

**THIRU.VI.KA GOVT ARTS COLLEGE**

**THIRUVARUR-610 001.**



## **DEPARTMENT OF COMPUTER SCIENCE**

### **MACHINE LEARNING WITH PYTHON**

**Project Title:** Optimizing Flight Booking Decisions through Machine Learning Price Predictions

**Team Leader:** NIRMALA R

**Team member:** PRASANNA A

**Team member:** SABITHA M

**Team member:** PRAVEENKUMAR S

## 1. INTRODUCTION

### 1.1 Overview:

Perfect time for purchasing plane ticket by the passenger's view is difficult since passengers get very less information of future business price rates. Different models figure out future business price on plane and categorise the best time to obtain flight ticket. Airlines use different strategies of pricing for their tickets, later taking the decision on price because order shows higher value for the approximation models. The causes behind the difficult system is each Planes has limited number of seats to be filled, so airlines must regulate demand. Suppose when demand is expected to increase capacity, the airline may increase prices, to decrease the rate at which seats fill.

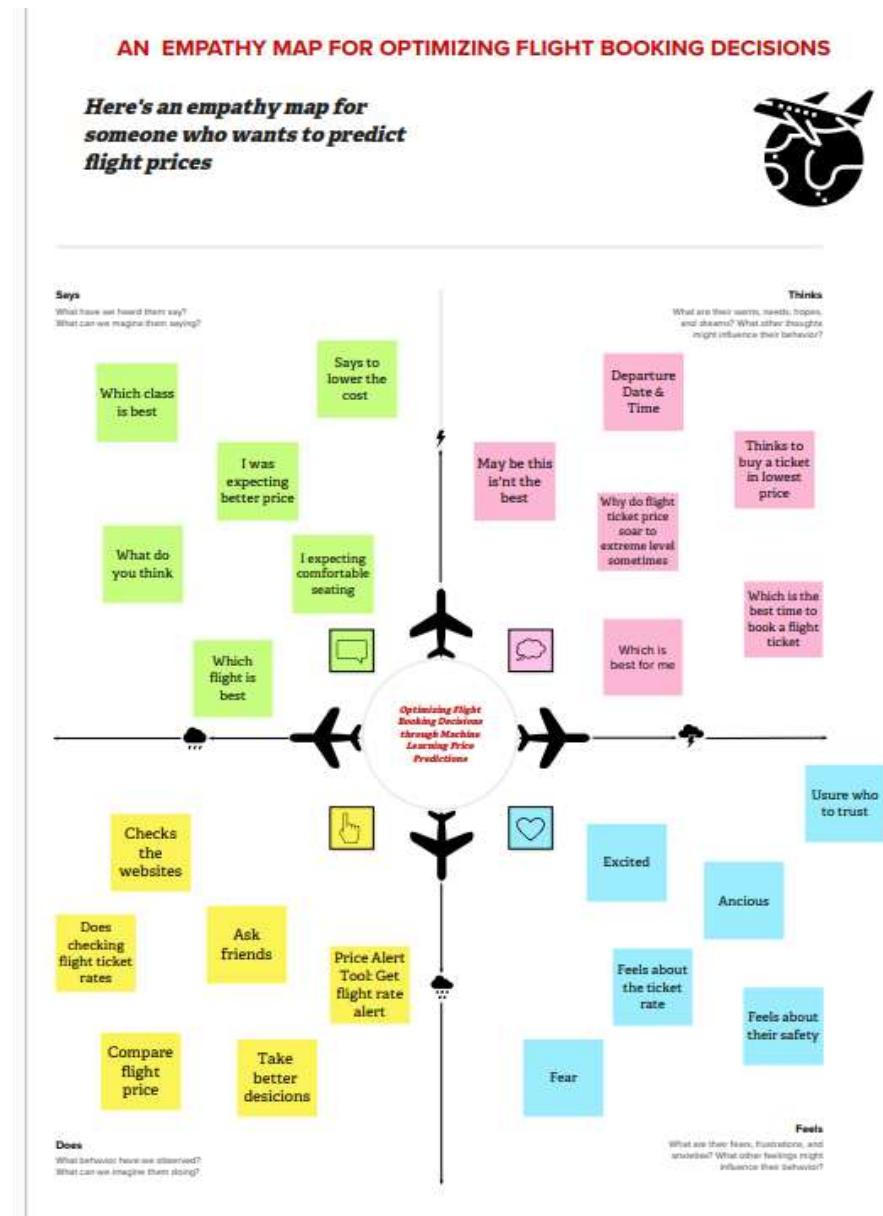
Also, seating arrangements in flight which is not occupied shows the loss of the amount invested for the business airline companies and making them purchase the ticket to fill the seats for any price this would be the best idea to get profit in loss too. Passengers should be compatible with the airline companies to get adjusted for the increase and decrease of the price. Passengers or customers should make their own planning to get the best offers available on different airlines and travel through less price. Planes ticket prices changes as time passes, pulling out the elements which creates the difference. Reporting the correlated and models which is used to price the flight tickets. Then, using that information, building the model which helps passengers to make pull out the ticket to buy and predicting air ticket prices which progresses in the future. Duration, Arrival time, Price, Source, Destination and much more these are the attribute used for flight price prediction.

## 1.2 Purpose

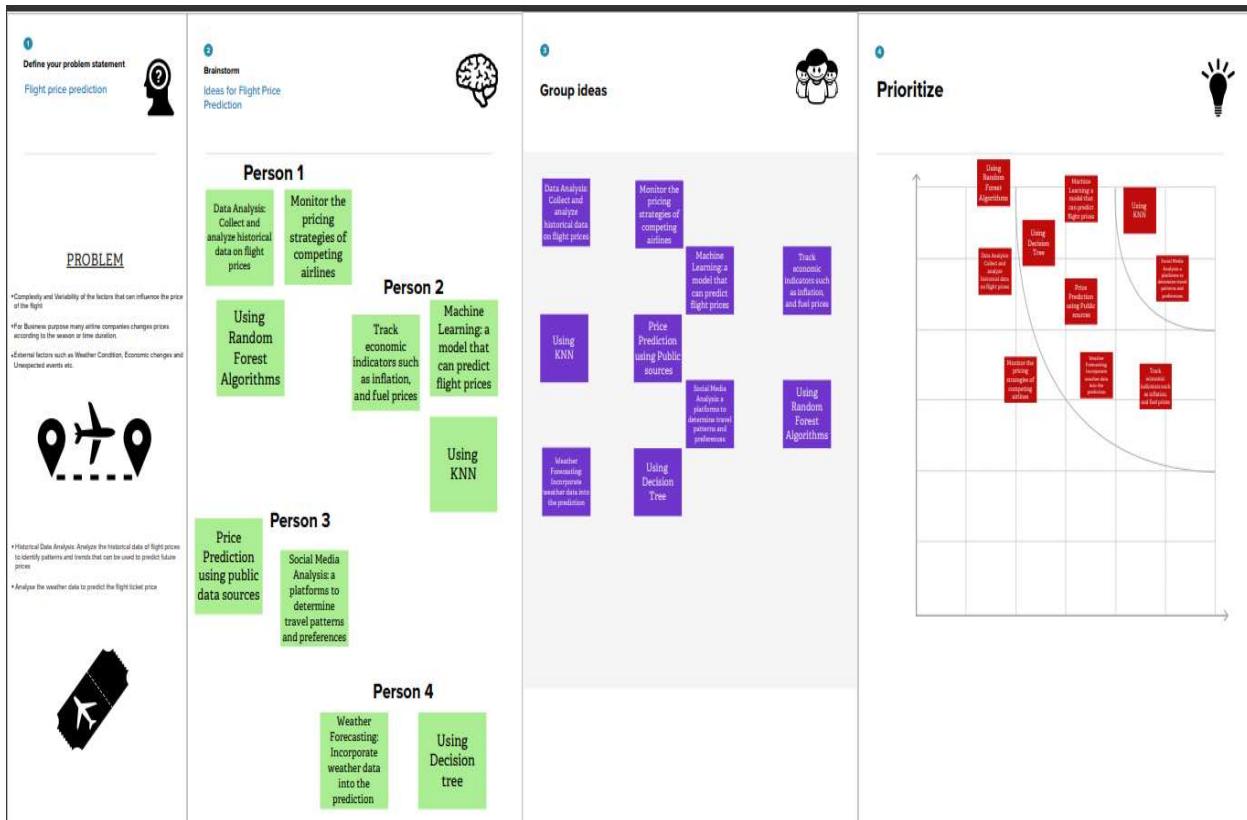
- ❖ The primary purpose of flight price prediction is to help travelers save money by predicting when prices are likely to go up or down. By using this information, travelers can time their purchases to get the best possible deals on flights.
- ❖ A Flight price prediction application which predicts fares of flight for a particular date based on various parameters like source, Destination, Stops and Airlines. The prediction will help a traveller to decide a specific airline as per his/her budget.
- ❖ The main purpose of our system is to predict the flight prices with comparison of today to another any day because this customer can book their tickets of Flight according to their comfortability, according to their affordability, Means whichever cheaper cost they want they can easily choose. The purpose is to provide customers with the relevant information they need to decide the best time to purchase a ticket.
- ❖ By helping travelers save money and plan their trips more effectively, flight price prediction tools can improve overall travel experiences. This reduces stress and enhances enjoyment, making travel more accessible and enjoyable for everyone.

## 2. PROBLEM DEFINITION AND DESIGN THINKING

### 2.1 Empathy map

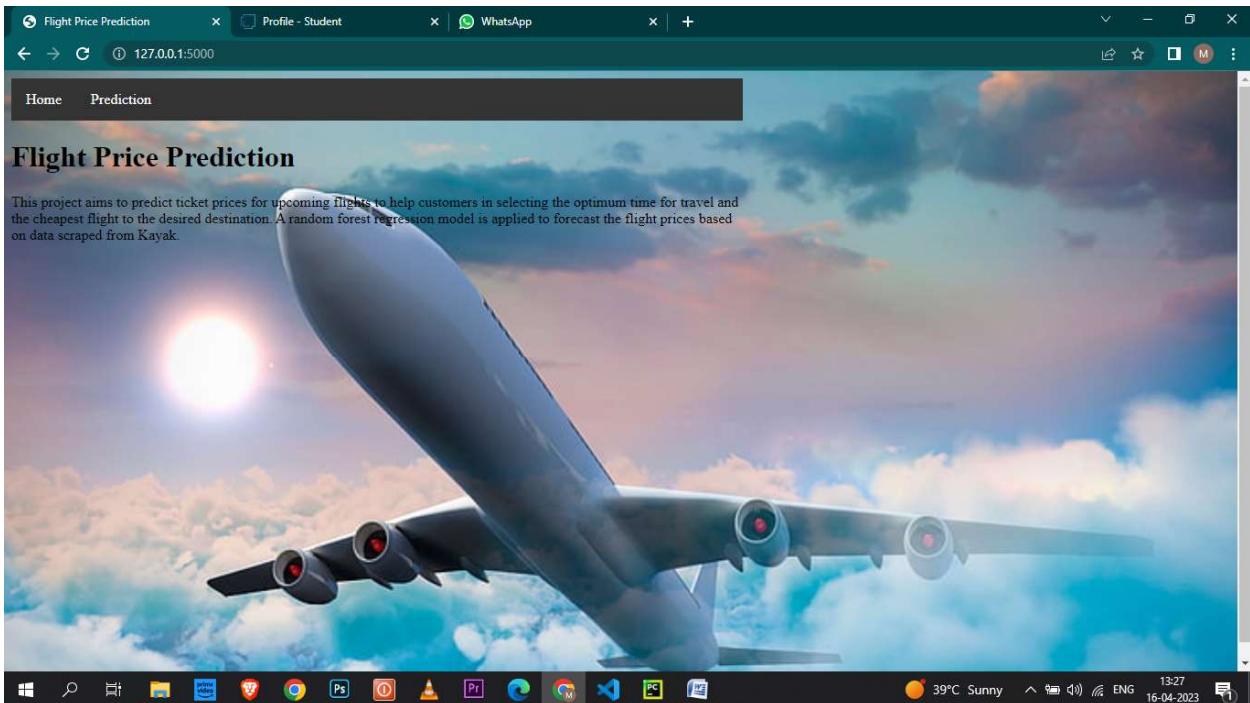


## 2.2 Ideation & Brainstorming Map

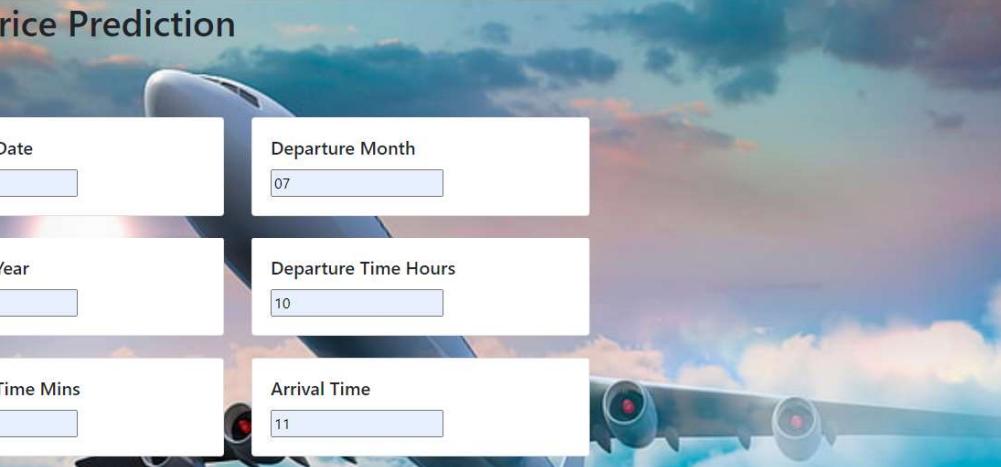


### 3. RESULT

Home Page:



## Predict page:



Flight Price Prediction

Home Prediction

# Flight Price Prediction

Departure Date: 12

Departure Month: 07

Departure Year: 2023

Departure Time Hours: 10

Departure Time Mins: 60

Arrival Time: 11

Arrival Time Hours: 1

Arrival Time Mins: 60

Flight Price Prediction

127.0.0.1:5000/predict

Departure Time Mins: 60

Arrival Time: 11

Arrival Time Hours: 1

Arrival Time Mins: 60

Source: Chennai

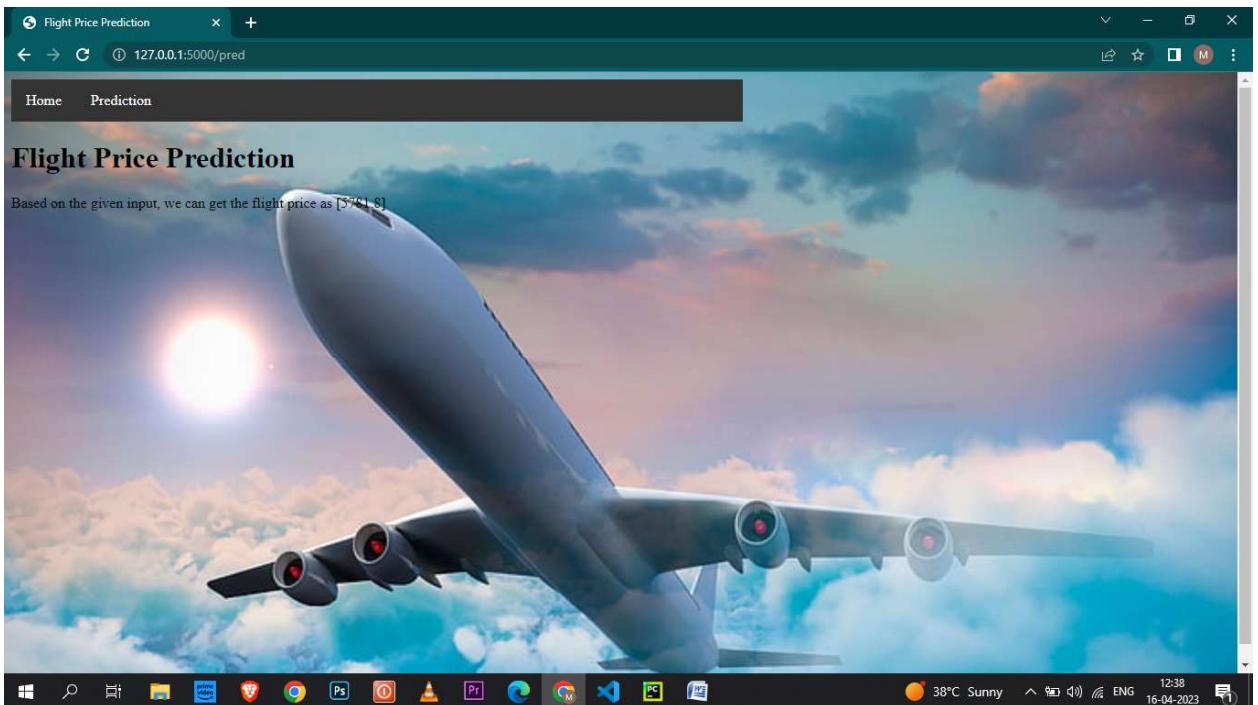
Destination: New Delhi

Which Airline you want to travel? Jet Airways

Submit

38°C Sunny 12:38 16-04-2023

## Submit Page (Final Output):



#### 4. ADVANTAGES AND DISADVANTAGES

##### Advantages:

- Traveler get the fare prediction handy using which it's easy to decide the airlines.
- Saves time in searching/deciding for airlines.
- Flight price prediction tools can help you save money on your flights by alerting you to when the prices are expected to drop, allowing you to book your flights as the optimal time.
- Flight price prediction tools allow you to track multiple flights and airlines at once, so you can easily compare price and choose the best option for you.
- Rather than spending hours searching for the best deals on flights, you can use a flight price prediction tool to do the work for you, giving you more time to focus on planning the rest of your trip.
- If you have flexible travel dates, flight price prediction tools can help you find the cheapest times to travel, allowing you to adjust your plans accordingly.
- Knowing that you got the best deal possible on your flights can provide a sense of satisfaction and peace of mind, making your travel experience more enjoyable overall.

## Disadvantages:

- Improper data will result in incorrect fare predictions.
- Flight price prediction tools use historical data and algorithms to predict future prices, but there are many factors that can influence the accuracy of these predictions, including unexpected events such as natural disasters or political unrest.
- Not all airlines or flights are covered by flight price prediction tools, so travelers may not be able to get accurate predictions for every flight they are interested in.
- Flight price prediction tools can provide travelers with information about when to book their flights, but they cannot control the actual prices or availability of flights. This means that even if a prediction suggests that prices will drop in the future, there is no guarantee that this will happen.
- Some travelers may become too reliant on flight price prediction tools and may miss out on good deals if they rely solely on these tools rather than actively searching for the best deals.
- Some flight price prediction tools may collect personal data from travelers, raising concerns about privacy and data protection.

## 5. APPLICATIONS

Flight price prediction has several applications for both individual travelers and travel companies:

- **Personal travel planning:** Individuals can use flight price prediction tools to find the best deals on flights for personal travel. By monitoring prices and predicting future changes, travelers can save money and plan their trips more effectively.
- **Business travel planning:** Companies can use flight price prediction tools to find the best deals on flights for business travel. This can help reduce travel costs and increase overall efficiency.
- **Travel agency services:** Travel agencies can use flight price predictions tools to offer their clients the best possible deals on flights. This can help increase customer satisfaction and loyalty.
- **Airline revenue management:** Airlines can use flight price prediction tools to adjust prices in real-time based on predicted demand. This can help airlines optimize revenue and increase profitability.
- **Tourism industry:** Flight price prediction tools can help the tourism industry plan and promote travel packages and events more effectively. By understanding future price changes, travel companies can tailor their offerings to meet the needs of their customers.

## 6. CONCLUSION

Evaluating the algorithmic rule, a dataset is collected, pre-processed, performed data modelling and studied a value difference for the number of restricted days by the passengers for travelling. Machine Learning algorithms with square measure for forecasting the accurate fare of airlines and it gives accurate value of plane price ticket at limited and highest value. Information is collected from Kaggle websites that sell the flight tickets therefore restricting data which are often accessed. The results obtained by the random forest and decision tree algorithm has better accuracy, but best accuracy is predicted by random Forest algorithm to fitting/split the train & test data, (x\_train, y\_train)accuracy= 93%, (x\_test, y\_test)accuracy = 77%.

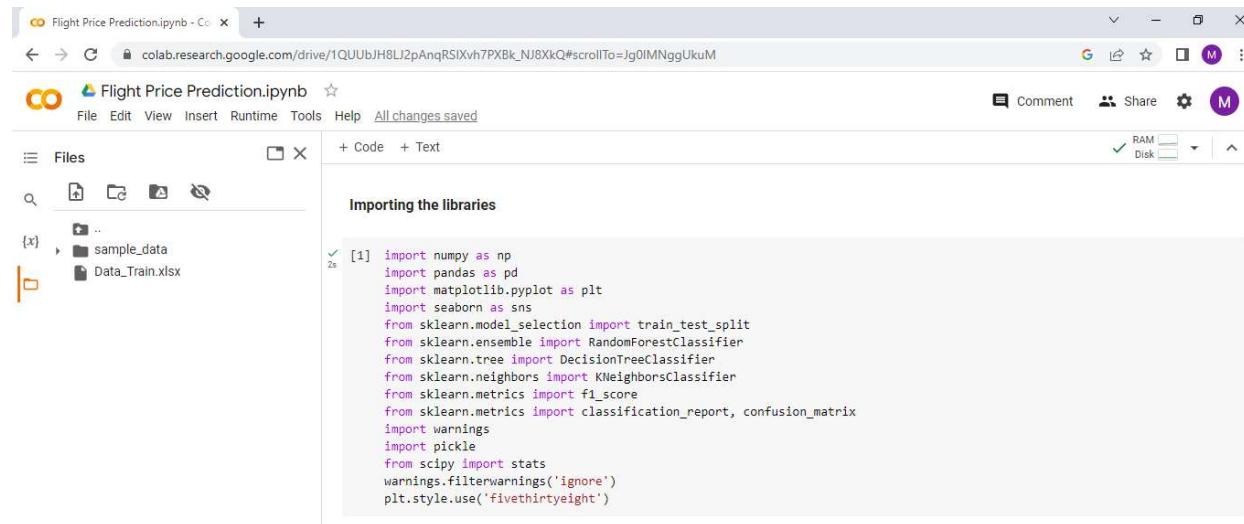
## 7. FUTURE SCOPE

In Upcoming days when huge amount of information is accessed as in detailed information in the dataset, the expected results in future are highly correct. For further research anyone desire to expand upon it ought to request different sources of historical data or be a lot of organized in collection knowledge manually over amount of your time to boot, a lot of different combination of plane are going to be traversed. There is whole possibility that planes differ their execution ideas consisting characteristics of the plane. At last, it is curious to match our model accuracy with that of the business models accuracy offered nowadays.

## 8. APPENDIX

### A. Source Code :

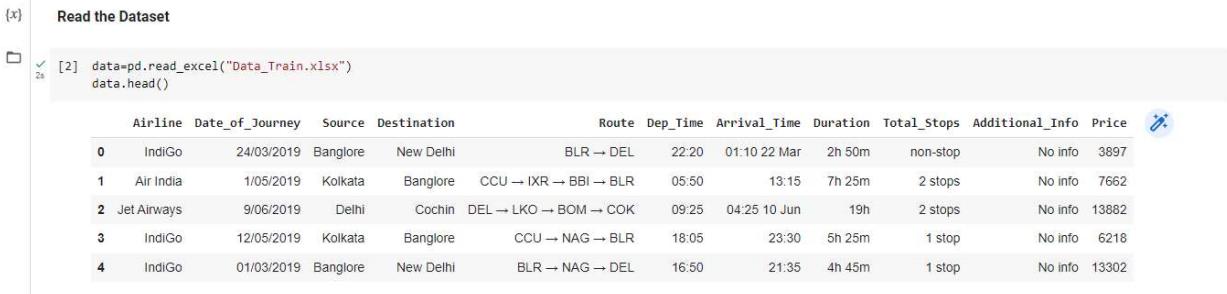
#### Import the libraries:



The screenshot shows a Google Colab notebook titled "Flight Price Prediction.ipynb". The code cell contains imports for numpy, pandas, matplotlib, seaborn, and various classifiers from scikit-learn. The code is as follows:

```
[1] 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
    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')
```

#### Read the data set:



The screenshot shows a Google Colab notebook. A code cell reads a dataset from "Data\_Train.xlsx" and displays its head. The code is:

```
[2] data=pd.read_excel("Data_Train.xlsx")
    data.head()
```

The resulting DataFrame is displayed below:

	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

#### Descriptive Statistical:

The screenshot shows a Google Colab notebook titled "Flight Price Prediction.ipynb". The code cell [3] contains the command `data.describe()`, which outputs a detailed statistical summary of the dataset. The code cell [4] defines a list of categorical columns: ['Airline', 'Source', 'Destination', 'Additional\_Info']. The status bar at the bottom indicates the code completed at 7:52PM.

```
+ Code + Text  
Descriptive Statistical  
{x} [3] 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  
<> [4] categorical=['Airline','Source','Destination','Additional_Info']  
categorical  
['Airline', 'Source', 'Destination', 'Additional_Info']  
0s completed at 7:52PM  
Type here to search
```

## Data Preparation:

### Check the unique values:

The screenshot shows a Google Colab notebook titled "Flight Price Prediction.ipynb". The code cell [5] uses a for loop to print the unique values for each categorical column. The output lists all unique categories for Airline, Source, Destination, and Additional\_Info.

```
+ Code + Text  
Data Preparation  
{x} Checking Unique values  
[5] for i in categorical:  
    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']
```

### Split the data column:

```
[6] data.Date_of_Journey=data.Date_of_Journey.str.split('/')
data.Date_of_Journey
```

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

Check the maximum Number of stops:

```
[8] data.Total_Stops.unique()
```

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

Split the Values:

```
[9] data.Route=data.Route.str.split('>')
data.Route
```

```
[BLR , DEL]
[CCU , IXR , BBI , BLR]
[DEL , LKO , BOM , COK]
[CCU , NAG , BLR]
[BLR , NAG , DEL]
...
[CCU , BLR]
[CCU , BLR]
[BLR , DEL]
[BLR , DEL]
[DEL , GOI , BOM , COK]
Name: Route, Length: 10683, dtype: object
```

The image shows two vertically stacked screenshots of a Google Colab notebook titled "Flight Price Prediction.ipynb".

**Top Screenshot:**

```

0s [10] data['City1']=data.Route.str[0]
0s data['City2']=data.Route.str[1]
0s data['City3']=data.Route.str[2]
0s data['City4']=data.Route.str[3]
0s data['City5']=data.Route.str[4]
0s data['City6']=data.Route.str[5]

0s [11] data.Dep_Time=data.Dep_Time.str.split(':')

0s [12] data['Dep_Time_Hour']=data.Dep_Time.str[0]
0s data['Dep_Time_Mins']=data.Dep_Time.str[1]

0s [13] data.Arrival_Time=data.Arrival_Time.str.split(' ')
0s [14] data['Arrival_date']=data.Arrival_Time.str[1]
0s data['Time_of_Arrival']=data.Arrival_Time.str[0]

0s [15] data['Time_of_Arrival']=data.Time_of_Arrival.str.split(':')

0s [16] data['Arrival_Time_Hour']=data.Time_of_Arrival.str[0]
0s data['Arrival_Time_Mins']=data.Time_of_Arrival.str[1]

```

**Bottom Screenshot:**

```

0s [17] data.Duration=data.Duration.str.split(' ')
0s [18] data['Travel_Hours']=data.Duration.str[0]
0s data['Travel_Hours']=data['Travel_Hours'].str.split('h')
0s data['Travel_Hours']=data['Travel_Hours'].str[0]
0s data.Travel_Hours=data.Travel_Hours
0s data['Travel_Mins']=data.Duration.str[1]

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

```

Replace non-stop flights with 0 value:

A screenshot of a Jupyter Notebook cell containing the following code:

```

Replace non-stop flights with 0 value

0s [19] data.Total_Stops.replace('non_stop',0,inplace=True)
0s data.Total_Stops=data.Total_Stops.str.split(' ')
0s data.Total_Stops=data.Total_Stops.str[0]

```

## Check the additional information:

```
✓ [20] data.Additional_Info.unique()
[ ] 0s
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)
```

```
✓ [21] data.Additional_Info.replace('No Info','No Info',inplace=True)
```

## Check the null values:

```
{x} Check the null values
[ ] 0s
[ ] data.isnull().sum()
[ ] Airline      0
[ ] Date_of_Journey 0
[ ] Source      0
[ ] Destination 0
[ ] Route        1
{x} 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
[ ] City5        10637
[ ] City6        10682
[ ] 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
```

## Drop the columns:

```
Drop the columns
[ ] 0s
{x} ✓ [22] data.drop(['City4','City5','City6'],axis=1,inplace=True)
{x} ✓ [23] data.drop(['Date_of_Journey','Route','Dep_Time','Arrival_Time','Duration'],axis=1,inplace=True)
[ ] data.drop(['Time_of_Arrival'],axis=1,inplace=True)
```

## Checking null values:

The screenshot shows a Jupyter Notebook interface with the title "Flight Price Prediction.ipynb". The code cell [24] contains the command `data.isnull().sum()`. The output shows the count of null values for each column:

Column	Count
Airline	0
Source	0
Destination	0
Total_Stops	1
Additional_Info	0
Price	0
Date	0
Month	0
Year	0
City1	1
City2	1
City3	3492
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

The column "dtype: int64" is also listed at the bottom.

## Replacing missing values:

The screenshot shows a Jupyter Notebook interface with the title "Flight Price Prediction.ipynb". The code cells [25], [26], and [27] contain commands to replace missing values with "None":

- [25] `data['City1'].fillna('None', inplace=True)`
- [25] `data['City2'].fillna('None', inplace=True)`
- [25] `data['City3'].fillna('None', inplace=True)`
- [25] `data['Total_Stops'].fillna('None', inplace=True)`
- [26] `data['Arrival_date'].fillna(data['Date'], inplace=True)`
- [27] `data['Travel_Mins'].fillna(0, inplace=True)`

## Check Info:

The screenshot shows a Jupyter Notebook interface with the title "Flight Price Prediction.ipynb". The code cell [28] contains the command `data.info()`. The output provides detailed information about the DataFrame:

Column	Non-Null Count	Dtype
Airline	10683	object
Source	10683	object
Destination	10683	object
Total_Stops	10683	object
Additional_Info	10683	object
Price	10683	int64
Date	10683	object
Month	10683	object
Year	10683	object
City1	10683	object
City2	10683	object
City3	10683	object
Dep_Time_Hour	10683	object
Dep_Time_Mins	10683	object
Arrival_date	10683	object
Arrival_Time_Hour	10683	object
Arrival_Time_Mins	10683	object
Travel_Hours	10683	object
Travel_Mins	10683	object

Additional details from the output:

- dtypes: int64(1), object(18)
- memory usage: 1.5+ MB

## Change the Datatype:

The screenshot shows two code cells in a Google Colab notebook titled "Flight Price Prediction.ipynb".

**Code Cell 1:**

```
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_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_Mins=data.Travel_Mins.astype('int64')

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

This cell contains code to change various date and time columns to integers and a filter to select rows where Travel\_Hours is '5m'.

**Code Cell 2:**

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

[32] data.Travel_Hours=data.Travel_Hours.astype('int64')

[33] 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']
```

This cell contains code to drop index 6474, change the Travel\_Hours column to integer type, and define categorical and numerical columns for a machine learning pipeline.

## Label Encoding:

```
Label Encoding

[34]: from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()

[35]: data.Airline=le.fit_transform(data.Airline)
data.Source=le.fit_transform(data.Source)
data.Destination=le.fit_transform(data.Destination)
data.Total_Stops=le.fit_transform(data.Total_Stops)
data.City1=le.fit_transform(data.City1)
data.City2=le.fit_transform(data.City2)
data.City3=le.fit_transform(data.City3)
data.Additional_Info=le.fit_transform(data.Additional_Info)
data.head()

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

[36]: data.head()

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

## Output columns:

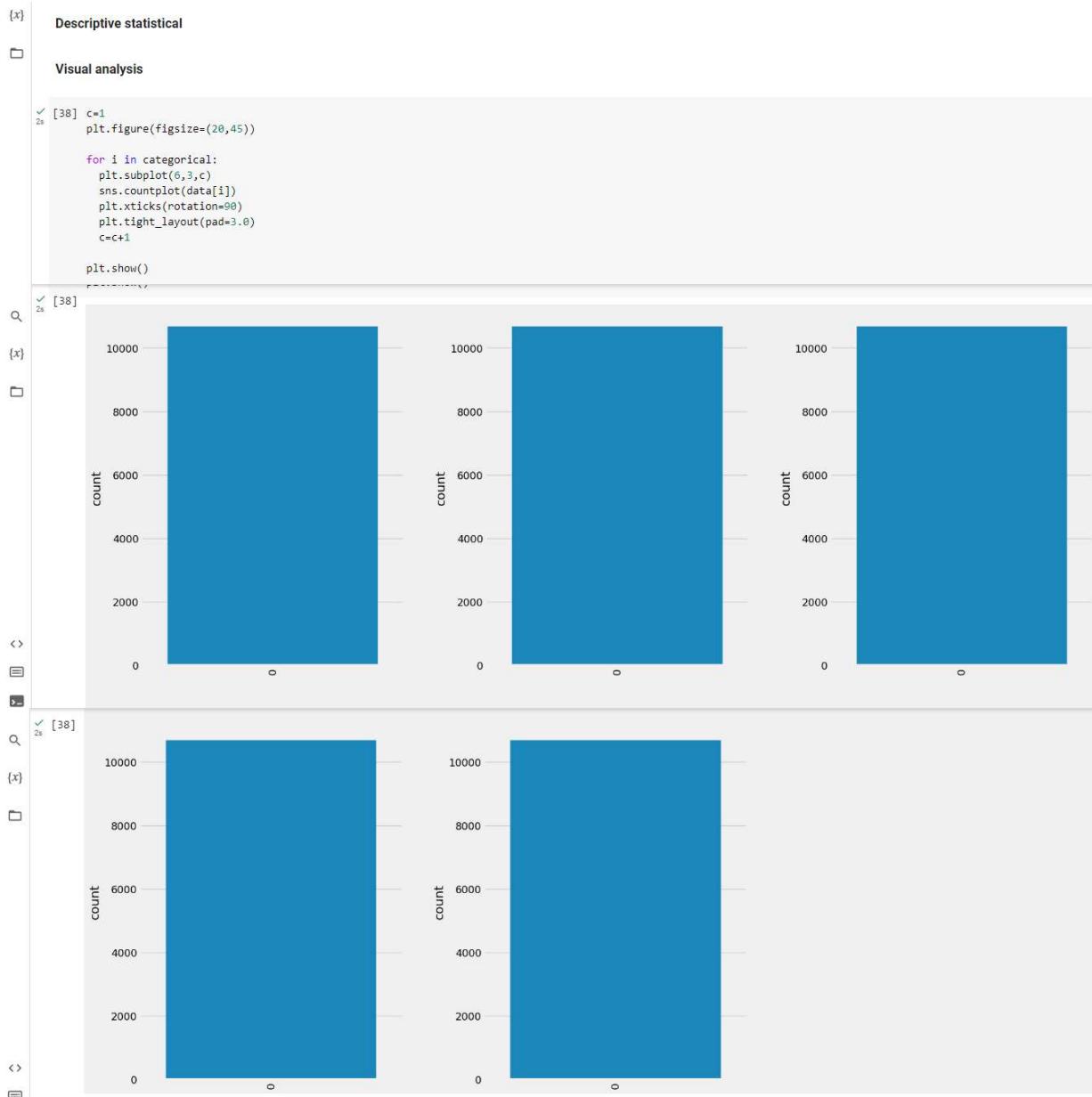
```
Output Columns

[37]: data.head()

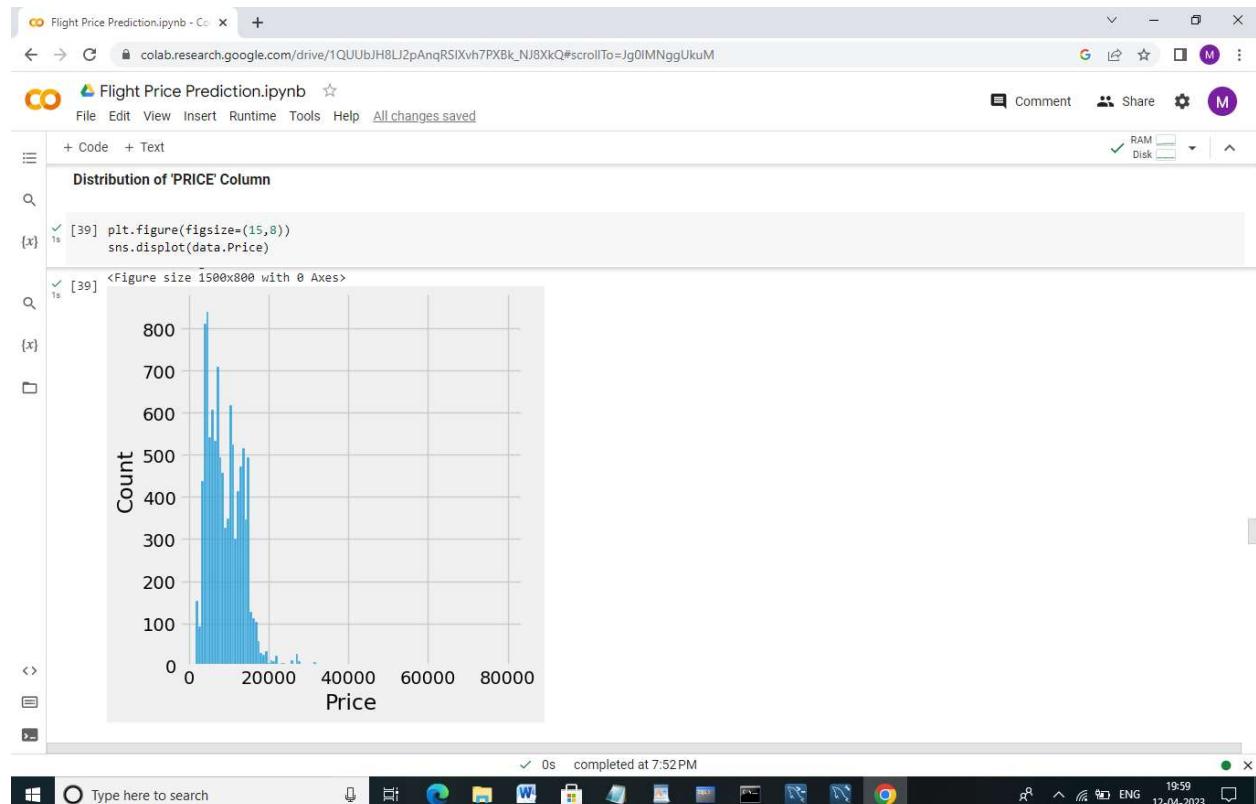
Airline Source Destination Total_Stops Additional_Info Price Date Month Year City1 City2 City3 Dep_Time_Hour Dep_Time_Mins Arrival_date Arrival_T:
0 3 0 5 5 8 3897 24 3 2019 0 13 29 22 20 22
1 1 3 0 1 8 7662 1 5 2019 2 25 1 5 50 1
2 4 2 1 1 8 13882 9 6 2019 3 32 4 9 25 10
3 3 3 0 0 8 6218 12 5 2019 2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

## Descriptive Statistical:

## Visual Analysis:



## Distribution of 'PRICE' Column:



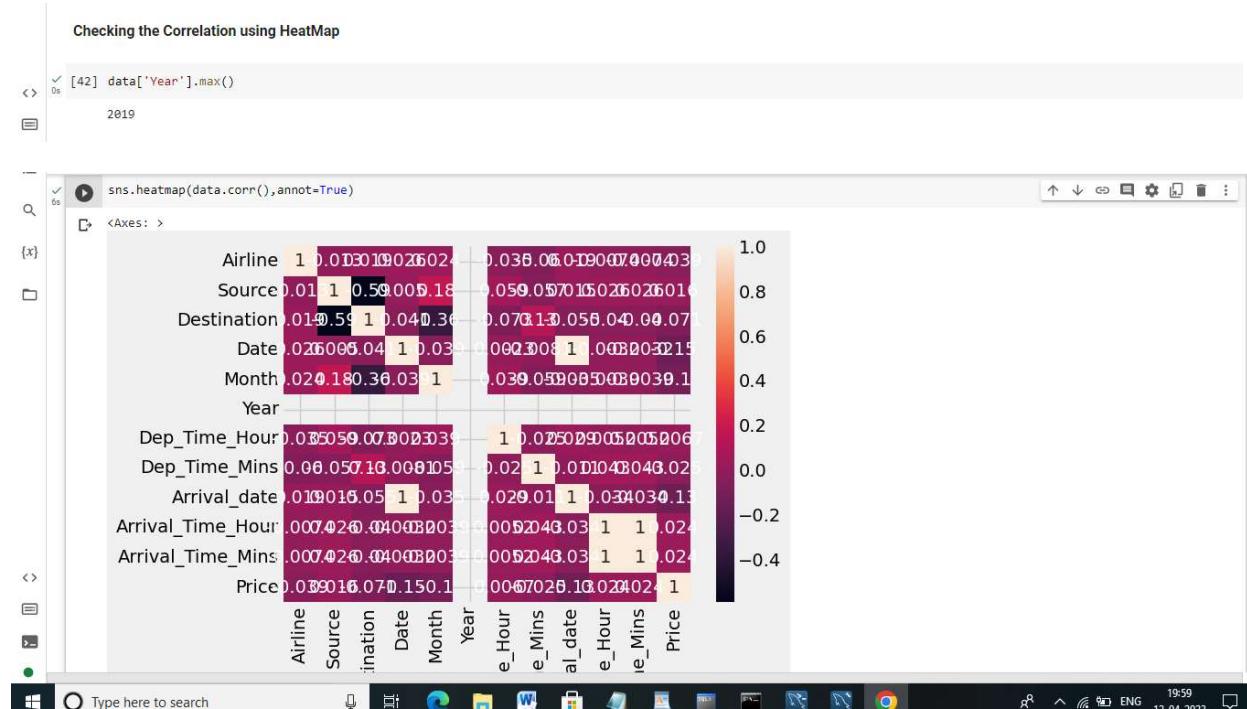
## Output Column:

A screenshot of a Jupyter Notebook cell displaying the first five rows of a dataset. The columns listed are Airline, Source, Destination, Date, Month, Year, Dep\_Time\_Hour, Dep\_Time\_Mins, Arrival\_date, Arrival\_Time\_Hour, Arrival\_Time\_Mins, and Price. The data shows various flight details and their corresponding prices.

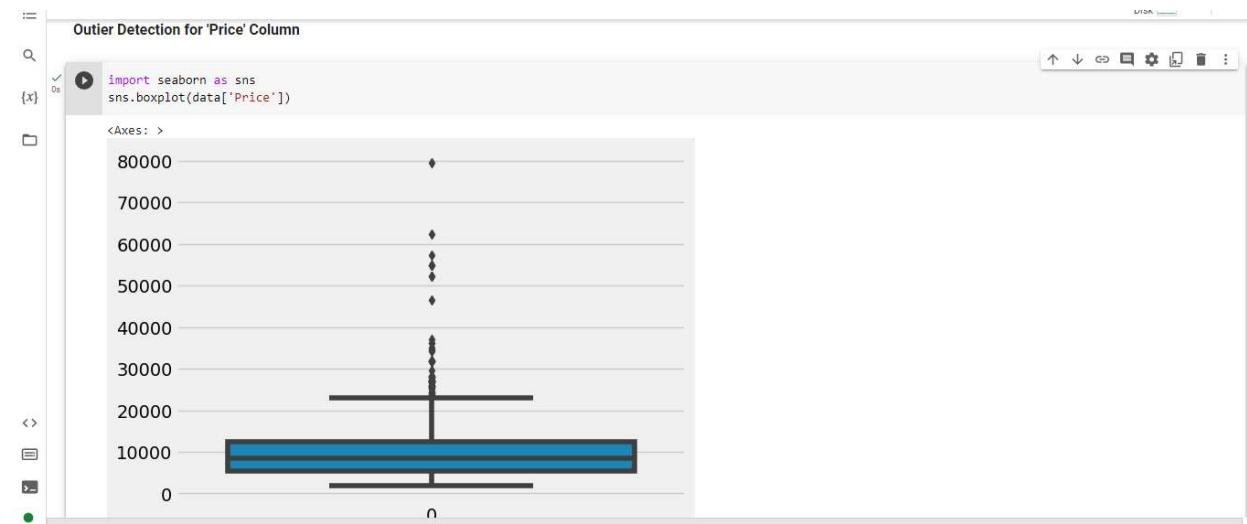
	Airline	Source	Destination	Date	Month	Year	Dep_Time_Hour	Dep_Time_Mins	Arrival_date	Arrival_Time_Hour	Arrival_Time_Mins	Price
0	3	0	5	24	3	2019	22	20	22	1	1	3897
1	1	3	0	1	5	2019	5	50	1	13	13	7662
2	4	2	1	9	6	2019	9	25	10	4	4	13882
3	3	3	0	12	5	2019	18	5	12	23	23	6218
4	3	0	5	1	3	2019	16	50	1	21	21	13302

```
[40] data=data[['Airline','Source','Destination','Date','Month','Year','Dep_Time_Hour','Dep_Time_Mins','Arrival_date','Arrival_Time_Hour','Arrival_Time_Mins','Price']]
[41] data.head()
```

## Checking the correlation using HeatMap:



## Outlier Detection for 'PRICE' column:



## Scaling the data:

```
Scaling the Data

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

[46] from sklearn.preprocessing import StandardScaler
     ss=StandardScaler()

[47] x_scaled=ss.fit_transform(x)

{x} [48] 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   -0.410805  -1.658435  2.416778  1.237288  -1.467707  0.0   1.654268  -0.234932  0.955750  -1.800319  -1.800319
1   -1.261152  0.890299  -0.973732  -1.475307  0.250153  0.0   -1.303000  1.363674  -1.524648  -0.050813  -0.050813
2    0.014369  0.040721  -0.295630  -0.531796  1.109082  0.0   -0.607172  0.031502  -0.461621  -1.362943  -1.362943
3   -0.410805  0.890299  -0.973732  -0.177979  0.250153  0.0   0.958440  -1.034235  -0.225392  1.407109  1.407109
4   -0.410805  -1.658435  2.416778  -1.475307  -1.467707  0.0   0.610527  1.363674  -1.524648  1.115525  1.115525
```

## Splitting data into train and test:

```
Splitting data into train and test

[49] 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} [50] x_train.head()

Airline  Source  Destination  Date  Month  Year  Dep_Time_Hour  Dep_Time_Mins  Arrival_date  Arrival_Time_Hour  Arrival_Time_Mins
10005   6        2          1    27    5  2019           8            30          27           19             19
3684    4        2          1     9    5  2019           11           30          10           12             12
1034    8        2          1    24    4  2019           15            45          24           22             22
3909    6        2          1    21    3  2019           12            50          22            1              1
3088    1        2          1    24    6  2019           17            15          25           19             19
```

## Using Ensemble techniques:

```
Using Ensemble Technique

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

```

[52] 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 0.8412340005067588
R2 for train data 0.9469745512589411
Mean Absolute Error is 1227.442066923497
Mean Squared Error is 3356157.945718379
Root Mean Squared Error is 1831.9819719960071
GradientBoostingRegressor()
R2 score is 0.7659806545178971
R2 for train data 0.730149720400205
Mean Absolute Error is 1687.5222333706563
Mean Squared Error is 4946940.877210986
Root Mean Squared Error is 2224.171773314954

```

## Regression Model:

Regression Model

```

[53] 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 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))

SVR()
R2 score is -0.03013111523471812
R2 for train data -0.02307250115390276
Mean Absolute Error is 3629.626247681
Mean Squared Error is 21775964.240214497
Root Mean Squared Error is 4666.4723550252065

```

## Checking cross validation for RandomForestRegressor:

```
Checking Cross Validation for RandomForestRegressor

[54] 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() 0.7831682532521107
RandomForestRegressor() 0.7858397445972436
RandomForestRegressor() 0.7907232713280463
```

## Hypertuning the Model:

```
Hypertuning the model

[55] from sklearn.model_selection import RandomizedSearchCV

[56] 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)

[56] Fitting 3 folds for each of 10 candidates, totalling 30 fits
     RandomizedSearchCV
     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)
         > estimator: RandomForestRegressor
             > RandomForestRegressor

[57] 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
     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)
         > estimator: GradientBoostingRegressor
             > GradientBoostingRegressor
```

## Accuracy:

```
Accuracy

[58] 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("train accuracy",r2_score(y_test_pred,y_test))

train accuracy 0.9226894111129454
train accuracy 0.7530453919241885
```

## Checking Train and Test Accuracy by RandomSearch using KNN Model2:

```
Checking Train and Test Accuracy by RandomSearchCV using KNN Model2

[59]: 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("train accuracy",r2_score(y_test_pred,y_test))

train accuracy 0.8278803211792515
train accuracy 0.3768761884959687
```

## Evaluating performance of the model and saving the model:

```
Evaluating performance of the model and saving the model

[61]: 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("train accuracy",r2_score(y_test_pred,y_test))

train accuracy 0.9222765980728352
train accuracy 0.75251903350719

[62]: prices=rfr.predict(x_test)

[64]: price_list=pd.DataFrame({'Price':prices})

[65]: price_list

[65]:      Price
0    14620.30
1     5990.80
2     8887.80
3    3774.10
4    15391.85
...
2132  13587.00
2133  6683.90
2134  6718.70
2135  11304.20
2136  11536.90
2137 rows × 1 columns

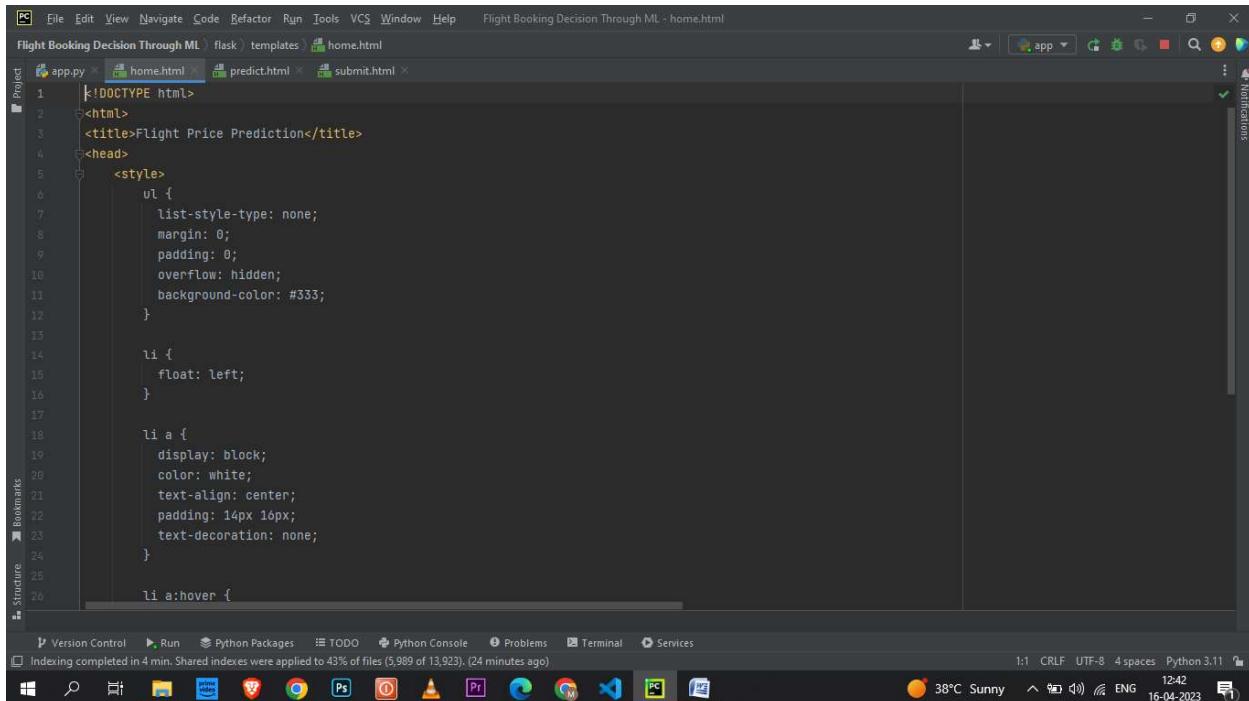
[66]: import pickle
       pickle.dump(rfr,open('model1.pk1','wb'))
```

## Save the best model:

```
Save the best model

[67]: import pickle
       pickle.dump(rfr,open('model1.pk1','wb'))
```

## Home.html:

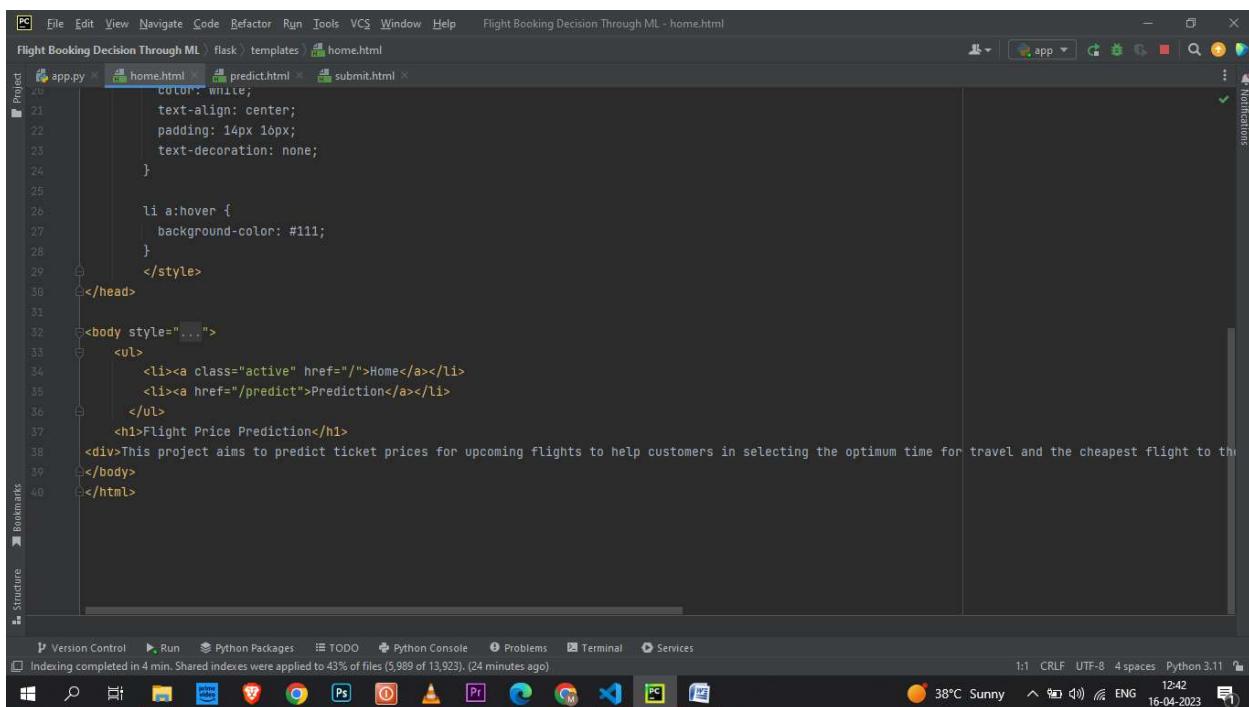


```
<!DOCTYPE html>
<html>
<title>Flight Price Prediction</title>
<head>
    <style>
        ul {
            list-style-type: none;
            margin: 0;
            padding: 0;
            overflow: hidden;
            background-color: #333;
        }

        li {
            float: left;
        }

        li a {
            display: block;
            color: white;
            text-align: center;
            padding: 14px 16px;
            text-decoration: none;
        }

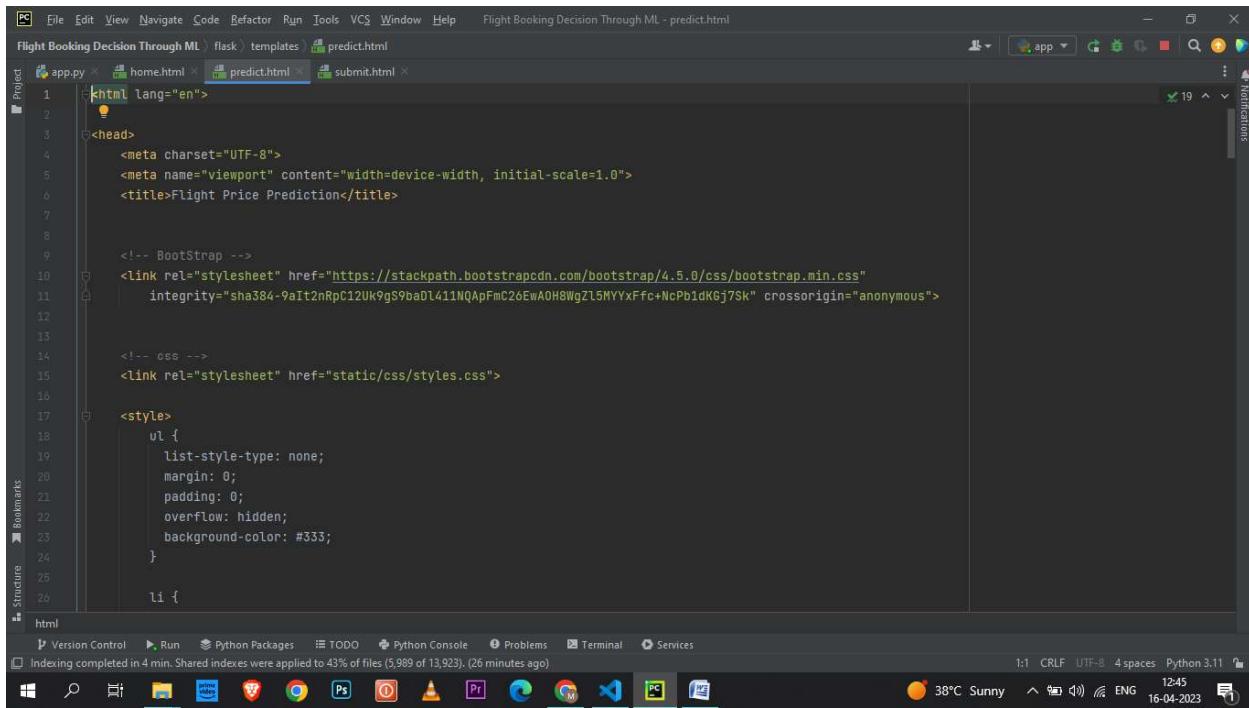
        li a:hover {
```



```
        color: white;
        text-align: center;
        padding: 14px 16px;
        text-decoration: none;
    }

    li a:hover {
        background-color: #111;
    }
</style>
</head>
<body style="...">
    <ul>
        <li><a class="active" href="/">Home</a></li>
        <li><a href="/predict">Prediction</a></li>
    </ul>
    <h1>Flight Price Prediction</h1>
    <div>This project aims to predict ticket prices for upcoming flights to help customers in selecting the optimum time for travel and the cheapest flight to their destination.</div>
</body>
</html>
```

## predict.html:



```
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Flight Price Prediction</title>

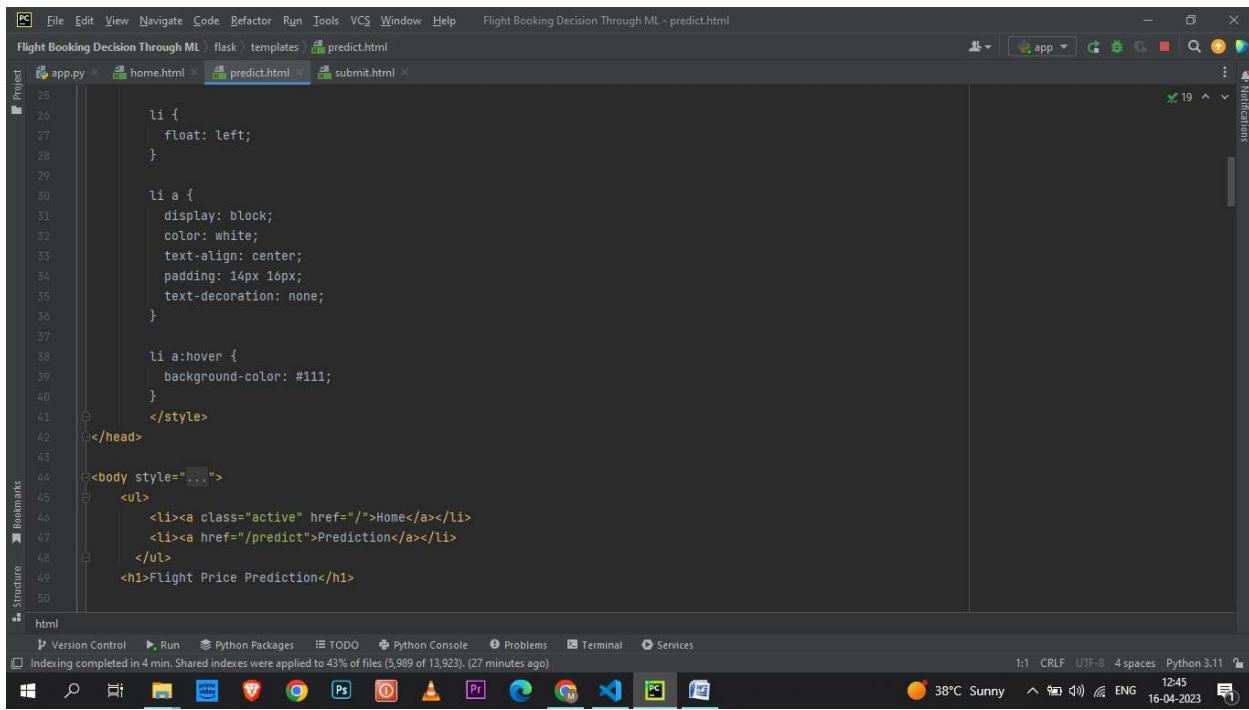
        <!-- Bootstrap -->
        <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css"
              integrity="sha384-ShJfK7tIj+L8iRJ3ZCq3Dd7gRbGJ30oQKXKuJX3O+QkZPfKXoYnXWzJG&lt;!-->
              crossorigin="anonymous">

        <!-- CSS -->
        <link rel="stylesheet" href="static/css/styles.css">

    <style>
        ul {
            list-style-type: none;
            margin: 0;
            padding: 0;
            overflow: hidden;
            background-color: #333;
        }

        li {

```



```
        li {
            float: left;
        }

        li a {
            display: block;
            color: white;
            text-align: center;
            padding: 14px 16px;
            text-decoration: none;
        }

        li a:hover {
            background-color: #111;
        }
    </style>
</head>

<body style="...>
    <ul>
        <li><a class="active" href="/">Home</a></li>
        <li><a href="/predict">Prediction</a></li>
    </ul>
    <h1>Flight Price Prediction</h1>
```

```
PC File Edit View Navigate Code Refactor Run Tools VCS Window Help Flight Booking Decision Through ML - predict.html
Flight Booking Decision Through ML flask> templates predict.html submit.html
Project app.py x home.html x predict.html x submit.html x
49
50 <h1>Flight Price Prediction</h1>
51
52 <!-- As a heading -->
53 <nav class="navbar navbar-inverse navbar-fixed-top">
54   <div class="container-fluid">
55     <div class="navbar-header">
56       <a class="navbar-brand" href="/">FLIGHT PRICE PREDICTION</a>
57     </div>
58   </div>
59 </nav> -->
60
61 <br><br><br>
62
63 <div class="container">
64
65
66
67   <form action="\pred" method="post">
68
69     <div class="row">
70       <div class="col-sm-6">
71         <div class="card">
72           <div class="card-body">
73             <h5 class="card-title">Departure Date</h5>
```

```
PC File Edit View Navigate Code Refactor Run Tools VCS Window Help Flight Booking Decision Through ML - predict.html
Flight Booking Decision Through ML flask> templates predict.html submit.html
Project app.py x home.html x predict.html x submit.html x
73   <div class="card-body">
74     <h5 class="card-title">Departure Date</h5>
75     <!-- Departure -->
76     <input type="text" name="Date" id="Date" required="required">
77   </div>
78 </div>
79 <div class="col-sm-6">
80   <div class="card">
81     <div class="card-body">
82       <h5 class="card-title">Departure Month</h5>
83       <!-- Departure -->
84       <input type="text" name="Month" id="Month" required="required">
85     </div>
86   </div>
87
88 </div>
89 <br>
90 <div class="row">
91   <div class="col-sm-6">
92     <div class="card">
93       <div class="card-body">
94         <h5 class="card-title">Departure Year</h5>
95         <!-- Departure -->
```

```
<h5 class="card-title">Departure Year</h5>
<!-- Departure -->
<input type="text" name="Year" id="Year" required="required">
</div>
</div>
<div class="col-sm-6">
<div class="card">
<div class="card-body">
<h5 class="card-title">Departure Time Hours</h5>
<!-- Departure -->
<input type="text" name="Dep_Time_Hour" id="Dep_Time_Hour" required="required">
</div>
</div>
<br>
<div class="row">
<div class="col-sm-6">
<div class="card">
<div class="card-body">
<h5 class="card-title">Departure Time Mins</h5>
<!-- Arrival -->
<input type="text" name="Dep_Time_Mins" id="Dep_Time_Mins" required="required">
</div>
</div>
<br>
<div class="col-sm-6">
<div class="card">
<div class="card-body">
<h5 class="card-title">Arrival Time</h5>
<!-- Arrival -->
<input type="text" name="Arrival_date" id="Arrival_date" required="required">
</div>
</div>
<br>
<div class="row">
<div class="col-sm-6">
<div class="card">
<div class="card-body">
<h5 class="card-title">Arrival Time Hours</h5>
<!-- Arrival -->
<input type="text" name="Arrival_Time_Hours" id="Arrival_Time_Hours" required="required">
</div>
</div>
<div class="col-sm-6">
```

```
<input type="text" name="Dep_Time_Mins" id="Dep_Time_Mins" required="required">
</div>
</div>
<div class="col-sm-6">
<div class="card">
<div class="card-body">
<h5 class="card-title">Arrival Time</h5>
<!-- Arrival -->
<input type="text" name="Arrival_date" id="Arrival_date" required="required">
</div>
</div>
<br>
<div class="row">
<div class="col-sm-6">
<div class="card">
<div class="card-body">
<h5 class="card-title">Arrival Time Hours</h5>
<!-- Arrival -->
<input type="text" name="Arrival_Time_Hours" id="Arrival_Time_Hours" required="required">
</div>
</div>
<div class="col-sm-6">
```

The screenshot shows a code editor interface with multiple tabs open. The tabs include 'app.py', 'home.html', 'predict.html' (the current active tab), and 'submit.html'. The code in 'predict.html' is as follows:

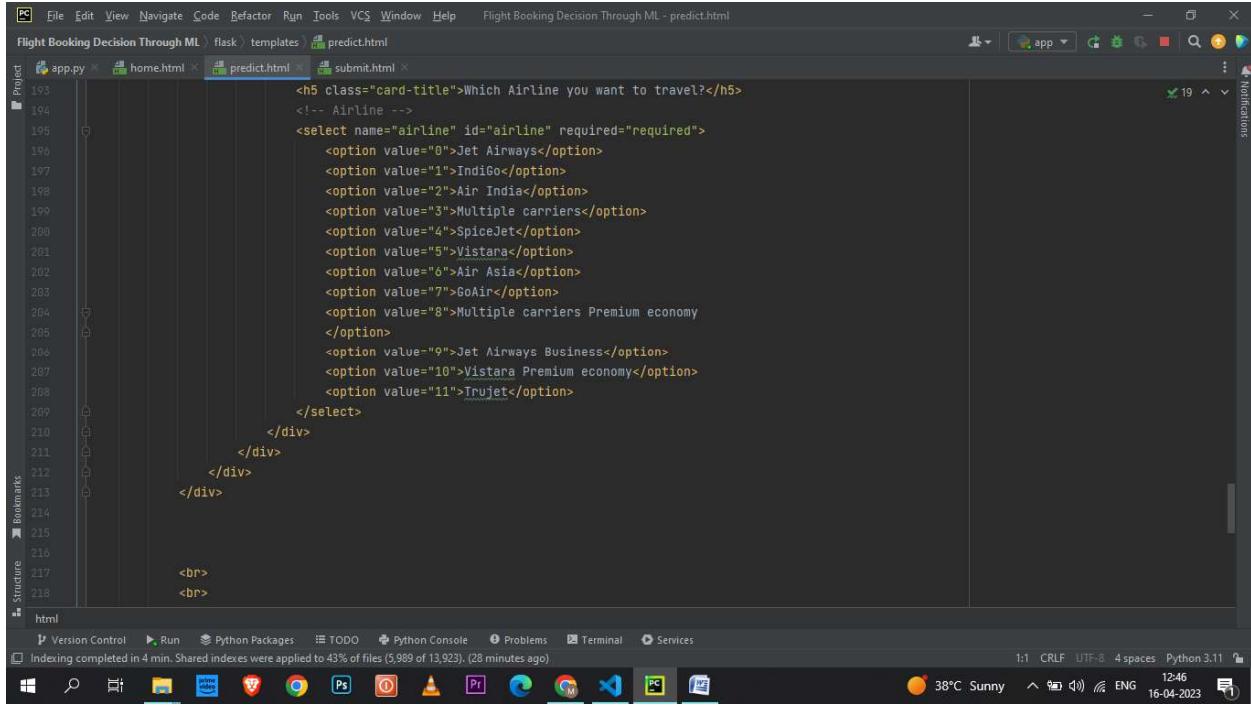
```
</div>
<div class="col-sm-6">
    <div class="card">
        <div class="card-body">
            <h5 class="card-title">Arrival Time Mins</h5>
            <!-- Arrival -->
            <input type="text" name="Arrival_Time_Mins" id="Arrival_Time_Mins" required="required">
        </div>
    </div>
</div>
<br>
<div class="row">
    <div class="col-sm-6">
        <div class="card">
            <div class="card-body">
                <!-- Source -->
                <h5 class="card-title">Source</h5>
                <select name="Source" id="Source" required="required">
                    <option value="0">Delhi</option>
                    <option value="1">Kolkata</option>
                    <option value="2">Mumbai</option>
                    <option value="3">Chennai</option>
                </select>
            </div>
        </div>
    </div>
</div>
```

The screenshot shows the PyCharm IDE interface with the following details:

- Project:** Flight Booking Decision Through ML
- File:** predict.html
- Code Content:** HTML code for a flight booking form. It includes fields for Destination (with options: Cochin, Delhi, New Delhi, Hyderabad, Kolkata) and Airline.

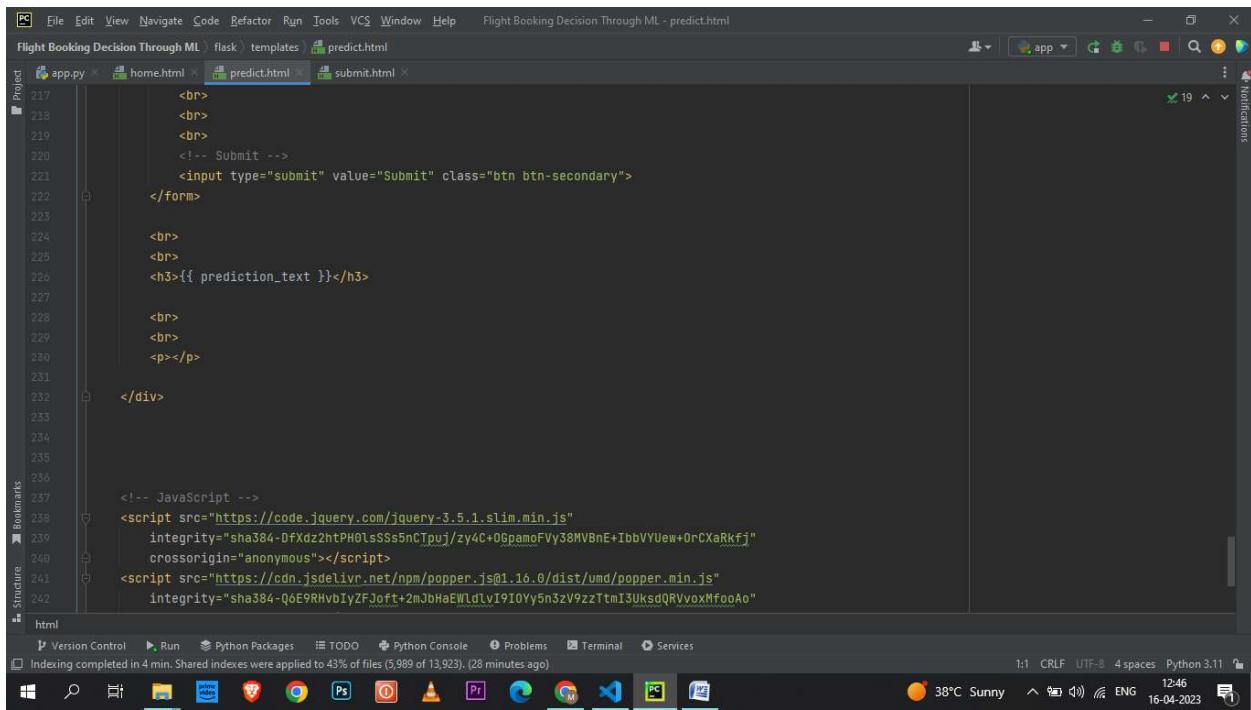
```
<div>
    </div>
</div>
<div class="col-sm-6">
    <div class="card">
        <div class="card-body">
            <h5 class="card-title">Destination</h5>
            <!-- Destination -->
            <select name="Destination" id="Destination" required="required">
                <option value="0">Cochin</option>
                <option value="1">Delhi</option>
                <option value="2">New Delhi</option>
                <option value="3">Hyderabad</option>
                <option value="4">Kolkata</option>
            </select>
        </div>
    </div>
</div>
<br>
<div class="row">
    <div class="col-sm-6">
        <div class="card">
            <div class="card-body">
                <h5 class="card-title">Which Airline you want to travel?</h5>
                <!-- Airline -->
            </div>
        </div>
    </div>
</div>
```

- Toolbars:** Version Control, Run, Python Packages, TODO, Python Console, Problems, Terminal, Services
- Status Bar:** Indexing completed in 4 min. Shared indexes were applied to 43% of files (5,989 of 13,923). (28 minutes ago), 1:1, CRLF, UTF-8, 4 spaces, Python 3.11, 38°C Sunny, 12:46, 16-04-2023



```
<h5 class="card-title">Which Airline you want to travel?</h5>
<!-- Airline -->
<select name="airline" id="airline" required="required">
    <option value="0">Jet Airways</option>
    <option value="1">IndiGo</option>
    <option value="2">Air India</option>
    <option value="3">Multiple carriers</option>
    <option value="4">SpiceJet</option>
    <option value="5">Vistara</option>
    <option value="6">Air Asia</option>
    <option value="7">GoAir</option>
    <option value="8">Multiple carriers Premium economy
    </option>
    <option value="9">Jet Airways Business</option>
    <option value="10">Vistara Premium economy</option>
    <option value="11">TruJet</option>
</select>
</div>
</div>
</div>

<br>
<br>
```



```
<br>
<br>
<br>
<!-- Submit -->
<input type="submit" value="Submit" class="btn btn-secondary">
</form>

<br>
<br>
<h3>{{ prediction_text }}</h3>
<br>
<br>
<p></p>
</div>

<!-- JavaScript -->
<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"
    integrity="sha384-DfXd2htPH0sSS5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkfj"
    crossorigin="anonymous"></script>
<script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js"
    integrity="sha384-QfE9HbIyZfQ2oEFzqmpc4fTQJ4Z+aWgBZQ8kXOwqDZLXuHdC" data-bbox="188 718 580 745" data-kind="parent"></script>
```

A screenshot of a code editor (PyCharm) displaying the file `predict.html`. The code is an HTML page with a script section containing several external JavaScript and CSS imports from CDNs. The code editor interface includes tabs for `app.py`, `home.html`, `predict.html`, and `submit.html`. The bottom status bar shows system information like temperature (38°C), battery level (Sunny), and date (16-04-2023).

```
<p></p>
<div>
<!-- JavaScript -->
<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"
integrity="sha384-Dfxz2htPHols5nCTpuj/zY4C+06pamoFV38MVBN+E+bvYYUew+OrCxarKfj"
crossorigin="anonymous"></script>
<script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/vm/popper.min.js"
integrity="sha384-Q6e9RHbIyZFOft+2J0HAEWLdVi91OYy5n3zV9zTTin3UksdQRVvoxMfooAo"
crossorigin="anonymous"></script>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/js/bootstrap.min.js"
integrity="sha384-OgVRvuATP1z7Jjlku0U7Xw704+h835Lr+6QL9UvYjZE3Ipu6Tp75j7Bh/kR0JKI"
crossorigin="anonymous"></script>
</body>
</html>
```

## submit.html:

A screenshot of a code editor (PyCharm) displaying the file `submit.html`. The code is an HTML page with a title and a style section containing CSS rules for a navigation menu. The code editor interface includes tabs for `app.py`, `home.html`, `predict.html`, and `submit.html`. The bottom status bar shows system information like temperature (39°C), battery level (Sunny), and date (16-04-2023).

```
<!DOCTYPE html>
<html>
<title>Flight Price Prediction</title>
<head>
<style>
ul {
list-style-type: none;
margin: 0;
padding: 0;
overflow: hidden;
background-color: #333;
}

li {
float: left;
}

li a {
display: block;
color: white;
text-align: center;
padding: 14px 16px;
text-decoration: none;
}

li a:hover {
```

Flight Booking Decision Through ML - submit.html

```
app.py x home.html x predict.html x submit.html x
20         color: white;
21         text-align: center;
22         padding: 14px 16px;
23         text-decoration: none;
24     }
25
26     li a:hover {
27         background-color: #111;
28     }
29 </style>
30 </head>
31
32 <body style="...">
33     <ul>
34         <li><a class="active" href="/">Home</a></li>
35         <li><a href="/predict">Prediction</a></li>
36     </ul>
37     <h1>Flight Price Prediction</h1>
38     <div>Based on the given input, we can get the flight price as {{prediction_text }}</div>
39 </body>
40 </html>
```

app.py:

The screenshot shows the PyCharm IDE interface with the following details:

- File Menu:** File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, Help.
- Title Bar:** Flight Booking Decision Through ML - flask > app.py
- Project Tree:** Shows 'app.py' as the active file.
- Code Editor:** The content of 'app.py' is displayed:

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

model = pickle.load(open(r"model.pkl", 'rb'))
app = Flask(__name__)
@app.route("/")
def home():
    return render_template('home.html')
@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)

if __name__ == '__main__':
    app.run()
```
- Toolbars:** Version Control, Run, Python Packages, TODO, Python Console, Problems, Terminal, Services.
- Status Bar:** PEP 8: E101 indentation contains mixed spaces and tabs. PEP 8: W191 indentation contains tabs., 27:24, CRLF, UTF-8, 4 spaces, Python 3.11, 12:57, 39°C Sunny, ENG, 16-04-2023.

The screenshot shows the PyCharm IDE interface with the following details:

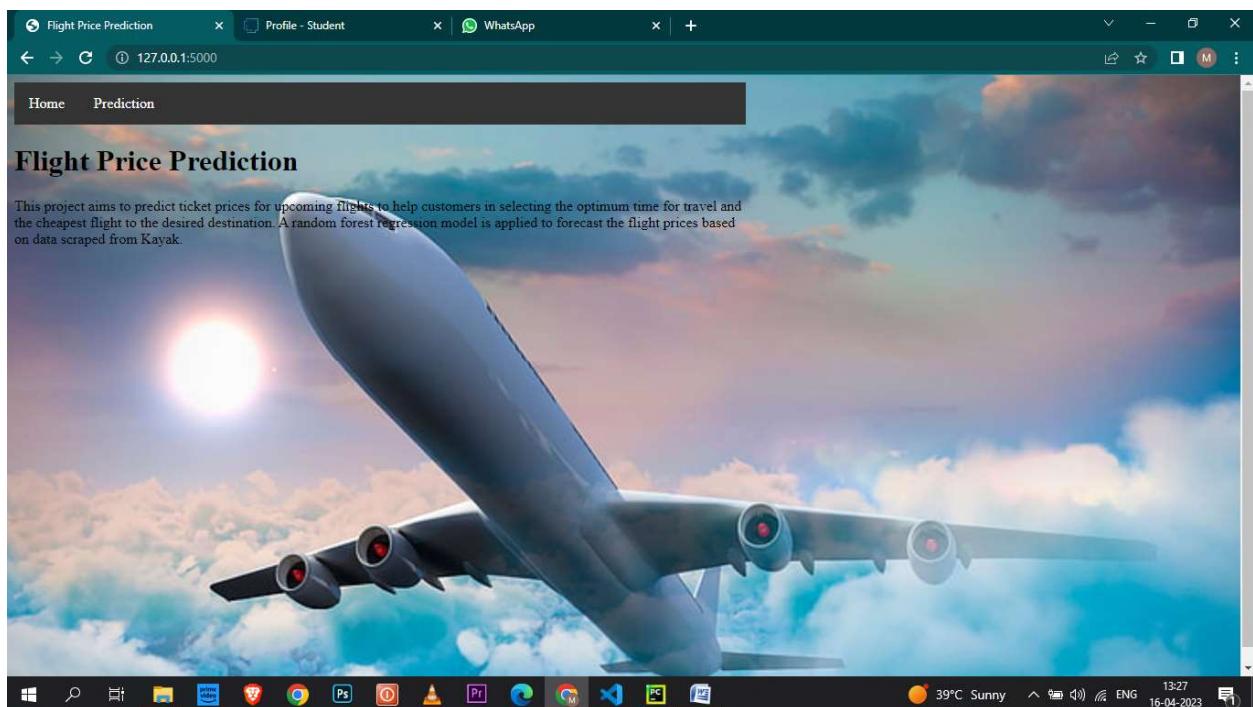
- Project:** Flight Booking Decision Through ML
- File:** app.py
- Code:** Python code for a Flask application. The code includes importing numpy, defining a predict function, and running the app in debug mode if the script is run directly.

```
15: x=[int(x) for x in request.form.values()]
16: print(x)
17: x = np.array(x)
18: print(x.shape)
19:
20: print(x)
21: pred = model.predict(x)
22: print(pred)
23: return render_template('submit.html',prediction_text=pred)
24:
25: if __name__ == '__main__':
26:     app.run(debug=True)
27:
28:
29: if __name__ == '__main__':
30:
```

- Run:** Shows the application is running on `http://127.0.0.1:5000` in debug mode.
- Output:** Displays a warning about using a development server in production and provides the debugger PIN: 839-133-537.
- Bottom Status Bar:** Shows indexing completed, file statistics (261 files, 43% applied), encoding (CRLF), character set (UTF-8), spaces (4 spaces), Python version (Python 3.11), and system information (39°C, Sunny, ENG, 16-04-2023).

## OUTPUT

Home Page:



## Predict page:

This screenshot shows the 'Flight Price Prediction' application running on a Windows 10 desktop. The browser window title is 'Flight Price Prediction' and the URL is '127.0.0.1:5000/predict'. The page displays a flight search form with the following inputs:

- Departure Date: 12
- Departure Month: 07
- Departure Year: 2023
- Departure Time Hours: 10
- Departure Time Mins: 60
- Arrival Time: 11
- Arrival Time Hours: 1
- Arrival Time Mins: 60

The background of the application features a photograph of a large airplane flying through a cloudy sky. The desktop taskbar at the bottom shows various open applications, including Microsoft Word, Excel, and Adobe Photoshop. The system tray indicates it's 12:37 on April 16, 2023, with a temperature of 38°C and sunny weather.

This screenshot shows the same 'Flight Price Prediction' application after some inputs have been changed. The browser window title is 'Flight Price Prediction' and the URL is '127.0.0.1:5000/predict'. The page displays a flight search form with the following inputs:

- Departure Time Mins: 60
- Arrival Time: 11
- Arrival Time Hours: 1
- Arrival Time Mins: 60
- Source: Chennai
- Destination: New Delhi
- Which Airline you want to travel?: Jet Airways

The background of the application features a photograph of a large airplane flying through a cloudy sky. The desktop taskbar at the bottom shows various open applications, including Microsoft Word, Excel, and Adobe Photoshop. The system tray indicates it's 12:38 on April 16, 2023, with a temperature of 38°C and sunny weather.

## Summit Page(Final Output):

