir'  
'Vistara' 'Air Asia' 'Vistara Premium economy' 'Jet Airways Business'  
'Multiple carriers Premium economy' 'Trujet']  
Source ['Banglore' 'Kolkata' 'Delhi' 'Chennai' 'Mumbai']  
Destination ['New Delhi' 'Banglore' 'Cochin' 'Kolkata' 'Delhi' 'Hyderabad']  
Additional Info ['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']
```

- 1. Airline column has 12 unique values - 'IndiGo' , 'Air India', 'Jet Airways' , 'SpiceJet' , 'Multiple carriers' , 'GoAir', 'Vistara', 'Air Asia', 'Vistara Premium economy' , 'Jet Airways Business', 'Multiple carriers Premium economy', 'Trujet'.
2. Source column has 5 unique values – ‘Bangalore’, ‘Kolkata’, ‘Chennai’, ‘Delhi’ and ‘Mumbai’.
3. Destination column has 6 unique values - 'New Delhi', 'Banglore', 'Cochin', 'Kolkata', 'Delhi' , 'Hyderabad'.
4. Additional info column has 10 unique values - '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'.

We now split the Date column to extract the ‘Date’, ‘Month’ and ‘Year’ values, and store them in new columns in our dataframe.

```
#We now split the Date column to extract the 'Date', 'Month' and 'Year' values, and store them in new columns in our dataframe.
```

```
data.Date_of_Journey=data.Date_of_Journey.str.split('/')
```

```
data.Date_of_Journey
```

```
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: 10682, dtype: object
```

```
#Treating the data_column
```

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

- Further, we split the Route column to create multiple columns with cities that the flight travels through. We check the maximum number of stops that a flight has, to confirm what should be the maximum number of cities in the longest route

```
data.Total_Stops.unique()
```

```
array(['non-stop', '2 stops', '1 stop', '3 stops', '4 stops'],
      dtype=object)
```

- Since the maximum number of stops is 4, there should be maximum 6 cities in any particular route. We split the data in route column, and store all the city names in separate columns

```
#Since the maximum number of stops is 4, there should be maximum 6 cities in any particular route. We split the data in route col
data.Route=data.Route.str.split(',')
data.Route
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]
10681      [BLR ,  DEL]
10682  [DEL ,  GOI ,  BOM ,  COK]
Name: Route, Length: 10682, dtype: object
```

```
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]
```

- In the similar manner, we split the Dep_time column, and create separate columns for departure hours and minutes

#In the similar manner, we split the Dep_time column, and create separate columns for departure hours and minutes -

```
data.Dep_Time=data.Dep_Time.str.split(':')
```

```
data['Dep_Time_Hour']=data.Dep_Time.str[0]
data['Dep_Time_Mins']=data.Dep_Time.str[1]
```

- Further, for the arrival date and arrival time separation, we split the 'Arrival_Time' column, and create 'Arrival_date' column. We also split the time and divide it into 'Arrival_time_hours' and 'Arrival_time_minutes', similar to what we did with the 'Dep_time' column

```
data.Arrival_Time=data.Arrival_Time.str.split(' ')
```

```
data['Arrival_date']=data.Arrival_Time.str[1]  
data['Time_of_Arrival']=data.Arrival_Time.str[0]
```

```
data['Time_of_Arrival']=data.Time_of_Arrival.str.split(':')
```

```
data['Arrival_Time_Hour']=data.Time_of_Arrival.str[0]  
data['Arrival_Time_Mins']=data.Time_of_Arrival.str[1]
```

- Next, we divide the ‘Duration’ column to ‘Travel_hours’ and ‘Travel_mins’

```
#Next, we divide the 'Duration' column to 'Travel_hours' and 'Travel_mins'

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
data['Travel_Mins']=data.Duration.str[1]

data.Travel_Mins=data.Travel_Mins.str.split('m')
data.Travel_Mins=data.Travel_Mins.str[0]

#We also treat the 'Total_stops' column, and replace non-stop flights with 0 value and extract the integer part of the 'Total_St
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 also treat the 'Total_stops' column, and replace non-stop flights with 0 value and extract the integer part of the 'Total_Stops' column

```
#We also treat the 'Total_stops' column, and replace non-stop flights with 0 value and extract the integer part of the 'Total_St
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, where we observe that there are 2 categories signifying ‘No info’, which are divided into 2 categories since ‘I’ in ‘No Info’ is capital. We replace ‘No Info’ by ‘No info’ to merge it into a single category

```
data.Additional_Info.unique()
```

```
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)
```

```
data.Additional_Info.replace('No Info', 'No info', inplace=True)
```

We now drop all the columns from which we have extracted the useful information (original columns). We also drop some columns like ‘city4’,‘city5’ and ‘city6’, since majority of the data in these columns was NaN(null). As a result, we now obtain 20 different columns, which we will be feeding to our ML model. But first, we treat the missing values and explore the contents in the columns and its impact on the flight price, to separate a list of final set of columns.

```
data.isnull().sum()
Airline          0
Date_of_Journey 0
Source           0
Destination      0
Route            0
Dep_Time         0
Arrival_Time     0
Duration          0
Total_Stops      0
Additional_Info   0
Price             0
City1             0
City2             0
City3            3491
City4            9116
City5            18636
City6            10681
Date              0
Month             0
Year              0
Dep_Time_Hour    0
Dep_Time_Mins    0
Arrival_date     6348
Time_of_Arrival  0
Arrival_Time_Hour 0
Arrival_Time_Mins 0
Travel_Hours      0
Travel_Mins       1032
dtype: int64
```

```
#We also drop some columns like 'city6' and 'city5', since majority of the data in these columns was NaN(null)
data.drop(['City4','City5','City6'],axis=1,inplace=True)
```

```
data.drop(['Date_of_Journey','Route','Dep_Time','Arrival_Time','Duration'],axis=1, inplace=True)
data.drop(['Time_of_Arrival'],axis=1,inplace=True)
```

- After dropping some columns, here we can see the meaningful columns to predict the flight price without the NaN values.

```
#Checking Null Values
data.isnull().sum()
```

Airline	0
Source	0
Destination	0
Total_Stops	0
Additional_Info	0
Price	0
City1	0
City2	0
City3	3491
Date	0
Month	0
Year	0
Dep_Time_Hour	0
Dep_Time_Mins	0
Arrival_date	6348
Arrival_Time_Hour	0
Arrival_Time_Mins	0
Travel_Hours	0
Travel_Mins	1032
dtype:	int64

Activity 2.1: Replacing Missing Values

We further replace ‘NaN’ values in ‘City3’ with ‘None’, since rows where ‘City3’ is missing did not have any stop, just the source and the destination.

We also replace missing values in ‘Arrival_date’ column with values in ‘Date’ column, since the missing values are those values where the flight took off and landed on the same date.

We also replace missing values in ‘Travel_mins’ as 0, since the missing values represent that the travel time was in terms on hours only, and no additional minutes.

```
#filling City3 as None, the missing values are less  
data['City3'].fillna('None',inplace=True)
```

```
#filling Arrival_Date as Departure_Date  
data['Arrival_date'].fillna(data['Date'],inplace=True)
```

```
#filling Travel_Mins as Zero(0)  
data['Travel_Mins'].fillna(0,inplace=True)
```

- Using the above steps, we were successfully able to treat all the missing values from our data. We again check the info in our data and find out that the dataset still has data types for multiple columns as 'object', where it should be 'int'

```
data.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 10682 entries, 0 to 10682
Data columns (total 19 columns):
 #   Column            Non-Null Count  Dtype  
--- 
 0   Airline           10682 non-null   object  
 1   Source            10682 non-null   object  
 2   Destination       10682 non-null   object  
 3   Total_Stops       10682 non-null   object  
 4   Additional_Info   10682 non-null   object  
 5   Price             10682 non-null   int64  
 6   City1             10682 non-null   object  
 7   City2             10682 non-null   object  
 8   City3             10682 non-null   object  
 9   Date              10682 non-null   object  
 10  Month             10682 non-null   object  
 11  Year              10682 non-null   object  
 12  Dep_Time_Hour    10682 non-null   object  
 13  Dep_Time_Mins    10682 non-null   object  
 14  Arrival_date     10682 non-null   object  
 15  Arrival_Time_Hour 10682 non-null   object  
 16  Arrival_Time_Mins 10682 non-null   object  
 17  Travel_Hours     10682 non-null   object  
 18  Travel_Mins       10682 non-null   object  
dtypes: int64(1), object(18)
memory usage: 1.6+ MB
```

- Hence, we try to change the datatype of the required columns

```

#changing the 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_Hour=data.Dep_Time_Hour.astype('int64')
data.Dep_Time_Mins=data.Dep_Time_Mins.astype('int64')
data.Arrival_date=data.Arrival_date.astype("int64")
data.Arrival_Time_Hour=data.Arrival_Time_Hour.astype('int64')
data.Arrival_Time_Mins=data.Arrival_Time_Mins.astype('int64')
#data.Travel_Hours=data.Travel_Hours.astype('int64')
data.Travel_Mins=data.Travel_Mins.astype('int64')

```

- During this step, we face issue converting the 'Travel_hours' column, saying that the column has data as '5m', which is not allowing its conversion to 'int'.

```
data[data['Travel_Hours']=='5m']
```

Price	City1	City2	City3	Date	Month	Year	Dep_Time_Hour	Dep_Time_Mins	Arrival_date	Arrival_Time_Hour	Arrival_Time_Mins	Travel_Hours	Travel_Mins
17327	BOM	GOL	PNQ	6	3	2019	16	50	6	16	55	5m	0

- The data signifies that the flight time is '5m', which is obviously wrong as the plane cannot fly from BOMBAY->GOA->PUNE->HYDERABAD in 5 mins! (The flight has 'Total_stops' as 2)

```
data.drop(index=6474,inplace=True,axis=0)
```

- We then convert the 'Travel_hours' column to 'int' data type, and the operation happens successfully.

We now have a treated dataset with 10682 rows and 17 columns (16 independent and 1 dependent variable).

We create separate lists of categorical columns and numerical columns for plotting and analyzing the data

```
data.Travel_Hours=data.Travel_Hours.astype('int64')
```

```
#Creating list of Different types of columns
categorical=['Airline','Source','Destination','Additional_Info','City1']
numerical=['Total_Stops','Date','Month','Year','Dep_Time_Hour','Dep_Time_Mins','Arrival_date','Arrival_Time_Hour',
          'Arrival_Time_Mins','Travel_Hours','Travel_Mins']
```

Activity 2.2: Label Encoding

- Label encoding converts the data in machine readable form, but it assigns a unique number (starting from 0) to each class of data. It performs the conversion of categorical data into numeric format.
- In our dataset I have converted these variables 'Airline','Source','Destination','Total_Stops','City1','City2','City3','Additional_Info' into number format. So that it helps the model in better understanding of the dataset and enables the model to learn more complex structures.

	Airline	Source	Destination	Total_Stops	Additional_Info	Price	City1	City2	City3	Date	Month	Year	Dep_Time_Hour	Dep_Time_Mins	Arrival_date	Arr
0	3	0	5	4	7	3897	0	13	29	24	3	2019	22	20	22	
1	1	3	0	1	7	7662	2	25	1	1	5	2019	5	50	1	
2	4	2	1	1	7	13882	3	32	4	9	6	2019	9	25	10	
3	3	3	0	0	7	6218	2	34	3	12	5	2019	18	5	12	
4	3	0	5	0	7	13302	0	34	8	1	3	2019	16	50	1	

Activity 2.3: Output Columns

- Initially in our dataset we have 19 features. So, in that some features are not more important to get output (Price).
- So i removed some unrelated features and I selected important features. So, it makes easy to understand. Now we have only 12 Output Columns.

data.head()																
	Airline	Source	Destination	Total_Stops	Additional_Info	Price	City1	City2	City3	Date	Month	Year	Dep_Time_Hour	Dep_Time_Mins	Arrival_date	Arrival_Time_Hour
0	3	0	5	4	7	3897	0	13	29	24	3	2019	22	20	22	10
1	1	3	0	1	7	7662	2	25	1	1	5	2019	5	50	1	15
2	4	2	1	1	7	13882	3	32	4	9	6	2019	9	25	10	23
3	3	3	0	0	7	6218	2	34	3	12	5	2019	18	5	12	30
4	3	0	5	0	7	13302	0	34	8	1	3	2019	16	50	1	35

data = data[['Airline', 'Source', 'Destination', 'Date', 'Month', 'Year', 'Dep_Time_Hour', 'Dep_Time_Mins', 'Arrival_date', 'Arrival_Time_Hour', 'Arrival_Time_Mins', 'Price']]	◀	▶
---	---	---

data.head()														
	Airline	Source	Destination	Date	Month	Year	Dep_Time_Hour	Dep_Time_Mins	Arrival_date	Arrival_Time_Hour	Arrival_Time_Mins	Price		
0	3	0	5	24	3	2019	22	20	22	1	10	3897		
1	1	3	0	1	5	2019	5	50	1	13	15	7662		
2	4	2	1	9	6	2019	9	25	10	4	25	13882		
3	3	3	0	12	5	2019	18	5	12	23	30	6218		
4	3	0	5	1	3	2019	16	50	1	21	35	13302		

Milestone 3: Exploratory Data Analysis

Activity 1: Descriptive statistical

Descriptive analysis is to study the basic features of data with the statistical process. Here pandas has a worthy function called describe. With this describe function we can understand the unique, top and frequent values of categorical features. And we can find mean, std, min, max and percentile values of continuous features.

data.describe()	
Price	
count	10683.000000
mean	9087.064121
std	4611.359167
min	1759.000000
25%	5277.000000
50%	8372.000000
75%	12373.000000
max	79512.000000

Activity 2: Visual Analysis

- Plotting countplots for categorical data

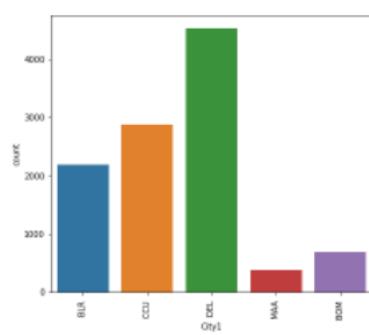
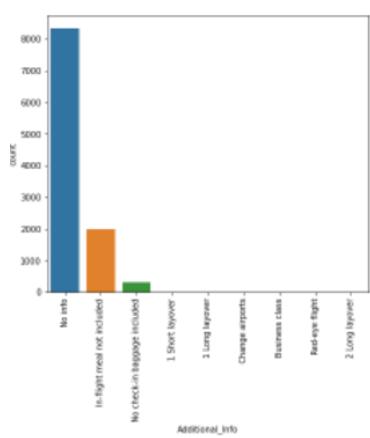
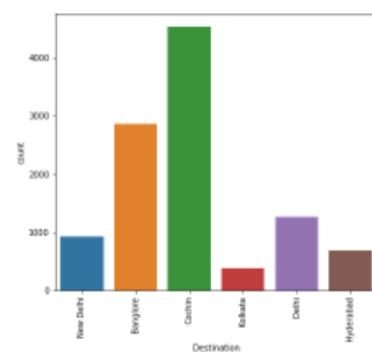
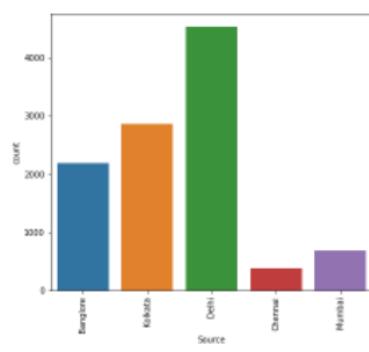
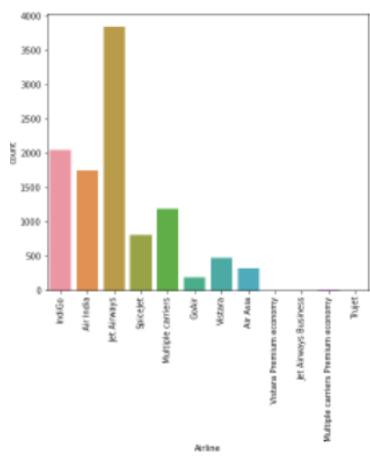
```
#plotting Countplots for Categorical Data

import seaborn as sns
c=1
plt.figure(figsize=(20,45))

for i in categorical:
    plt.subplot(6,3,c)
    sns.countplot(data[i])
    plt.xticks(rotation=90)
    plt.tight_layout(pad=3.0)
    c=c+1

plt.show()

C:\Users\SmartBridge-PC\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a
keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an ex
plicit keyword will result in an error or misinterpretation.
    warnings.warn(
C:\Users\SmartBridge-PC\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a
keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an ex
plicit keyword will result in an error or misinterpretation.
    warnings.warn(
C:\Users\SmartBridge-PC\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a
keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an ex
plicit keyword will result in an error or misinterpretation.
    warnings.warn(
C:\Users\SmartBridge-PC\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a
keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an ex
plicit keyword will result in an error or misinterpretation.
    warnings.warn(
C:\Users\SmartBridge-PC\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a
keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an ex
plicit keyword will result in an error or misinterpretation.
    warnings.warn(
```



Activate Windows
Go to Settings to activate Wind

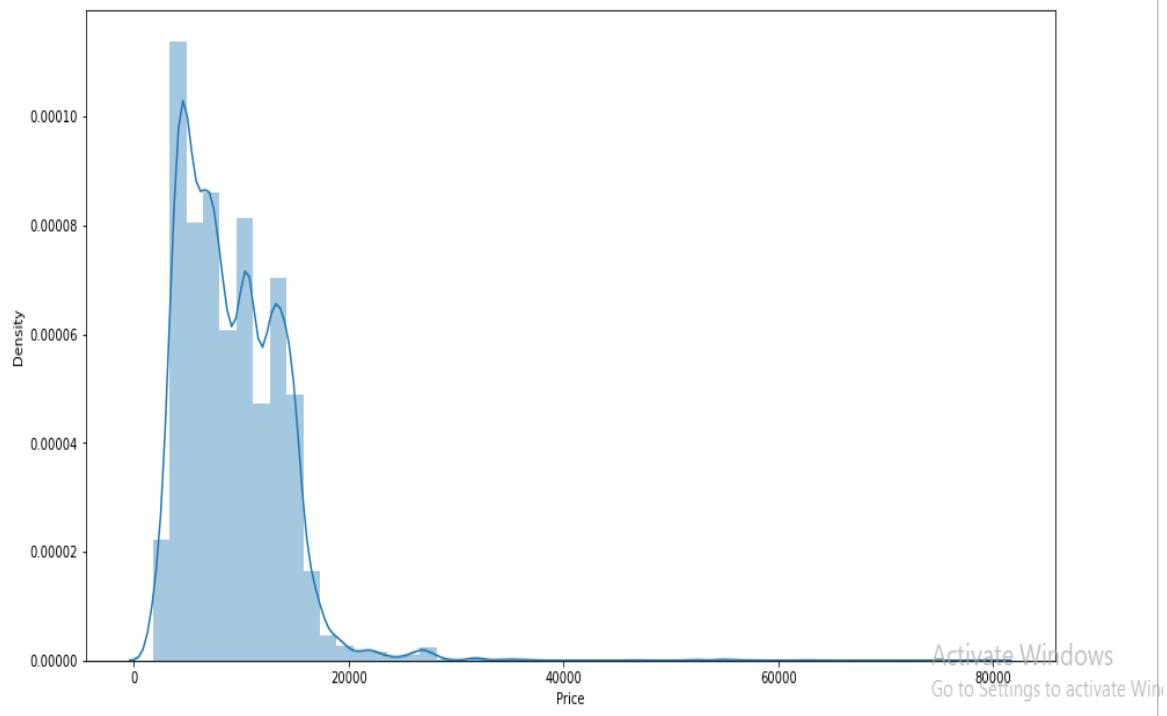
Activity 2.1: We now plot distribution plots to check the distribution in numerical data (Distribution of 'Price' Column)

- The seaborn.displot() function is used to plot the displot. The displot represents the univariate distribution of data variable as an argument and returns the plot with the density distribution. Here, I used distribution(displot) on 'Price' column.
- It estimates the probability of distribution of continuous variable across various data.


```
#Distribution of 'PRICE' Column  
plt.figure(figsize=(15,8))  
sns.distplot(data.Price)
```

```
C:\Users\SmartBridge-PC\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).  
warnings.warn(msg, FutureWarning)
```

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

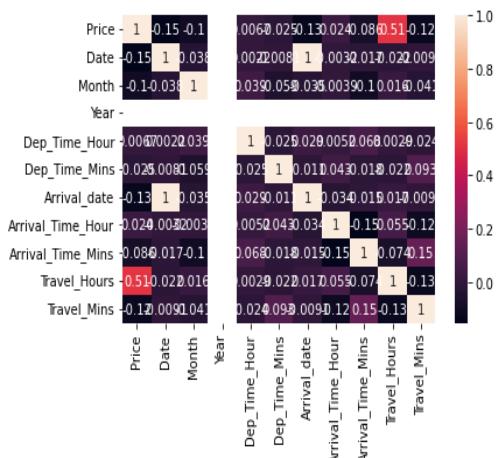


Activity 2.2: Checking the Correlation Using HeatMap

- Here, I'm finding the correlation using HeatMap. It visualizes the data in 2-D colored maps making use of color variations. It describes the relationship variables in form of colors instead of numbers it will be plotted on both axes.
- So, by this heatmap we found that correlation between 'Arrival_date' and 'Date'. Remaining all columns don't have the any Correlation.

```
sns.heatmap(data.corr(), annot=True)
```

```
<AxesSubplot:>
```



Activity 2.3: Outlier Detection for 'Price' Column

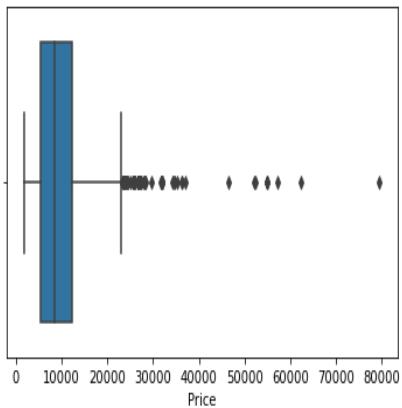
- Sometimes it's best to keep outliers in your data. It captures the valuable information and they can effect on statistical results and detect any errors in your statistical process. Here, we are checking Outliers in the 'Price' column.

```
# Detecting the Outliers
import seaborn as sns
sns.boxplot(data['Price'])
```

C:\Users\SmartBridge-PC\anaconda3\lib\site-packages\seaborn_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be 'data', and passing other arguments without an explicit keyword will result in an error or misinterpretation.

```
warnings.warn(
```

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



Scaling the Data

- We are taking two variables 'x' and 'y' to split the dataset as train and test.
- On x variable, drop is passed with dropping the target variable. And on y target variable('Price') is passed.
- Scaling the features makes the flow of gradient descent smooth and helps algorithms quickly reach the minima of the cost function.
- Without scaling features, the algorithm maybe biased toward the feature which has values higher in magnitude. it brings every feature in the same range and the model uses every feature wisely.

- We have popular techniques used to scale all the features but I used StandardScaler in which we transform the feature such that the changed features will have mean=0 and standard deviation=1.

```

y = data['Price']
x = data.drop(columns=['Price'],axis=1)

### Scaling the Data

from sklearn.preprocessing import StandardScaler
ss=StandardScaler()

x_scaled = ss.fit_transform(x)

x_scaled = pd.DataFrame(x_scaled,columns=x.columns)
x_scaled.head()

```

	Airline	Source	Destination	Date	Month	Year	Dep_Time_Hour	Dep_Time_Mins	Arrival_date	Arrival_Time_Hour	Arrival_Time_Mins
0	-0.410934	-1.658354	2.416648	1.237192	-1.467619	0.0	1.654162	-0.234832	0.955658	-1.800328	-0.889941
1	-1.261305	0.890262	-0.973718	-1.475375	0.250165	0.0	-1.303018	1.363790	-1.524701	-0.050871	-0.586988
2	0.014251	0.040723	-0.295645	-0.531874	1.109057	0.0	-0.607211	0.031605	-0.461690	-1.362964	0.018919
3	-0.410934	0.890262	-0.973718	-0.178060	0.250165	0.0	0.958355	-1.034142	-0.225465	1.407010	0.321872
4	-0.410934	-1.658354	2.416648	-1.475375	-1.467619	0.0	0.610452	1.363790	-1.524701	1.115434	0.624825

Splitting data into train and test

Now let's split the Dataset into train and test sets.

For splitting training and testing data we are using `train_test_split()` function. from `sklearn`. As parameters, we are passing `x`, `y`, `test_size`, `random_state`.

```
from sklearn.model_selection import train_test_split  
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)
```

```
x_train.head()
```

	Airline	Source	Destination	Date	Month	Year	Dep_Time_Hour	Dep_Time_Mins	Arrival_date	Arrival_Time_Hour	Arrival_Time_Mins
10611	4	4	3	18	5	2019	7	5	18	8	30
1034	8	2	1	24	4	2019	15	45	24	22	5
8123	4	2	1	27	6	2019	2	15	27	12	35
4779	4	3	0	1	4	2019	6	30	1	18	15
3207	3	3	0	24	5	2019	18	5	24	23	30

Milestone 4: Model Building

Now our data is cleaned and it's time to build the model. We can train our data on different algorithms. for this project we are applying four regression algorithms. The best model is saved based on its performance.

Activity 1: Using Ensemble Techniques

RandomForestRegressor, GradientBoostingRegressor, AdaBoostRegressor

A function named RandomForest, GradientBoosting, AdaBoost is created and train and test data are passed as the parameters. Inside the function, RandomForest, GradientBoosting, AdaBoost algorithm is initialized and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in new variable. For evaluating the model, r2_score, mean_absolute_error, and mean_squared_error report is done.


```
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor, AdaBoostRegressor
rfr=RandomForestRegressor()
gb=GradientBoostingRegressor()
ad=AdaBoostRegressor()
```

```
from sklearn.metrics import r2_score,mean_absolute_error,mean_squared_error

for i in [rfr,gb,ad]:
    i.fit(x_train,y_train)
    y_pred=i.predict(x_test)
    test_score=r2_score(y_test,y_pred)
    train_score=r2_score(y_train, i.predict(x_train))
    if abs(train_score-test_score)<=0.2:
        print(i)

    print("R2 score is",r2_score(y_test,y_pred))
    print("R2 for train data",r2_score(y_train, i.predict(x_train)))
    print("Mean Absolute Error is",mean_absolute_error(y_pred,y_test))
    print("Mean Squared Error is",mean_squared_error(y_pred,y_test))
    print("Root Mean Squared Error is", (mean_squared_error(y_pred,y_test,squared=False)))
```

```
RandomForestRegressor()
R2 score is 0.8227214297234019
R2 for train data 0.9510465962960551
Mean Absolute Error is 1182.0594710483324
Mean Squared Error is 3742662.8044006103
Root Mean Squared Error is 1934.596289772264
GradientBoostingRegressor()
R2 score is 0.7647464119441486
R2 for train data 0.7333243455087605
Mean Absolute Error is 1678.510006493234
Mean Squared Error is 4966617.523170804
Root Mean Squared Error is 2228.5909277323203
AdaBoostRegressor()
R2 score is 0.2582227532056507
R2 for train data 0.2911833713550127
Mean Absolute Error is 3276.5456982057563
Mean Squared Error is 15660223.942444455
Root Mean Squared Error is 3957.3000824355554
```

Activate Windows
Go to Settings to activate Win

Activity 2: Regression Model

KNeighborsRegressor, SVR, DecisionTreeRegressor

A function named KNN, SVR, DecisionTree is created and train and test data are passed as the parameters. Inside the function, KNN, SVR, DecisionTree algorithm is initialized and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in new variable. For evaluating the model, r2_score, mean_absolute_error, and mean_squared_error is done.


```

from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor

from sklearn.metrics import r2_score,mean_absolute_error,mean_squared_error

knn=KNeighborsRegressor()
svr=SVR()
dt=DecisionTreeRegressor()

for i in [knn,svr,dt]:
    i.fit(x_train,y_train)
    y_pred=i.predict(x_test)
    test_score=r2_score(y_test,y_pred)
    train_score=r2_score(y_train,i.predict(x_train))
    if abs(train_score-test_score)<=0.1:
        print(i)
        print('R2 Score is',r2_score(y_test,y_pred))
        print('R2 Score for train data',r2_score(y_train,i.predict(x_train)))
        print('Mean Absolute Error is',mean_absolute_error(y_test,y_pred))
        print('Mean Squared Error is',mean_squared_error(y_test,y_pred))
        print('Root Mean Squared Error is',(mean_squared_error(y_test,y_pred,squared=False)))

```

```

KNeighborsRegressor()
R2 Score is 0.7354576039734038
R2 Score for train data 0.7910150823510993
Mean Absolute Error is 1635.3106223678053
Mean Squared Error is 5584955.836743098
Root Mean Squared Error is 2363.2511158874117
SVR()
R2 Score is -0.007934481035057894
R2 Score for train data -0.012381130959185693
Mean Absolute Error is 3631.923243955232
Mean Squared Error is 21279271.857602067
Root Mean Squared Error is 4612.94611475162

```

Activate Windows

Activity 3: Checking Cross Validation for RandomForestRegressor

We perform the cross validation of our model to check if the model has any overfitting issue, by checking the ability of the model to make predictions on new data, using k-folds. We test the cross validation for Random forest and Gradient Boosting Regressor.

```
from sklearn.model_selection import cross_val_score
for i in range(2,5):
    cv=cross_val_score(rfr,x,y,cv=i)
    print(rfr.cv.mean())
```

```
RandomForestRegressor() 0.7916634416866438
RandomForestRegressor() 0.7929369032321089
RandomForestRegressor() 0.799914397784633
```

Activity 4: Hypertuning the model

RandomSearch CV is a technique used to validate the model with different parameter combinations, by creating a random of parameters and trying all the combinations to compare which combination gave the best results. We apply random search on our model.

From sklearn, cross_val_score is used to evaluate the score of the model. On the parameters, we have given rf (model name), x, y, cv (as 3 folds). Our model is performing well.


```

from sklearn.model_selection import RandomizedSearchCV

param_grid={'n_estimators':[10,30,50,70,100], 'max_depth':[None,1,2,3],
            'max_features':['auto','sqrt']}
rfr=RandomForestRegressor()
rf_res=RandomizedSearchCV(estimator=rfr,param_distributions=param_grid,cv=3,verbose=2,n_jobs=-1)

rf_res.fit(x_train,y_train)

```

Fitting 3 folds for each of 10 candidates, totalling 30 fits

```

RandomizedSearchCV(cv=3, estimator=RandomForestRegressor(), n_jobs=-1,
                    param_distributions={'max_depth': [None, 1, 2, 3],
                                         'max_features': ['auto', 'sqrt'],
                                         'n_estimators': [10, 30, 50, 70, 100]},
                    verbose=2)

```

```

gb=GradientBoostingRegressor()
gb_res=RandomizedSearchCV(estimator=gb,param_distributions=param_grid,cv=3,verbose=2,n_jobs=-1)

gb_res.fit(x_train,y_train)

```

Fitting 3 folds for each of 10 candidates, totalling 30 fits

```

RandomizedSearchCV(cv=3, estimator=GradientBoostingRegressor(), n_jobs=-1,
                    param_distributions={'max_depth': [None, 1, 2, 3],
                                         'max_features': ['auto', 'sqrt'],
                                         'n_estimators': [10, 30, 50, 70, 100]},
                    verbose=2)

```

Now let's see the performance of all the models and save the best model

Activity 5: Accuracy

Checking Train and Test Accuracy by RandomSearchCV using RandomForestRegression Model

```
rfr=RandomForestRegressor(n_estimators=10,max_features='sqrt',max_depth=None)
rfr.fit(x_train,y_train)
y_train_pred=rfr.predict(x_train)
y_test_pred=rfr.predict(x_test)
print("train accuracy",r2_score(y_train_pred,y_train))
print("test accuracy",r2_score(y_test_pred,y_test))
```

```
train accuracy 0.9299395776145483
test accuracy 0.7657841369272524
```

Checking Train and Test Accuracy by RandomSearchCV using KNN Model2

```
knn=KNeighborsRegressor(n_neighbors=2,algorithm='auto',metric_params=None,n_jobs=-1)
knn.fit(x_train,y_train)
y_train_pred=knn.predict(x_train)
y_test_pred=knn.predict(x_test)
print("train accuracy",r2_score(y_train_pred,y_train))
print("test accuracy",r2_score(y_test_pred,y_test))
```

```
train accuracy 0.8829162343701471
test accuracy 0.6874228308668873
```

By Observing two models train and test accuracy we are getting good accuracy in RandomForestRegression

Activity 6: Evaluating performance of the model and saving the model

From sklearn, cross_val_score is used to evaluate the score of the model. On the parameters, we have given rfr (model name), x, y, cv (as 3 folds). Our model is performing well. So, we are saving the model by pickle.dump().

Note: To understand cross validation, refer this link.

<https://towardsdatascience.com/cross-validation-explained-evaluating-estimator-performance-e51e5430ff85>.

```
rfr=RandomForestRegressor(n_estimators=10,max_features='sqrt',max_depth=None)
rfr.fit(x_train,y_train)
y_train_pred=rfr.predict(x_train)
y_test_pred=rfr.predict(x_test)
print("train accuracy",r2_score(y_train_pred,y_train))
print("test accuracy",r2_score(y_test_pred,y_test))
```

```
train accuracy 0.9299395776145483
test accuracy 0.7657841369272524
```

```
price_list=pd.DataFrame({'Price':prices})
```

```
price_list
```

	Price
0	5852.800000
1	9121.900000
2	10931.640000
3	14780.700000
4	6064.600000
..	..
2132	7171.200000
2133	7381.200000
2134	7820.900000
2135	12388.673333
2136	13314.400000

2137 rows × 1 columns

```
import pickle  
pickle.dump(rfr,open('model1.pkl','wb'))
```

Activate Windows

Go to Settings to activate Win

Milestone 6: Model Deployment

Activity 1: Save the best model

Saving the best model after comparing its performance using different evaluation metrics means selecting the model with the highest performance and saving its weights and configuration. This can be useful in avoiding the need to retrain the model every time it is needed and also to be able to use it in the future.

```
import pickle  
pickle.dump(rfr,open('model1.pkl','wb'))
```

Activity 2: Integrate with Web Framework

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the user where he has to enter the values for predictions. The entered values are given to the saved model and prediction is showcased on the UI.

This section has the following tasks

- Building HTML Pages
- Building server side script
- Run the web application

Activity 2.1: Building Html Pages:

For this project create two HTML files namely

- home.html

- predict.html
- submit.html

and save them in the templates folder.

Activity 2.2: Build Python code:

Import the libraries

```
y ×  
from flask import Flask, render_template, request  
import numpy as np  
import pickle
```

Load the saved model. Importing the flask module in the project is mandatory. An object of Flask class is our WSGI application. Flask constructor takes the name of the current module (`__name__`) as argument.

```
model = pickle.load(open(r"model1.pkl", 'rb'))
```

Render HTML page:

```
@app.route("/home")  
def home():  
    return render_template('home.html')
```

Here we will be using a declared constructor to route to the HTML page which we have created earlier.

In the above example, '/' URL is bound with the `home.html` function. Hence, when the home page of the web server is opened in the browser, the html page will be rendered. Whenever you enter the values from the html page the values can be retrieved using POST Method.

Retrieves the value from UI:

```

@app.route("/predict")
def home1():
    return render_template('predict.html')

@app.route("/pred", methods=[ 'POST', 'GET'])
def predict():
    x = [[int(x) for x in request.form.values()]]
    print(x)

    x = np.array(x)
    print(x.shape)

    print(x)
    pred = model.predict(x)
    print(pred)
    return render_template('submit.html', prediction_text=pred)

```

Here we are routing our app to predict() function. This function retrieves all the values from the HTML page using Post request. That is stored in an array. This array is passed to the model.predict() function. This function returns the prediction. And this prediction value will be rendered to the text that we have mentioned in the submit.html earlier.

Main Function:

```
if __name__ == "__main__":
    app.run(debug=False)
```

Activity 2.3: Run the web application

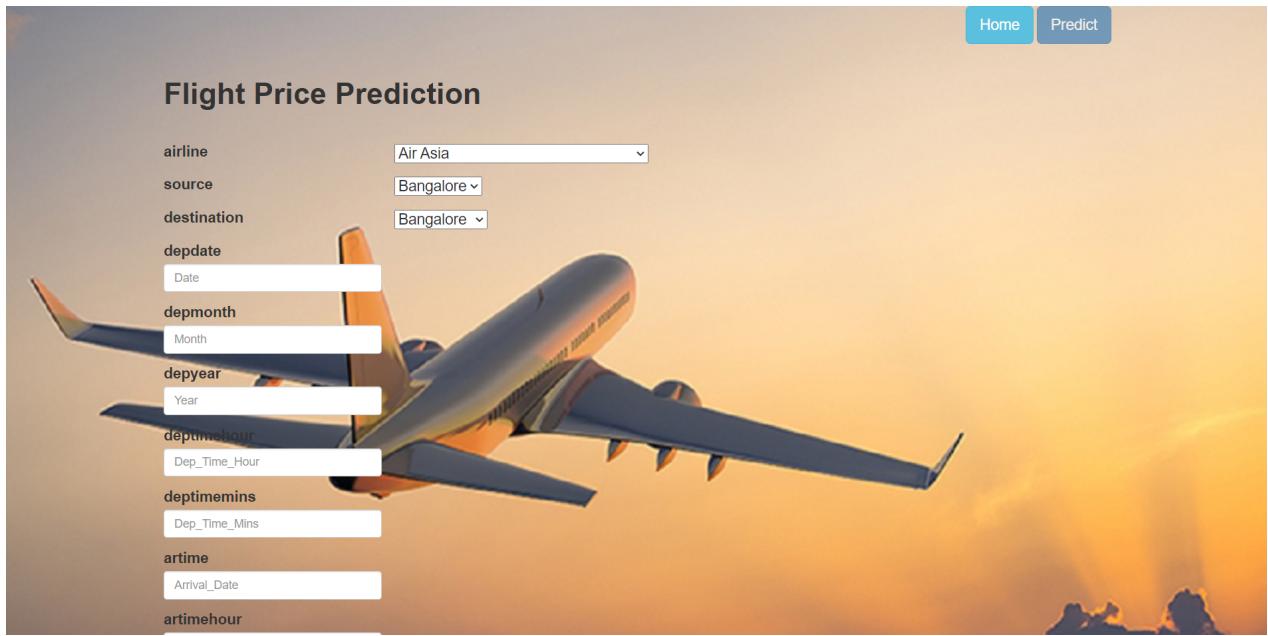
- Open anaconda prompt from the start menu
- Navigate to the folder where your python script is.
- Now type “python app.py” command
- Navigate to the localhost where you can view your web page.
- Click on the predict button from the top left corner, enter the inputs, click on the submit button, and see the result/prediction on the web.

```
* Serving Flask app "app" (lazy loading)
* Environment: production
WARNING: This is a development server. Do not use it in a p
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Now, Go the web browser and write the localhost url (<http://127.0.0.1:5000>) to get the below result



Now, when you click on Predict button you will get redirected to the prediction page.



Flight Price Prediction

airline: Air Asia

source: Bangalore

destination: Bangalore

depdate: Date

depmonth: Month

depyear: Year

deptimehour: Dep_Time_Hour

deptimemins: Dep_Time_Mins

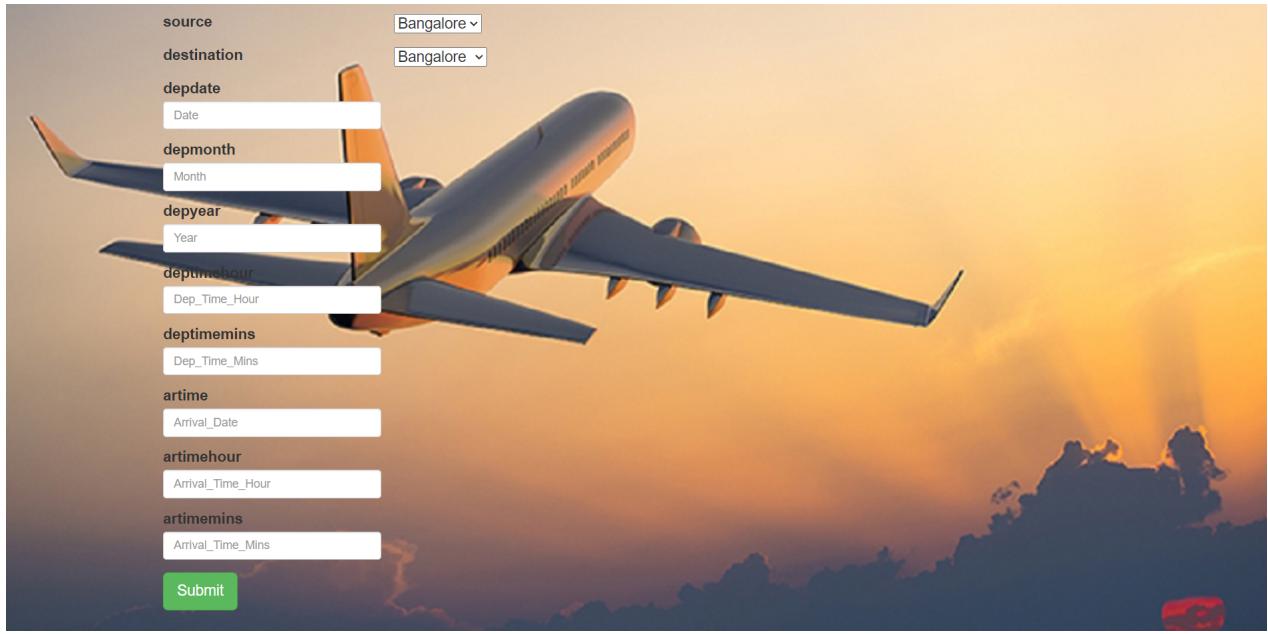
artime: Arrival_Date

artimehour: Arrival_Time_Hour

artimemins: Arrival_Time_Mins

Home **Predict**

This screenshot shows the first step of the flight price prediction process. The user has selected "Air Asia" as the airline and "Bangalore" as both the source and destination. All other input fields are currently empty or set to their default values. The background features a sunset sky with an airplane silhouette.



source: Bangalore

destination: Bangalore

depdate: Date

depmonth: Month

depyear: Year

deptimehour: Dep_Time_Hour

deptimemins: Dep_Time_Mins

artime: Arrival_Date

artimehour: Arrival_Time_Hour

artimemins: Arrival_Time_Mins

Submit

This screenshot shows the second step of the flight price prediction process. The user has selected "Bangalore" as both the source and destination. All other input fields are currently empty or set to their default values. A green "Submit" button is visible at the bottom left. The background features a sunset sky with an airplane silhouette.

Input 1- Now, the user will give inputs to get the predicted result after clicking onto the submit button.

[Home](#) | [Predict](#)

Flight Price Prediction

airline

source

destination

deptime

depmonth

depyear

deptimehour

deptimemins

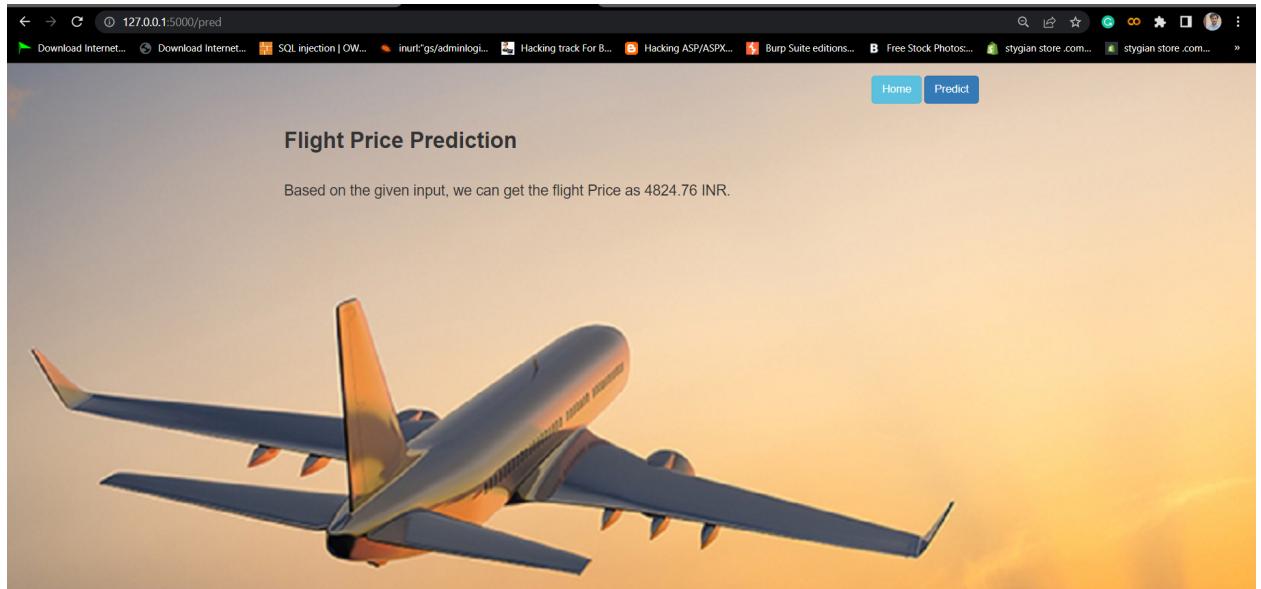
aritime

aritimehour

aritimemins

Activate Windows
Go to Settings to activate Windows

Now when you click on submit button from right top corner you will get redirected to submit.html



Milestone 7: Project Demonstration & Documentation

Below mentioned deliverables to be submitted along with other deliverables

Activity 1:- Record explanation Video for project end to end solution

Activity 2:- Project Documentation-Step by step project development procedure

Create document as per the template provided