

ARIGNAR ANNA GOVERNMENT ARTS COLLEGE VILLUPURAM – 605 602.



DEPARTMENT OF COMPUTER APPLICATIONS

MACHINE LEARNING WITH PYTHON

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

Team Id : NM2023TMID21142

Team Leader: MAHALAKSHMI R(74CC6BACEDB3ABA98864F72CB0D506BC)

Team member: RANJITHA M(B88D775C056C296832D46E4960D24516)

Team member: KAVIYA M(2283828CC7C0CF4BAD24D671DD0FC65C)

Team member: LOGAPRIYA T(8D8FDFB6D97F5FBFF94E64F6CDFF242)

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 of this customer can be book their tickets of Flight according to their comfortably, according to their affordability, Means whichever cheaper cost they want they can be 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

1 Define a problem statement

Optimizing Flight Booking decisions through Machine Learning price predictions

★ People who frequently travel through flight, will have better knowledge on best discount and right time to buy the ticket. For the business purpose, many airlines companies change prices according to the seasons or time duration.

★ They will increase the price when people travel more. Estimating the highest price of the airline data for the route is collected with features such as duration, Source, Destination, Arrival, Departure.



★ Features are taken from chosen dataset and in this paper, we have used machine learning techniques and regression strategies for prediction of the price whenever the airline price ticket costs very sensitive.

2 Brainstorm

Ideas for Flight Booking decisions and Price Predictions

Maheshkarni R

Analyse historical price data
Using Machine learning algorithm

Ranjitha M

Using Machine learning model to predict the flight prices
Using Machine learning model to predict the flight prices

Kavya M

Price prediction using public data sources
Analyse historical price data

Loganprey T

Performing statistical analysis
Using Linear Regression
Using Data

3 Group Ideas

Group Ideas for Flight Booking decisions and Price Predictions

Analyse historical price data

Using machine learning model to predict the flight prices

Mining airline data to minimize ticket purchase price

Analyse social media platform for price predictions

Artificial Neural Network (ANN)

Using Decision tree

Using KNN

Performing statistical analysis

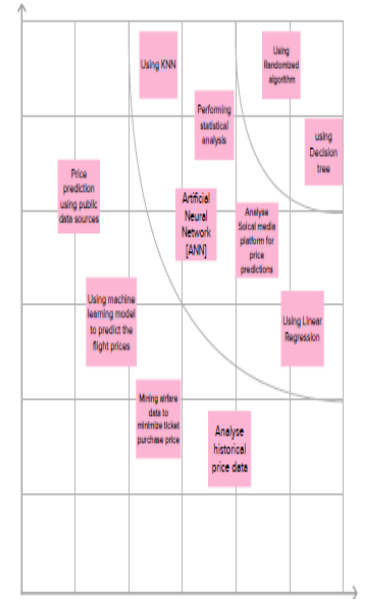
Price prediction using public data sources

Using Randomized algorithm

Using Linear Regression

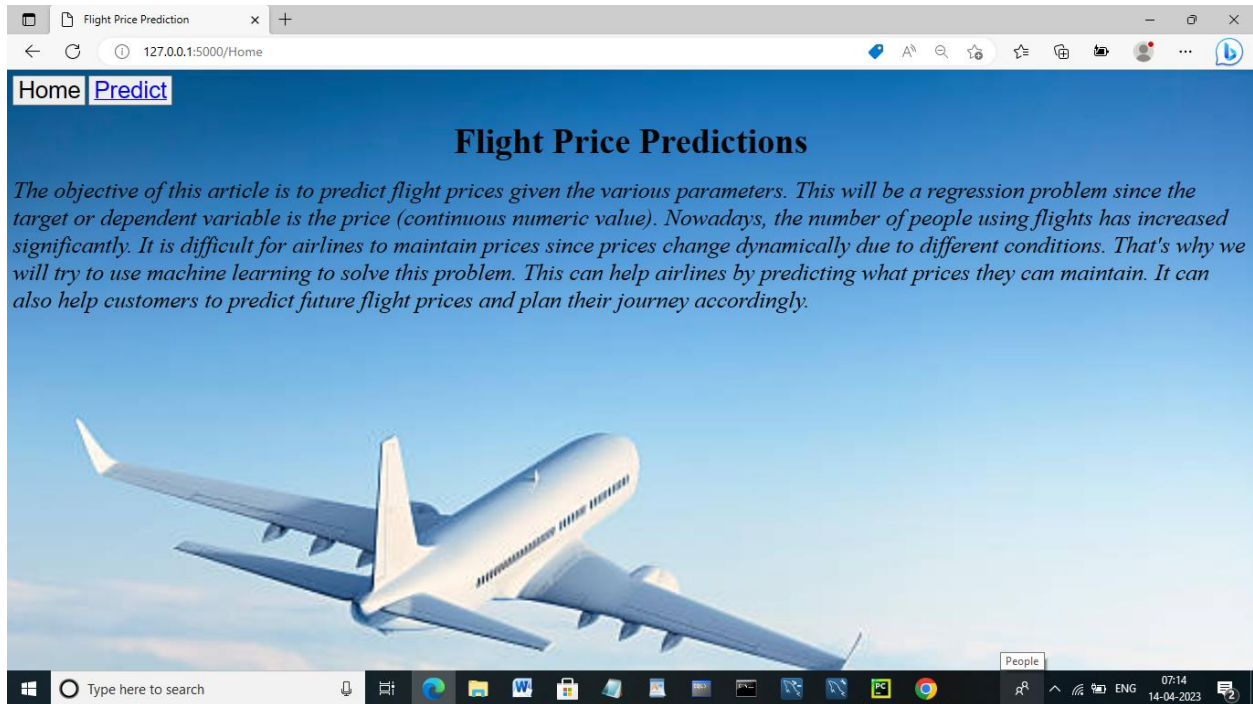
4 Prioritize

Prioritize the Ideas for Flight Booking decisions and Price Predictions



3. RESULT

Home Page:



Predict page:

Flight Price prediction

Airline
Air Asia

Source
Chennai

Destination
Delhi

depdate
13

depmonth
5

depyear
2019

deptimehour
10

deptimemins
20

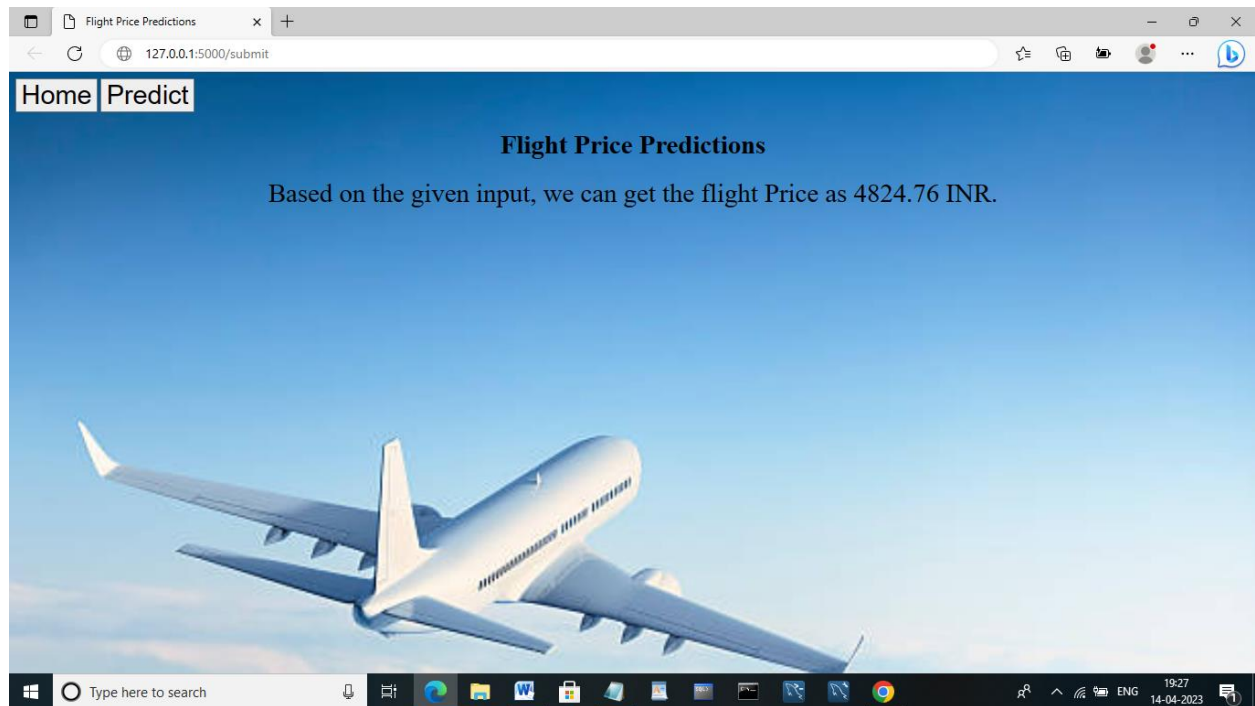
artime
14

artimehour
11

artimemins
15

Submit

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 at 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%.

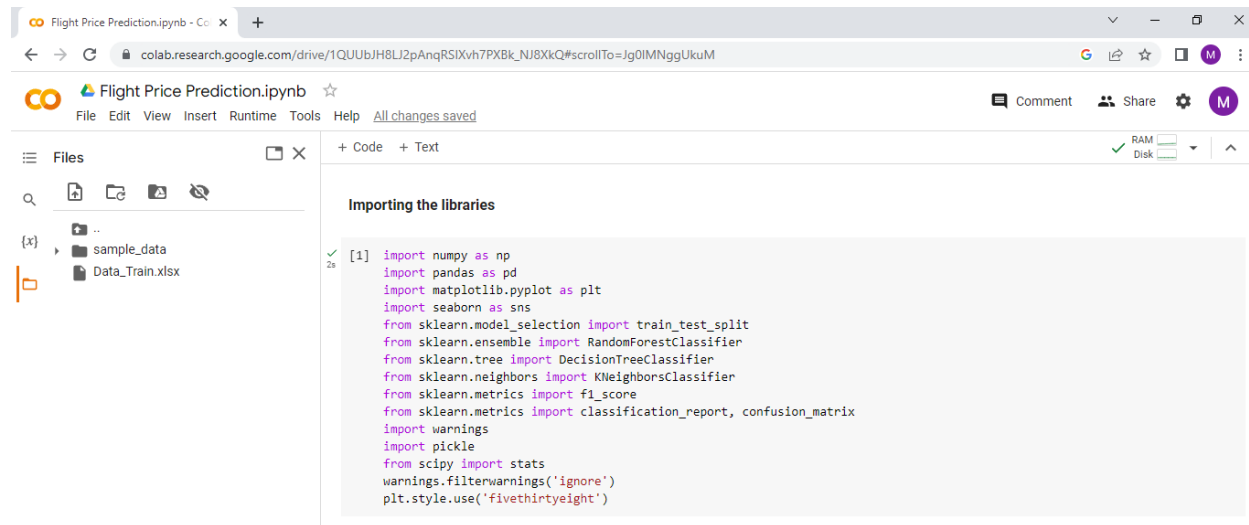
7. FUTURE SCOPE

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 left sidebar shows a file named "Data_Train.xlsx" under a folder named "sample_data". The main code cell, titled "Importing the libraries", contains the following Python code:

```
[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:



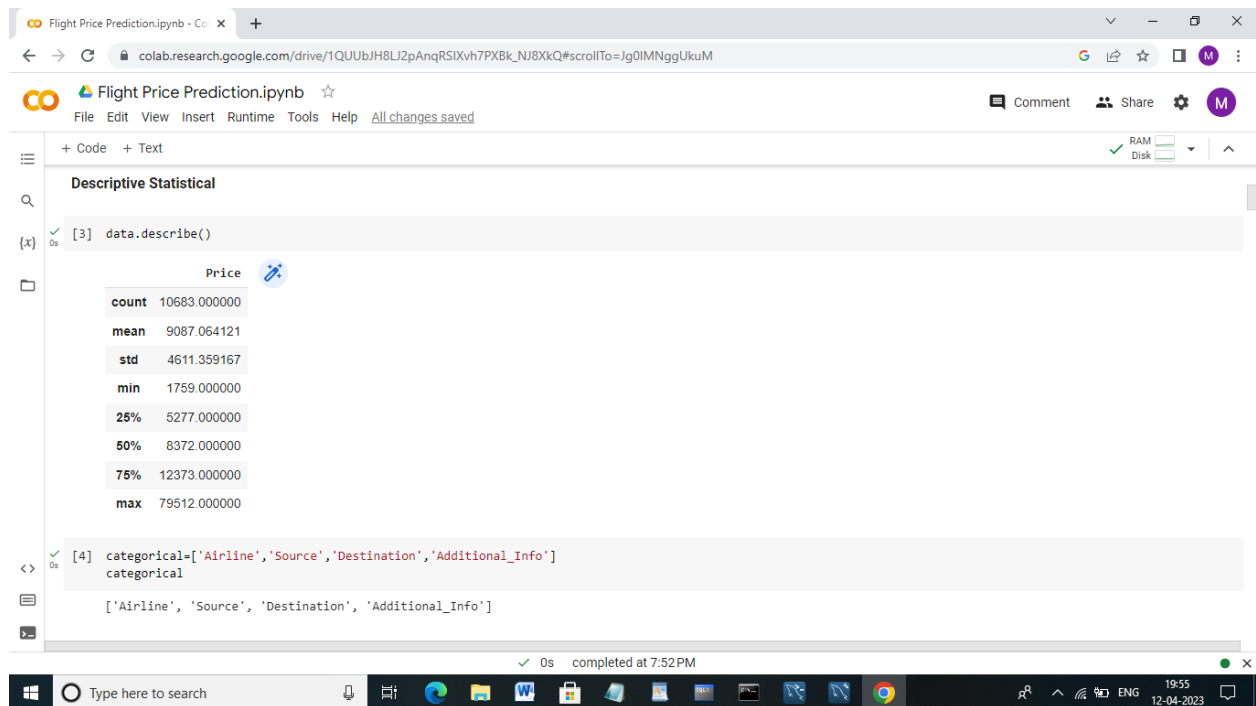
The screenshot shows the same Google Colab notebook. The code cell, titled "Read the Dataset", contains the following Python code:

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

The output of the code is a table showing the first 5 rows of the dataset:

	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 `data.describe()`, which has been executed, resulting in a summary of the data. The summary includes the count, mean, standard deviation, minimum, and maximum values for the 'Price' column, as well as the 25th, 50th (median), and 75th percentiles. The code cell [4] contains `categorical=['Airline','Source','Destination','Additional_Info']` and `categorical`, which has been executed, resulting in the list `['Airline', 'Source', 'Destination', 'Additional_Info']`. The notebook interface includes a menu bar with options like File, Edit, View, Insert, Runtime, Tools, and Help. The status bar at the bottom indicates that the code was completed at 7:52 PM.

```
[3] data.describe()
```

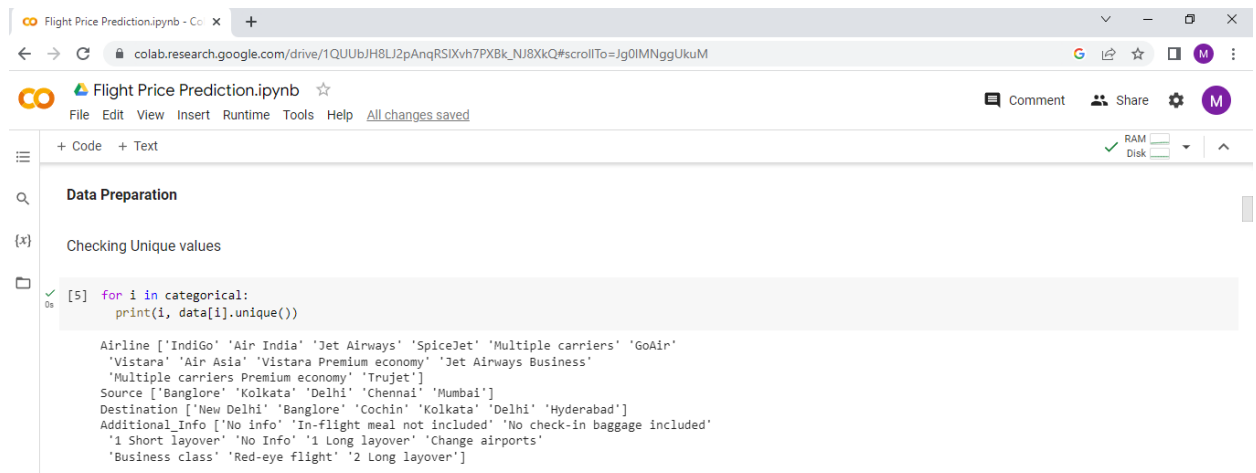
	Price
count	10683.000000
mean	9087.064121
std	4611.359167
min	1759.000000
25%	5277.000000
50%	8372.000000
75%	12373.000000
max	79512.000000

```
[4] categorical=['Airline','Source','Destination','Additional_Info']
categorical

['Airline', 'Source', 'Destination', 'Additional_Info']
```

Data Preparation:

Check the unique values:



The screenshot shows a Google Colab notebook titled "Flight Price Prediction.ipynb". The code cell [5] contains a loop that iterates over the categorical variables and prints the unique values for each. The output shows the unique values for 'Airline', 'Source', 'Destination', and 'Additional_Info'. The notebook interface includes a menu bar with options like File, Edit, View, Insert, Runtime, Tools, and Help. The status bar at the bottom indicates that the code was completed at 7:52 PM.

```
[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 ['Bangalore' 'Kolkata' 'Delhi' 'Chennai' 'Mumbai']
Destination ['New Delhi' 'Bangalore' '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:

The screenshot shows a Jupyter Notebook titled "Flight Price Prediction.ipynb" in a web browser. The notebook has a toolbar with options like File, Edit, View, Insert, Runtime, Tools, and Help. The current cell is titled "Split the data column". It contains two code blocks. The first block (cell [6]) splits the 'Date_of_Journey' column into a list of lists, where each inner list represents a date in [DD, MM, YYYY] format. The second block (cell [7]) assigns the first, second, and third elements of each date list to 'Date', 'Month', and 'Year' columns respectively. The output of cell [6] is visible, showing a sample of the data structure.

```
[6] 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: 10683, dtype: object

[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:

The screenshot shows a Jupyter Notebook cell titled "Check the maximum number of Stops". The code block (cell [8]) uses the 'unique()' method on the 'Total_Stops' column to find all distinct values. The output is an array of strings: 'non-stop', '2 stops', '1 stop', '3 stops', 'nan', and '4 stops'.

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

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

Split the Values:

The screenshot shows a Jupyter Notebook cell titled "Split the Values". The code block (cell [9]) splits the 'Route' column into a list of lists, where each inner list represents a route in [Origin, Stop1, Stop2, Destination] format. The output is visible, showing a sample of the data structure.

```
[9] 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: 10683, dtype: object
```



```
[10] 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]

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

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

[13] data.Arrival_Time=data.Arrival_Time.str.split(' ')

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

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

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

[17] data.Duration=data.Duration.str.split(' ')

[18] 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]
```

Replace non-stop flights with 0 value:

```
Replace non-stop flights with 0 value

[19] 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]
```

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)
```

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

Check the null values:

{x} Check the null values

```
[ ] data.isnull().sum()
Airline      0
Date_of_Journey  0
Source       0
Destination  0
Route        1
Dep_Time     0
Arrival_Time 0
Duration     0
Total_Stops  1
Additional_Info 0
Price        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
```

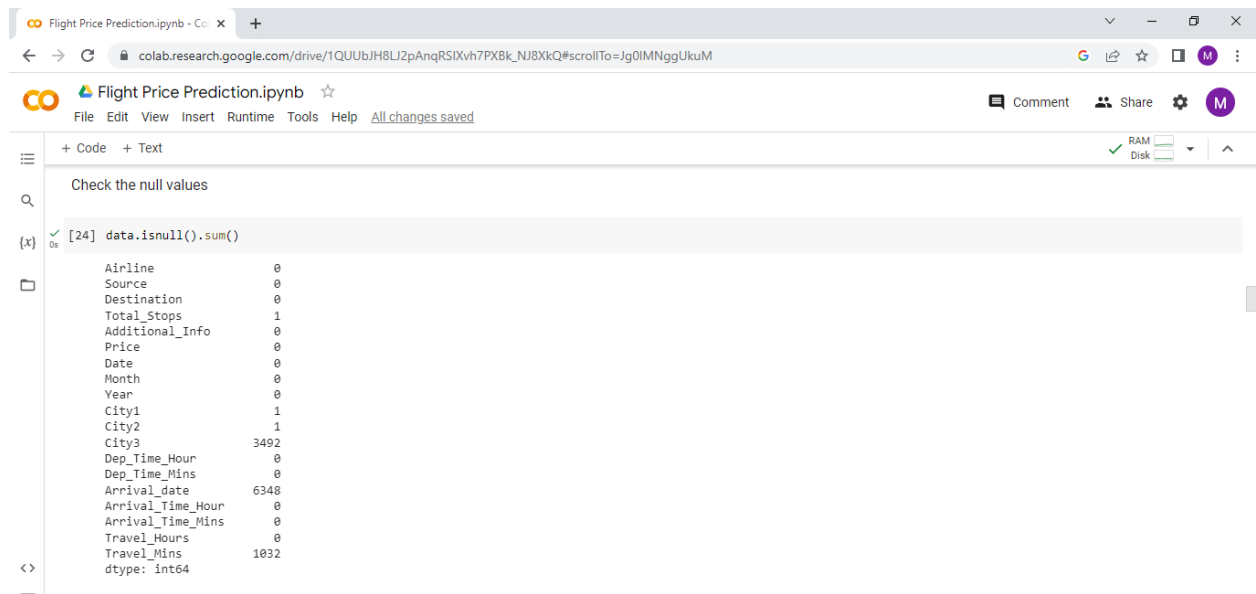
Drop the columns:

Drop the columns

```
<> [22] data.drop(['City4', 'City5', 'City6'], axis=1, inplace=True)
0s
```

```
[23] data.drop(['Date_of_Journey', 'Route', 'Dep_Time', 'Arrival_Time', 'Duration'], axis=1, inplace=True)
0s
data.drop(['Time_of_Arrival'], axis=1, inplace=True)
```

Checking null values:



The screenshot shows a Jupyter Notebook titled "Flight Price Prediction.ipynb" in a Google Colab environment. The code cell [24] contains the command `data.isnull().sum()`. The output is a series of 19 columns with their respective null counts. Most columns have 0 nulls, but `City3` has 3492 nulls, `Arrival_date` has 6348 nulls, and `Travel_Mins` has 1032 nulls. The data type for the series is `dtype: int64`.

```
[24] data.isnull().sum()

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
dtype: int64
```

Replacing missing values:



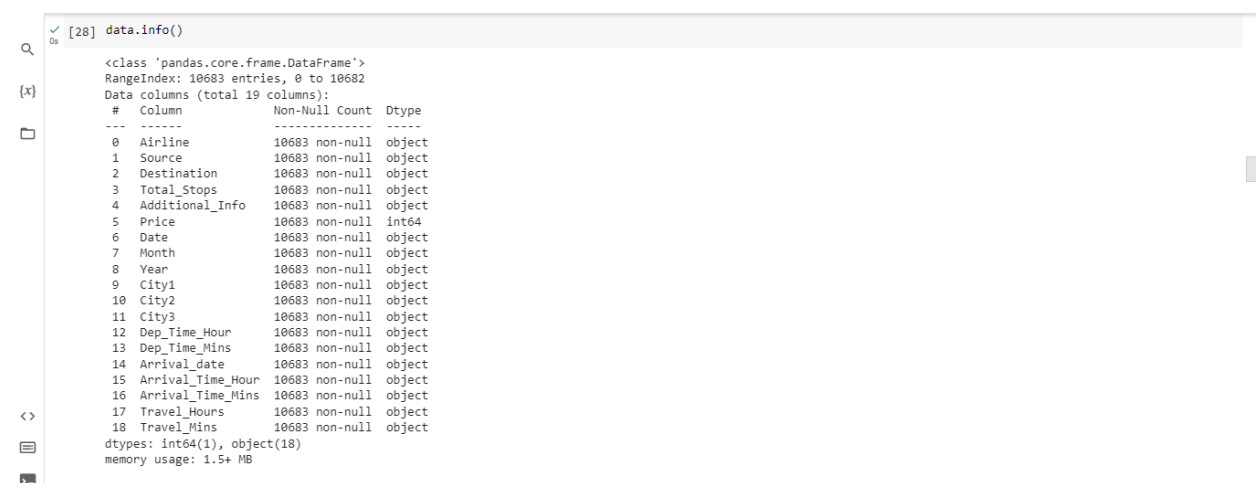
The screenshot shows a Jupyter Notebook titled "Replacing missing values". The code cell [25] contains commands to replace null values in `City1`, `City2`, `City3`, and `Total_Stops` with the string `'None'`. The code cell [26] replaces null values in `Arrival_date` with the value from the `Date` column. The code cell [27] replaces null values in `Travel_Mins` with the integer `0`.

```
[25] data['City1'].fillna('None',inplace=True)
data['City2'].fillna('None',inplace=True)
data['City3'].fillna('None',inplace=True)
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 titled "Check Info". The code cell [28] contains the command `data.info()`. The output provides a summary of the DataFrame, including the number of entries (10683), the number of columns (19), and the data types of each column. The data types are: `Airline` (object), `Source` (object), `Destination` (object), `Total_Stops` (object), `Additional_Info` (object), `Price` (int64), `Date` (object), `Month` (object), `Year` (object), `City1` (object), `City2` (object), `City3` (object), `Dep_Time_Hour` (object), `Dep_Time_Mins` (object), `Arrival_date` (object), `Arrival_Time_Hour` (object), `Arrival_Time_Mins` (object), `Travel_Hours` (object), and `Travel_Mins` (object). The memory usage is 1.5+ MB.

```
[28] data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10683 entries, 0 to 10682
Data columns (total 19 columns):
#   Column                Non-Null Count  Dtype
---  ---
0   Airline                10683 non-null  object
1   Source                 10683 non-null  object
2   Destination            10683 non-null  object
3   Total_Stops            10683 non-null  object
4   Additional_Info        10683 non-null  object
5   Price                  10683 non-null  int64
6   Date                   10683 non-null  object
7   Month                  10683 non-null  object
8   Year                   10683 non-null  object
9   City1                  10683 non-null  object
10  City2                  10683 non-null  object
11  City3                  10683 non-null  object
12  Dep_Time_Hour          10683 non-null  object
13  Dep_Time_Mins          10683 non-null  object
14  Arrival_date           10683 non-null  object
15  Arrival_Time_Hour      10683 non-null  object
16  Arrival_Time_Mins      10683 non-null  object
17  Travel_Hours           10683 non-null  object
18  Travel_Mins            10683 non-null  object
dtypes: int64(1), object(18)
memory usage: 1.5+ MB
```

Change the Datatype:

The screenshot shows a Google Colab notebook interface. The top bar indicates the notebook is titled "Flight Price Prediction.ipynb" and is located at a specific Google Drive link. The notebook has tabs for "Code" and "Text", and a status bar showing "RAM" and "Disk" usage.

The code cell contains the following Python code:

```
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_Hour.astype('int64')
data.Travel_Mins=data.Travel_Mins.astype('int64')
```

Below the code, a data table is displayed, showing the result of the code execution. The table has 14 columns: "Additional_Info", "Price", "Date", "Month", "Year", "City1", "City2", "City3", "Dep_Time_Hour", "Dep_Time_Mins", "Arrival_date", "Arrival_Time_Hour", "Arrival_Time_Mins", "Travel_Hours", and "Travel_Mins". The first row of data is:

Additional_Info	Price	Date	Month	Year	City1	City2	City3	Dep_Time_Hour	Dep_Time_Mins	Arrival_date	Arrival_Time_Hour	Arrival_Time_Mins	Travel_Hours	Travel_Mins
No info	17327	6	3	2019	BOM	GOI	PNQ	16	50	6	16	16	5m	0

The bottom part of the screenshot shows the next code cell, which contains the following Python code:

```
[31] 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']
```

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:

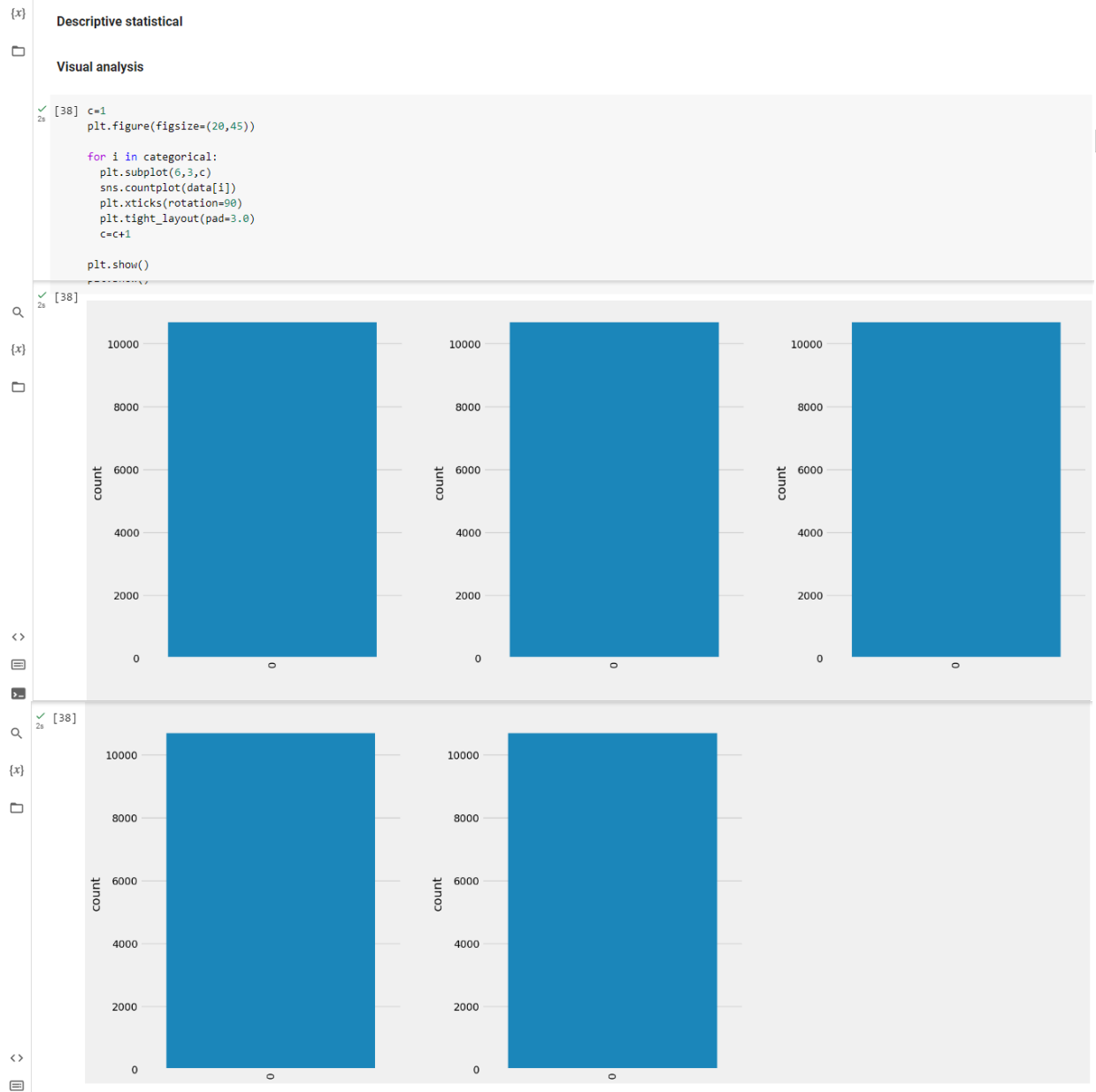
Output Columns

```
[37] data.head()
```

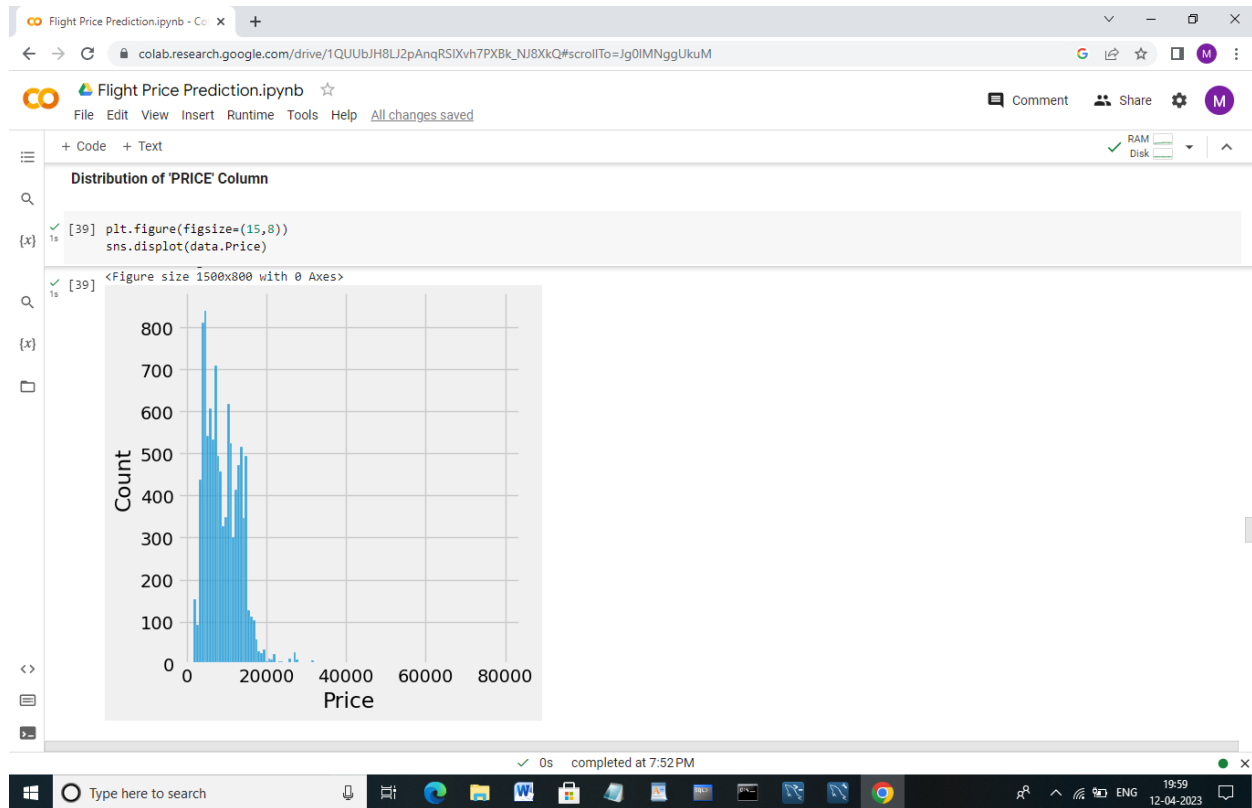
	Airline	Source	Destination	Total_Stops	Additional_Info	Price	Date	Month	Year	City1	City2	City3	Dep_Time_Hour	Dep_Time_Mins	Arrival_date	Arrival_T
0	3	0	5	5	8	3897	24	3	2019	0	13	29	22	20	22	
1	1	3	0	1	8	7662	1	5	2019	2	25	1	5	50	1	
2	4	2	1	1	8	13882	9	6	2019	3	32	4	9	25	10	
3	3	3	0	0	8	6218	12	5	2019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

Flight Price Prediction.ipynb - Co

colab.research.google.com/drive/1QUUbJH8LJ2pAnqRSIXvh7PXBk_NJ8XkQ#scrollTo=Jg0IMNggUkuM

Flight Price Prediction.ipynb

File Edit View Insert Runtime Tools Help All changes saved

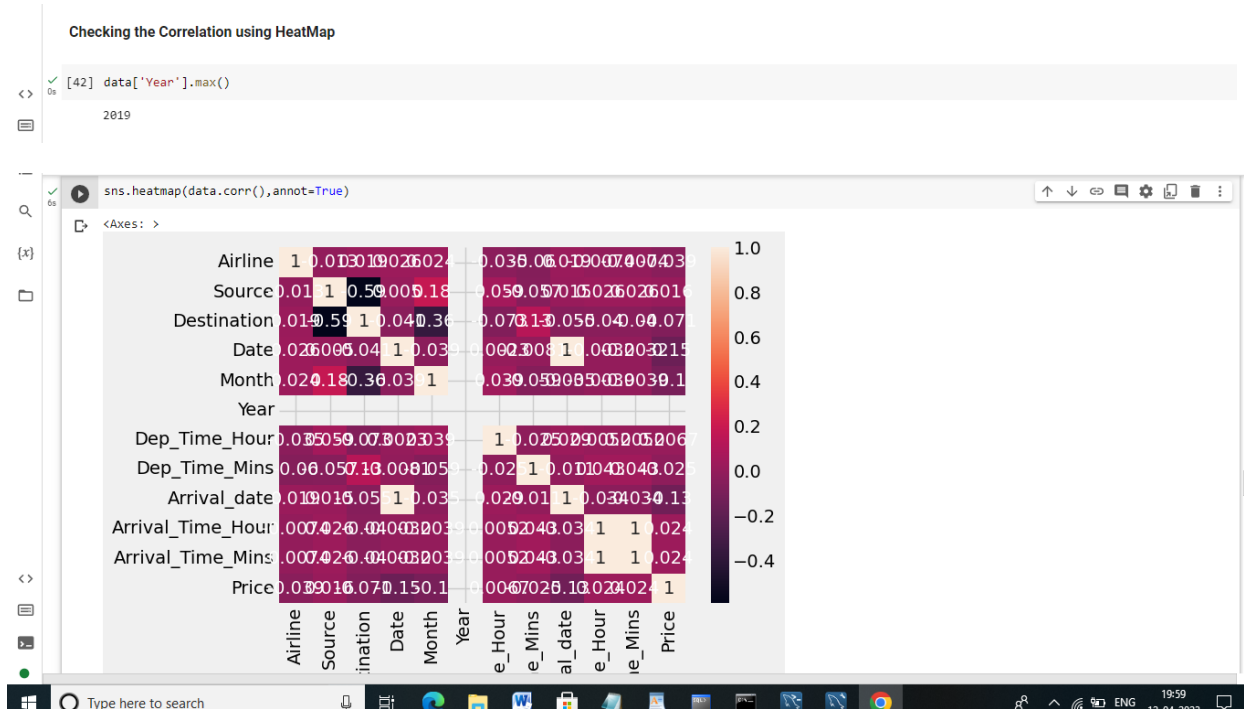
+ Code + Text

output column

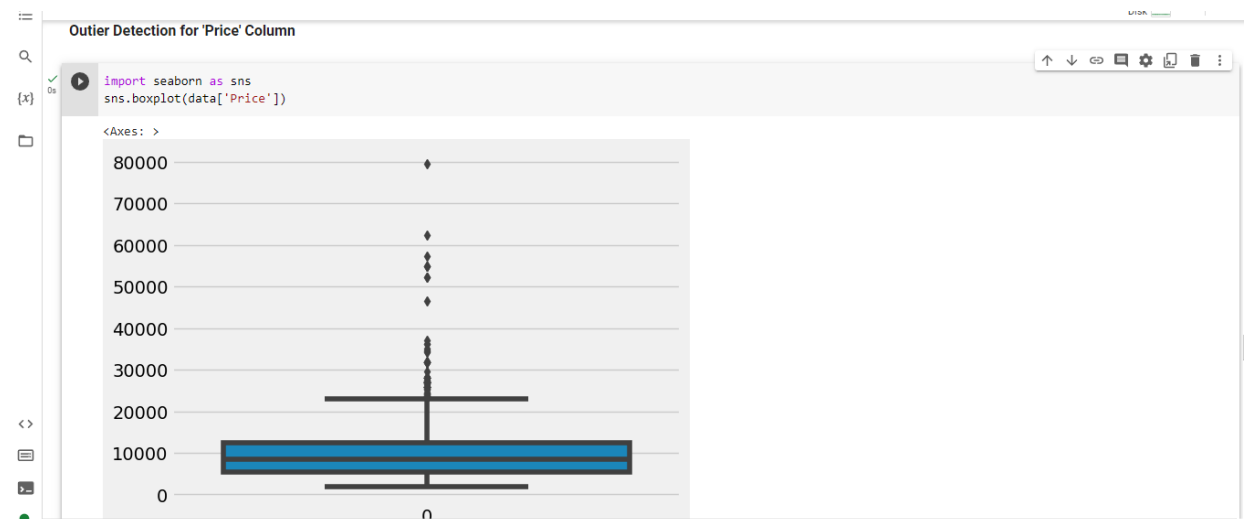
```
[40] data=data[['Airline','Source','Destination','Date','Month','Year','Dep_Time_Hour','Dep_Time_Mins','Arrival_date','Arrival_Time_Hour','Arrival_Time_Mins','Price']]
[41] data.head()
```

	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

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)
```

```
[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)
```

```
[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 1227.442866923497
Mean Squared Error is 3356157.945718379
Root Mean Squared Error is 1831.9819719960071
GradientBoostingRegressor()
R2 score is 0.7659806545178971
R2 for train data 0.738149720400205
Mean Absolute Error is 1687.5222333706563
Mean Squared Error is 4946940.077210986
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 0.9226094111129454
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 x 1 columns

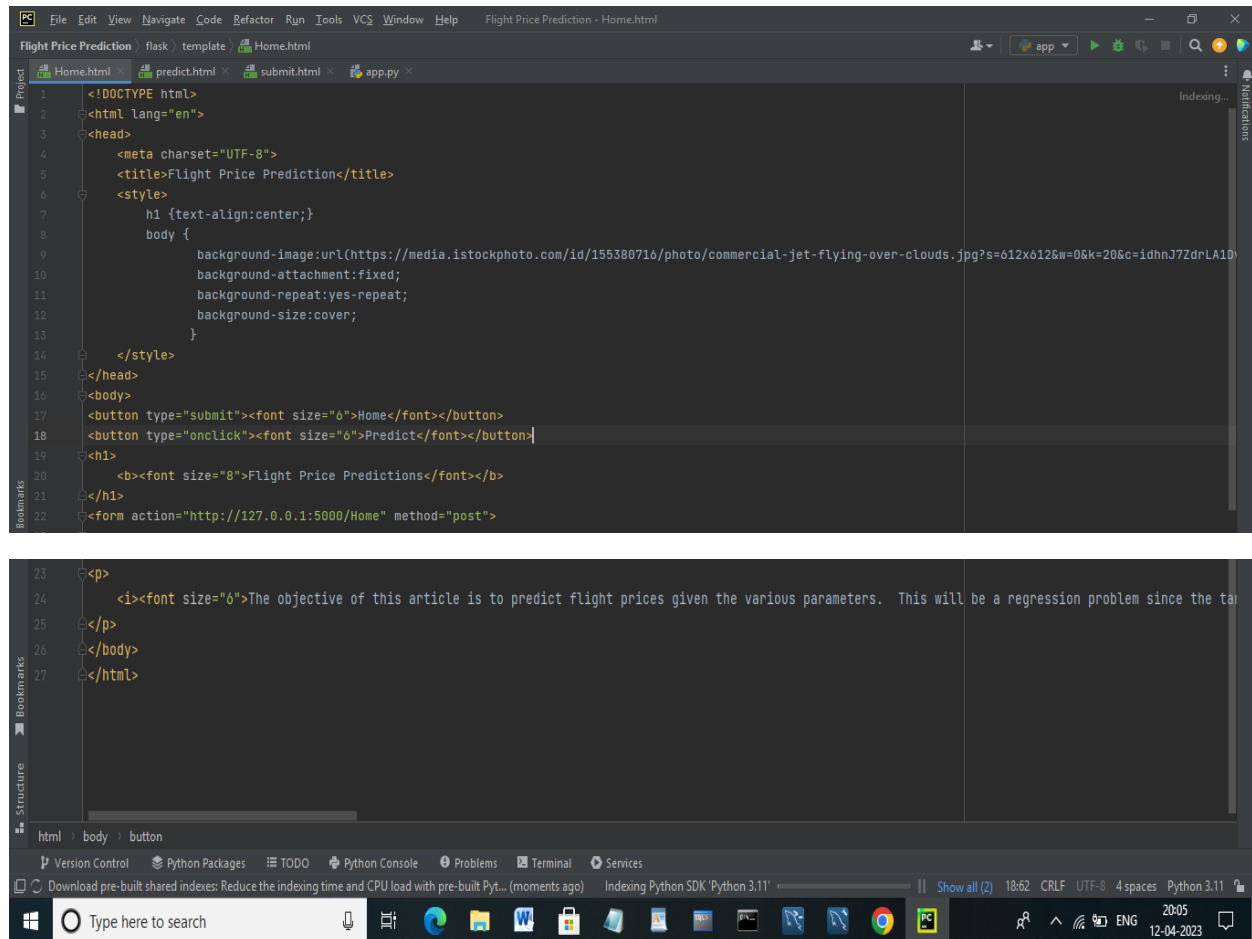
import pickle
pickle.dump(rfr,open('model1.pk1','wb'))
```

Save the best model:

```
Save the best model

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

Home.html:



```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4 <meta charset="UTF-8">
5 <title>Flight Price Prediction</title>
6 <style>
7   h1 {text-align:center;}
8   body {
9     background-image:url(https://media.istockphoto.com/id/155380716/photo/commercial-jet-flying-over-clouds.jpg?s=612x612&w=0&k=20&c=idhnJ7ZdrLA1D);
10    background-attachment:fixed;
11    background-repeat:yes-repeat;
12    background-size:cover;
13  }
14 </style>
15 </head>
16 <body>
17 <button type="submit"><font size="6">Home</font></button>
18 <button type="onClick"><font size="6">Predict</font></button>
19 <h1>
20 <b><font size="8">Flight Price Predictions</font></b>
21 </h1>
22 <form action="http://127.0.0.1:5000/Home" method="post">
23 <p>
24 <i><font size="6">The objective of this article is to predict flight prices given the various parameters. This will be a regression problem since the ta
25 </i></p>
26 </body>
27 </html>
```

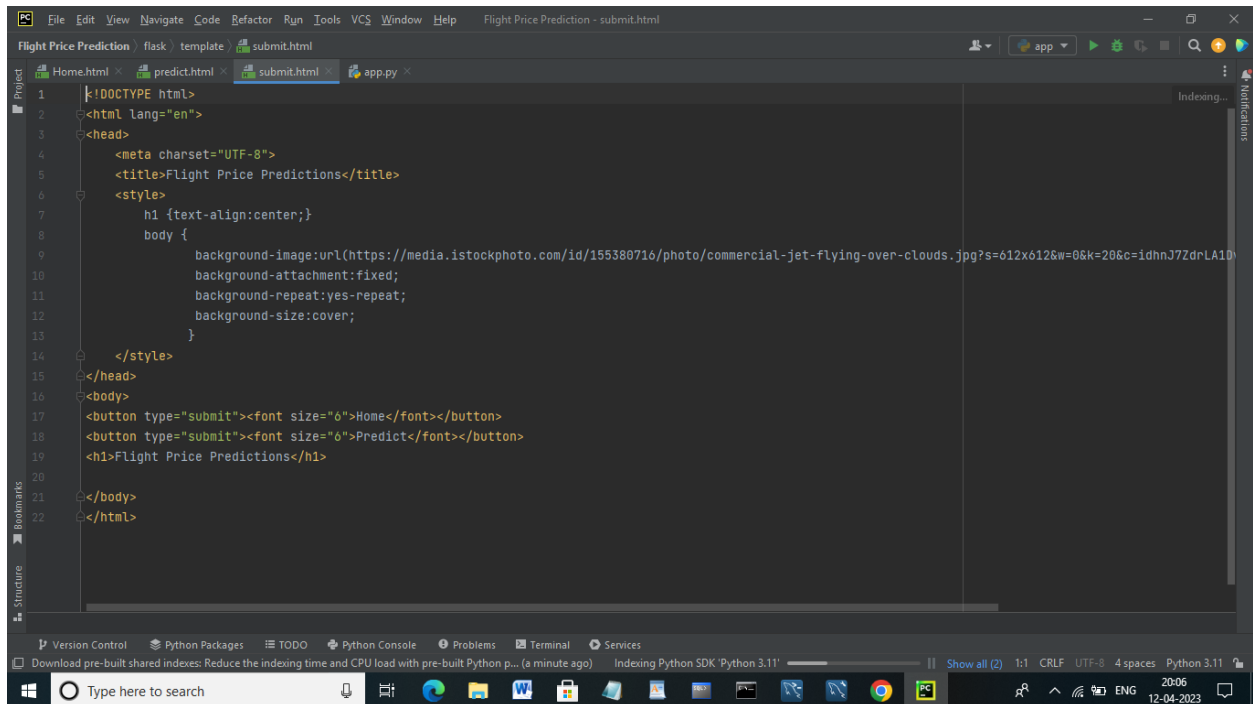
predict.html:

```
PC File Edit View Navigate Code Refactor Run Tools VCS Window Help Flight Price Prediction - predict.html
Flight Price Prediction flask template predict.html
Project Home.html predict.html submit.html app.py
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4 <meta charset="UTF-8">
5 <title>Flight Price Prediction</title>
6 <style>
7     body {
8         background-image:url(https://media.istockphoto.com/id/155380716/photo/commercial-jet-flying-over-clouds.jpg?s=612x612w=0&k=20&c=idhnJ7ZdrLA1D);
9         background-attachment:fixed;
10        background-repeat:yes-repeat;
11        background-size:cover;
12    }
13 </style>
14
15 </head>
16 <body>
17 <h1><font size="6">Flight Price prediction</font></h1>
18 <form action="http://127.0.0.1:5000/predict" method="post">
19 <h4>Airline</h4>
20 <select name="airline" id="airline" required="required">
21     <option value="Jet Airways">Jet Airways</option>
22     <option value="IndiGo">IndiGo</option>
23     <option value="Air India">Air India</option>
24     <option value="Multiple carriers">Multiple carriers</option>
25     <option value="SpiceJet">SpiceJet</option>
26     <option value="Vistara">Vistara</option>
```

```
27     <option value="Air Asia">Air Asia</option>
28     <option value="GoAir">GoAir</option>
29     <option value="Multiple carriers Premium economy">Multiple carriers Premium economy
30     </option>
31     <option value="Jet Airways Business">Jet Airways Business</option>
32     <option value="Vistara Premium economy">Vistara Premium economy</option>
33     <option value="Trujet">Trujet</option>
34 </select>
35 <h4>Source</h4>
36 <select name="Source" id="Source" required="required">
37     <option value="Delhi">Delhi</option>
38     <option value="Kolkata">Kolkata</option>
39     <option value="Mumbai">Mumbai</option>
40     <option value="Chennai">Chennai</option>
41 </select>
42 <h4>Destination</h4>
43 <select name="Destination" id="Destination" required="required">
44     <option value="Cochin">Cochin</option>
45     <option value="Delhi">Delhi</option>
46     <option value="New Delhi">New Delhi</option>
47     <option value="Hyderabad">Hyderabad</option>
48     <option value="Kolkata">Kolkata</option>
49 </select>
50 <h4>depdate</h4>
51 <input type="text" id="Date" name="Date">
```

```
52 <h4>depmonth</h4>
53 <input type="text" id="Month" name="Month">
54 <h4>depyear</h4>
55 <input type="text" id="Year" name="Year">
56 <h4>deptimehour</h4>
57 <input type="text" id="Dep_Time_Hour" name="Dep_Time_Hour">
58 <h4>deptimemins</h4>
59 <input type="text" id="Dep_Time_Mins" name="Dep_Time_Mins">
60 <h4>artime</h4>
61 <input type="text" id="Arrival_Date" name="Arrival_Date">
62 <h4>artimehour</h4>
63 <input type="text" id="Arrival_Time_Hour" name="Arrival_Time_Hour">
64 <h4>artimemins</h4>
65 <input type="text" id="Arrival_Time_Mins" name="Arrival_Time_Mins">
66 <br>
67 <br>
68 <input type="submit" value="Submit" class="btn btn-secondary">
69 </form>
70 </body>
71 </html>
```

submit.html:

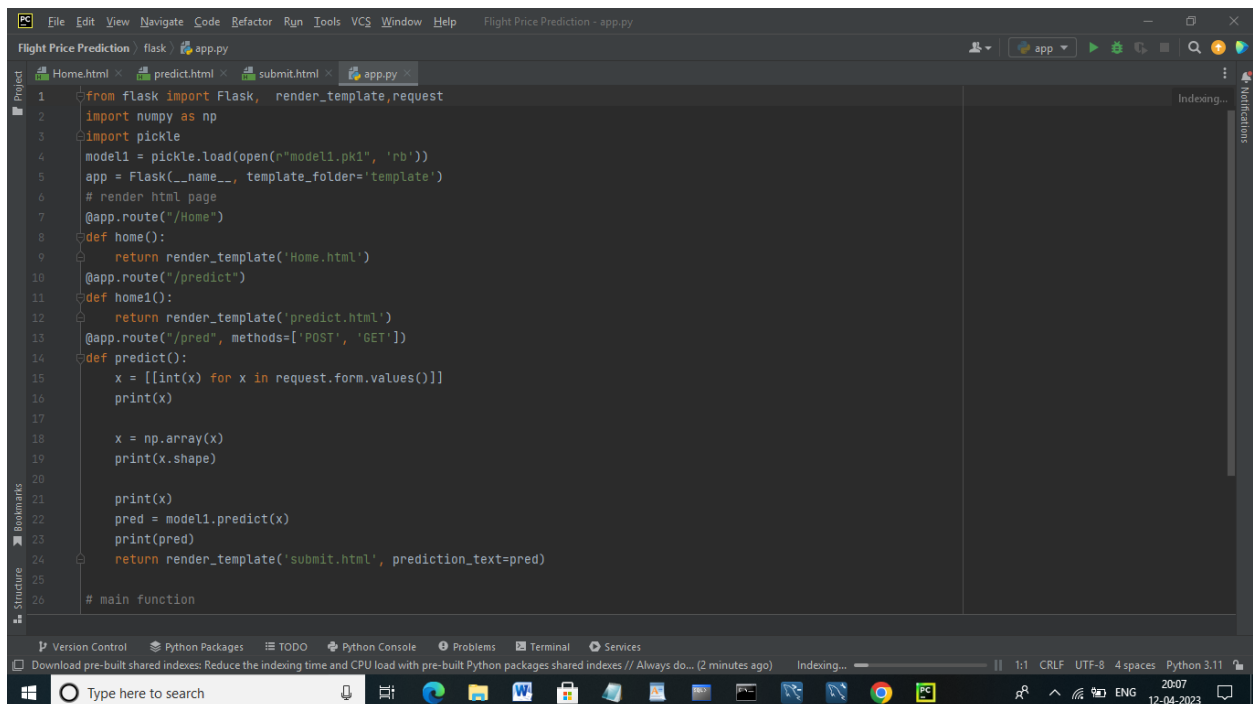


The screenshot shows a Visual Studio Code editor window with the file 'submit.html' open. The editor has a dark theme. The file content is as follows:

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>Flight Price Predictions</title>
6   <style>
7     h1 {text-align:center;}
8     body {
9       background-image:url(https://media.istockphoto.com/id/155380716/photo/commercial-jet-flying-over-clouds.jpg?s=612x612&w=0&k=20&c=idhnJ7ZdrLA10);
10      background-attachment:fixed;
11      background-repeat:yes-repeat;
12      background-size:cover;
13    }
14  </style>
15 </head>
16 <body>
17   <button type="submit"><font size="6">Home</font></button>
18   <button type="submit"><font size="6">Predict</font></button>
19   <h1>Flight Price Predictions</h1>
20
21 </body>
22 </html>
```

The bottom status bar shows 'Python 3.11' and '20:06 12-04-2023'.

app.py:



The screenshot shows a Visual Studio Code editor window with the file 'app.py' open. The editor has a dark theme. The file content is as follows:

```
1 from flask import Flask, render_template, request
2 import numpy as np
3 import pickle
4 model1 = pickle.load(open("model1.pkl", 'rb'))
5 app = Flask(__name__, template_folder='template')
6 # render html page
7 @app.route("/Home")
8 def home():
9     return render_template('Home.html')
10 @app.route("/predict")
11 def home1():
12     return render_template('predict.html')
13 @app.route("/pred", methods=['POST', 'GET'])
14 def predict():
15     x = [[int(x) for x in request.form.values()]]
16     print(x)
17
18     x = np.array(x)
19     print(x.shape)
20
21     print(x)
22     pred = model1.predict(x)
23     print(pred)
24     return render_template('submit.html', prediction_text=pred)
25
26 # main function
```

The bottom status bar shows 'Python 3.11' and '20:07 12-04-2023'.

```
Flight Price Prediction - app.py
File Edit View Navigate Code Refactor Run Tools VCS Window Help

Flight Price Prediction flask app.py
Project Home.html predict.html submit.html app.py
model1 = pickle.load(open("model1.pkl", 'rb'))
app = Flask(__name__, template_folder='template')
# render html page
@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 = model1.predict(x)
    print(pred)
    return render_template('submit.html', prediction_text=pred)

# main function
if __name__ == "__main__":
    app.run(debug=False)

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)

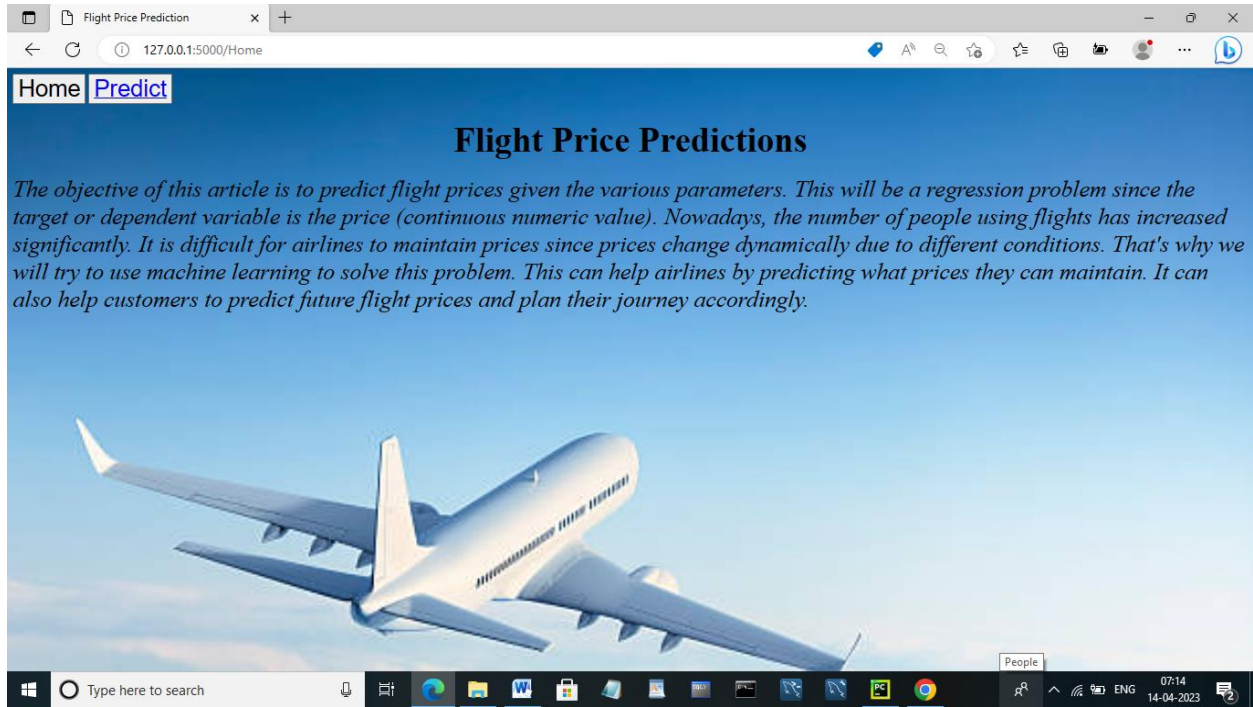
    print(x)
    pred = model1.predict(x)
    print(pred)
    return render_template('submit.html', prediction_text=pred)

Run: app
C:\Users\ELCOT\AppData\Local\Programs\Python\Python311\python.exe "D:\Flight Price Prediction\Flask\app.py"
* Serving Flask app 'app'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit

Version Control Run Python Packages TODO Python Console Problems Terminal Services
Download pre-built shared indexes: Reduce the indexing time and CPU load with pre-built Python packages shared indexes // Always download // Download once // Don't show ag... (3 minutes ago)
1:1 CRLF UTF-8 4 spaces Python 3.11
Type here to search 20:08 12-04-2023
```


OUTPUT

Home Page:



Predict page:

Flight Price Prediction

127.0.0.1:5000/predict

Flight Price prediction

Airline
Air Asia

Source
Chennai

Destination
Delhi

depdate
13

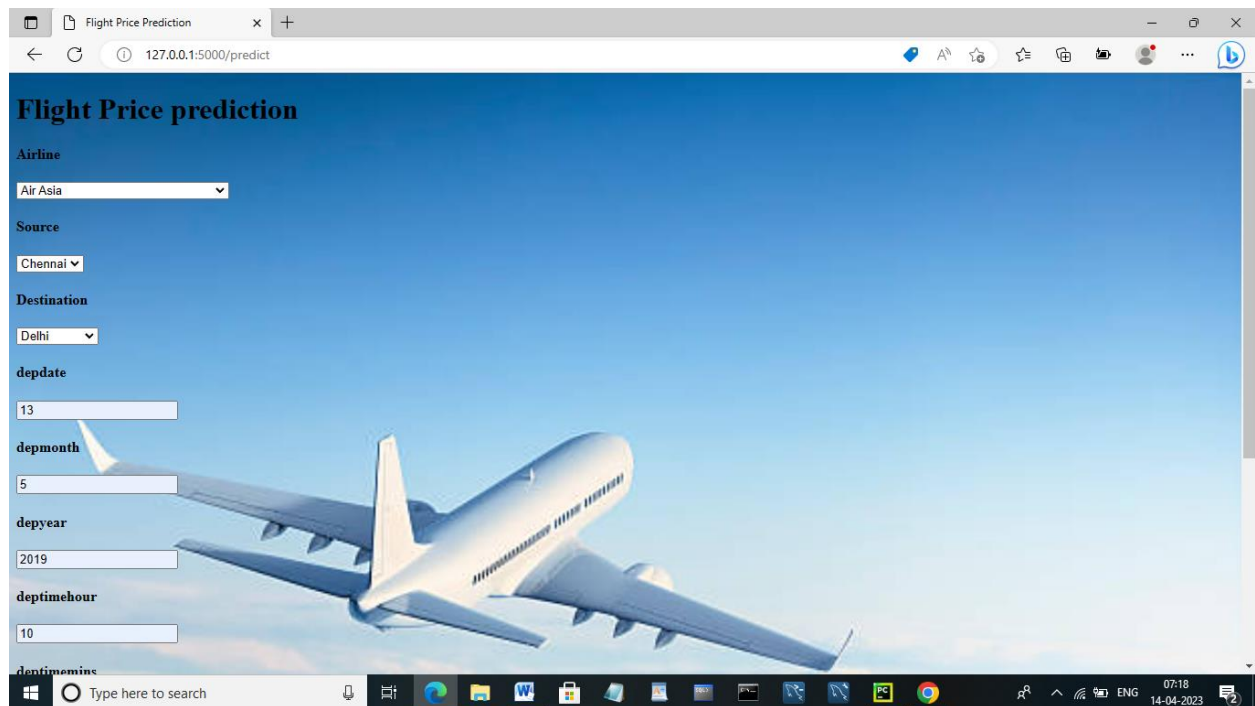
depmonth
5

depyear
2019

deptimehour
10

deptimemins

Type here to search



Flight Price Prediction

127.0.0.1:5000/predict

13

depmonth
5

depyear
2019

deptimehour
10

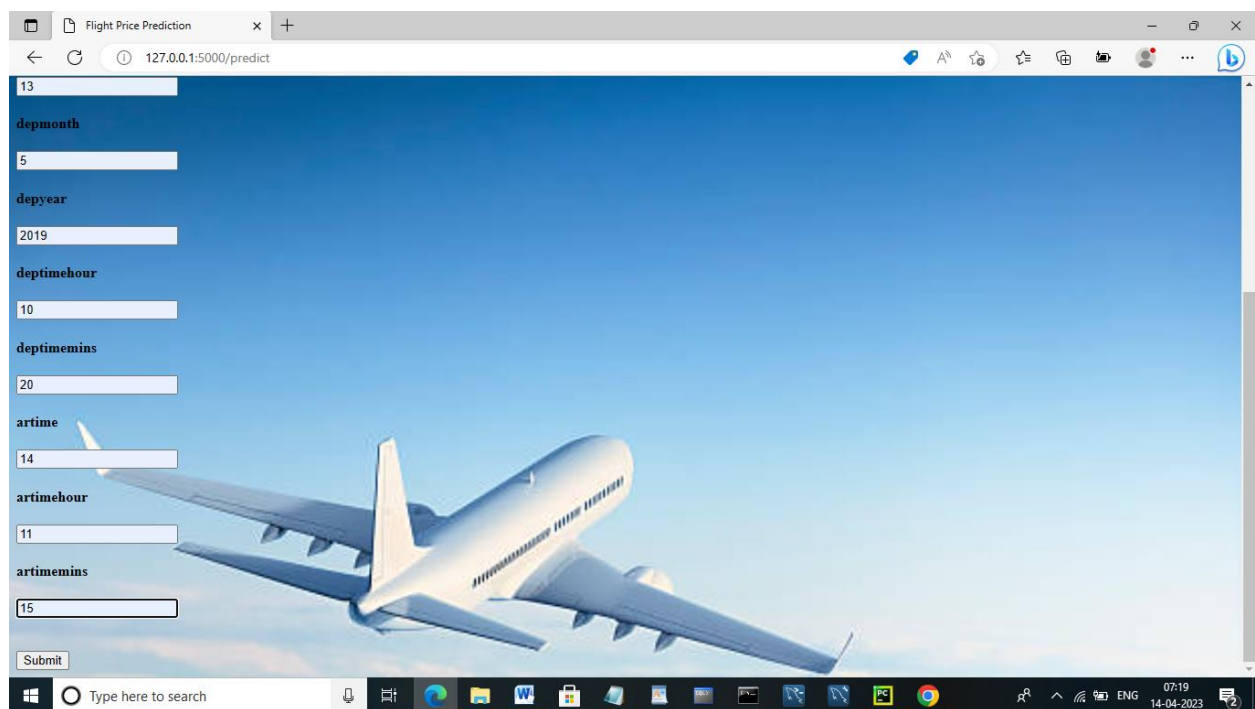
deptimemins
20

artime
14

artimehour
11

artimemins
15

Submit



Summit Page (Final Output):

