OPTIMIZING FLIGHT BOOKING DECISIONS THROUGH MACHINE LEARNING PRICE PREDICTIONS

PROJECT BASED EXPERIENTIAL LEARNING PROGRAM

Optimizing Flight Booking Decisions through Machine Learning Price Predictions

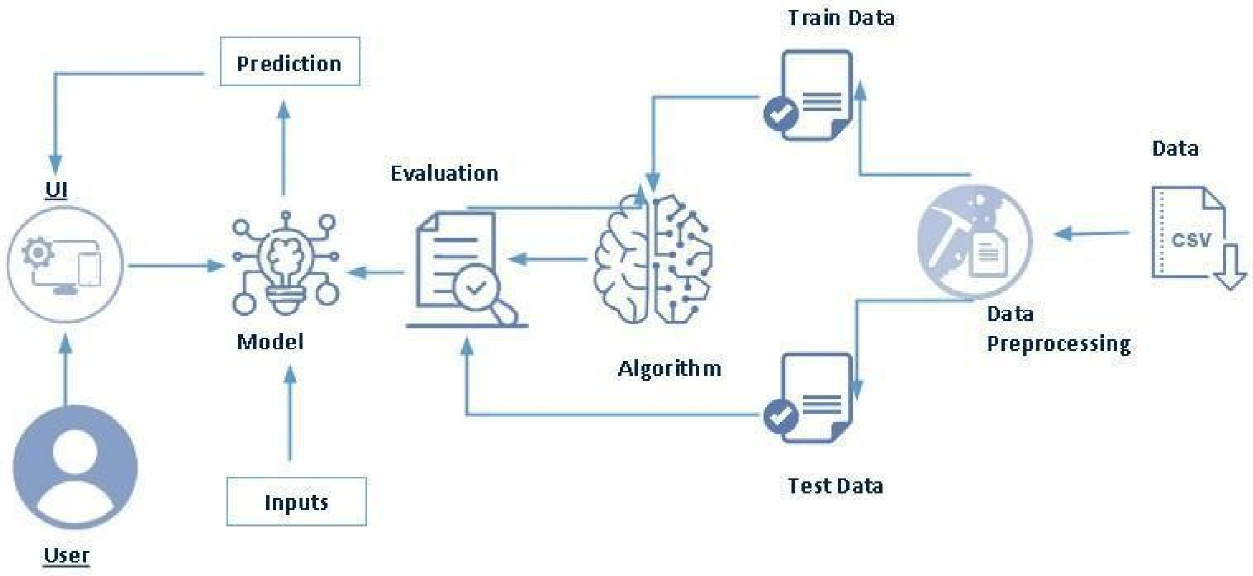
1. INTRODUCTION :

1.1 OVERVIEW

In this project, we will be analyzing the flight fare prediction using Machine Learning dataset using essential exploratory data analysis techniques then will draw some predictions about the price of the flight based on some features such as what type of airline it is, what is the arrival time, what is the departure time, what is the duration of the flight, source, destination and more.

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

<u>Technical Architecture:</u>



Project Flow:

- User interacts with the UI to enter the input.
- Entered input is analysed by the model which is integrated.
- Once model analyses the input the prediction is showcased on the UI

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

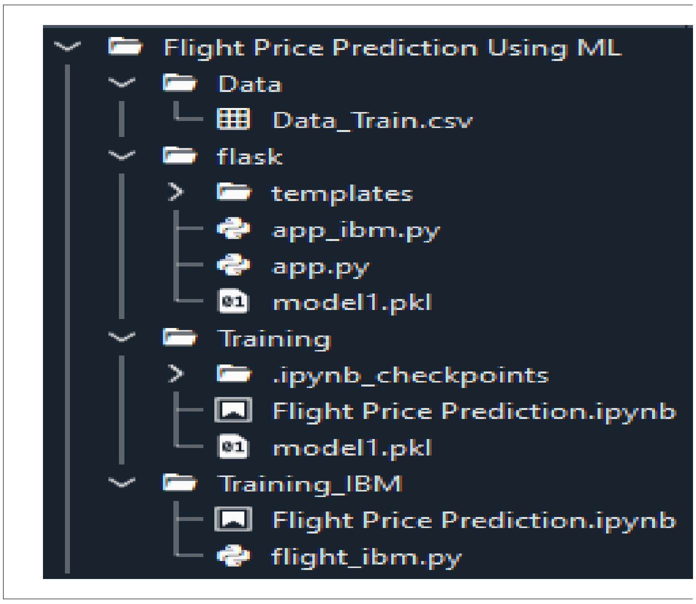
- Define Problem / Problem Understanding

- Specify the business problem
- Business requirements
- Literature Survey
- Social or Business Impact.
- Data Collection & Preparation
- Collect the dataset
- Data Preparation
- Exploratory Data Analysis
- Descriptive statistical
- Visual Analysis
- Model Building

- Training the model in multiple algorithms
- Testing the model
- Performance Testing & Hyperparameter Tuning
- Testing model with multiple evaluation metrics
- Comparing model accuracy before & after applying hyperparameter tuning
- Model Deployment
- Save the best model
- Integrate with Web Framework
- Project Demonstration & Documentation
- Record explanation Video for project end to end solution
- Project Documentation-Step by step project development procedure

PROJECT STRUCTURE

Create the Project folder which contains files as shown below



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

Milestone 1: Define Problem / Problem Understanding

Activity 1: Specify the business problem

Refer Project Description

Activity 2: Business Requirements

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

Activity 3: Literature Survey (Student Will Write)

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

Activity 4: Social or Business Impact. Social Impact: -

Personal loans can stimulate economic growth by providing individuals with the funds they need to make major purchases, start businesses, or invest in their education. Business Model/Impact: - Personal loan providers may charge fees for services such as loan origination, processing, and late payments. Advertising the brand awareness and marketing to reach out to potential borrowers to generate revenue.

Milestone 2: Data Collection & Preparation

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

Activity 1: Collect the dataset

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc. In this project we have used .csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.
Link: https://www.kaggle.com/code/anshigupta01/flight-price-prediction/data
As the dataset is downloaded. Let us read and understand the data properly with the help of some visualisation techniques and some analysing techniques.
Note: There are a number of techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

Activity 1.1: Importing the libraries

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

```python
#importing librares
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.metrics import f1_score,confusion_matrix,classification_report
from scipy import stats
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import GradientBoostingRegressor,RandomForestReg
ressor
from sklearn.model_selection import train_test_split
import pickle
import warnings
warnings.filterwarnings('ignore')
```

Activity 1.2: Read the Dataset

Our dataset format might be in .csv, excel files,.txt,.json, etc. We can read the dataset with the help of pandas. In pandas we have a function called read_csv() to read the dataset. As a parameter we have to give the directory of csv file.

```python
data=pd.read_excel('/content/FBPP.xlsx')
data.head()
```

dex	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	389
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	2 stops	No info	766
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL → LKO →	09:25	04:25 10 Jun	19h	2 stops	No info	1388

dex	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price
					BOM → COK						
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

Show 102550100 per page

Data Preparation

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

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

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

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

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

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

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

```
Airline ['IndiGo' 'Air India' 'Jet Airways' 'SpiceJet' 'Multiple carriers' 'GoAir'
 'Vistara' 'Air Asia' 'Vistara Premium economy' 'Jet Airways Business'
 'Multiple carriers Premium economy' 'Trujet']
Source ['Banglore' 'Kolkata' 'Delhi' 'Chennai' 'Mumbai']
Destination ['New Delhi' 'Banglore' 'Cochin' 'Kolkata' 'Delhi' 'Hyderabad']
Additional_Info ['No info' 'In-flight meal not included' 'No check-in baggage included'
 '1 Short layover' 'No Info' '1 Long layover' 'Change airports'
 'Business class' 'Red-eye flight' '2 Long layover']
```

1. Airline column has 12 unique values - 'IndiGo' , 'Air India', 'Jet Airways' , 'SpiceJet' , 'Multiple carriers' , 'GoAir', 'Vistara', 'Air Asia', 'Vistara Premium economy' , 'Jet Airways Business', 'Multiple carriers Premium economy', 'Trujet'.

2. Source column has 5 unique values – 'Bangalore', 'Kolkata', 'Chennai', 'Delhi' and 'Mumbai'.

3. Destination column has 6 unique values - 'New Delhi', 'Banglore', 'Cochin', 'Kolkata', 'Delhi' , 'Hyderabad'.

4. Additional info column has 10 unique values - 'No info', 'In-flight meal not included', 'No check-in baggage included', '1 Short layover' , 'No Info', '1 Long layover', 'Change airports' , 'Business class', 'Red-eye flight' , '2 Long layover'.

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

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

```
data.dropna(inplace=True)
Datedata._of_Journey=data.Date_of_Journey.str.split('/')
data.Date_of_Journey
```

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

```
#Treating the data_column
data['Date']=data.Date_of_Journey.str[0]
data['Month']=data.Date_of_Journey.str[1]
data['Year']=data.Date_of_Journey.str[2]
```

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

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

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

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

```
data.Route=data.Route.str.split('→')
```

```
data.Route
```

```
0          [BLR , DEL]
1    [CCU , IXR , BBI , BLR]
2    [DEL , LKO , BOM , COK]
```

```
3         [CCU , NAG , BLR]
4         [BLR , NAG , DEL]
     ...
10678          [CCU , BLR]
10679          [CCU , BLR]
10680          [BLR , DEL]
10681          [BLR , DEL]
10682 [DEL , GOI , BOM , COK]
Name: Route, Length: 10682, dtype: object
```

```
data['city1']=data.Route.str[0]
data['city2']=data.Route.str[1]
data['city3']=data.Route.str[2]
data['city4']=data.Route.str[3]
data['city5']=data.Route.str[4]
data['city6']=data.Route.str[5]
```

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

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

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

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

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

```
data.Arrival_Time=data.Arrival_Time.str.split(':')
data.Arrival_Time_Mins=data.Arrival_Time_Mins.str.split(' ')
data['Arrival_Time_Mins']=data.Arrival_Time_Mins.str[0]
data['Arrival_Day']=data.Arrival_Time_Mins.str[1]
data.Arrival_Time_Mins=data.Arrival_Time_Mins.str.split(' ')
data['Arrival_Time_Mins']=data.Arrival_Time_Mins.str[0]
data['Arrival_Day']=data.Arrival_Time_Mins.str[1]
```

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

```
#Next, we divide the ‘Duration’ column to ‘Travel_hours’ and ‘ Travel_mins’
data.Duration=data.Duration.str.split(' ')

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

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

#we also treat the ‘Total _stops’ column replace non-stop flights with 0 value
and extract the integer part of the ‘Total_stops’

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

•We also treat the ‘Total_stops’ column, and replace non-stop flights with 0 value and extract the integer part of the ‘Total_Stops’ column.

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

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

```
0      0
1      2
2      2
3      1
4      1
       ..
10678  0
10679  0
```

We proceed further to the ‘Additional_info’ column, where we observe that there are 2 categories signifying ‘No info’, which are divided into 2 categories since ‘I’ in ‘No Info’ is capital. We replace ‘No Info’ by ‘No info’ to merge it into a single category.

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

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

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

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

```
10680  0
10681  0
10682  2
```

Name: Total_Stops, Length: 10682, dtype: object

-

```
data.isnull().sum()
```

```
Airline                0
Date_of_Journey        0
Source                 0
Destination            0
Route                  0
Dep_Time               0
Arrival_Time           0
Duration               0
Total_Stops            0
Additional_Info        0
Price                  0
Date                   0
Month                  0
Year                   0
city1                  0
city2                  0
city3                  3491
city4                  9116
city5                  10636
city6                  10681
Dep_Time_Hours         0
Dep_Time_Mins          0
Arrival_Time_Hour      0
Arrival_Time_Mins      0
Arrival_Day            0
Travel_Hours           0
Travel_Mins            1032
dtype: int64
```

```
data.drop(['city4','city5','city6'],axis=1,inplace=True)
```

-

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

```
data.isnull().sum()
Airline              0
Source               0
 Destination         0
Total_Stops          0
 Additional_Info     0
 Price               0
Date                 0
 Month               0
 Year                0
 city1               0
city2                0
 city3 3491
Dep_Time_Hours       0
 Dep_Time_Mins       0
 Arrival_Time_Hour   0
 Arrival_Time_Mins   0
Arrival_Day          0
Travel_Hours         0
Travel_Mins          1032
dtype: int64
```

Activity 2.1: Replacing Missing Values

We further replace 'NaN' values in 'City3' with 'None', since rows where 'City3' is missing did not have any stop, just the source and the destination. We also replace missing values in 'Arrival_date' column with values in 'Date' column, since the missing values are those values where the flight took off and landed on the same date. We also replace missing values in 'Travel_mins' as 0, since the missing values represent that the travel time was in terms on hours only, and no additional minutes.

```
#filling City3 as name , the missing values are less
data['city3'].fillna('None',inplace=True)
```

#filling Arrival_Date as Departure_Date

```
data['Arrival_Day'].fillna(data['Date'],inplace=True)
```

#filling Travel_Mins as Zero(0)

```
data['Travel_Mins'].fillna(0,inplace=True)
```

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

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
 Int64Index: 10682 entries, 0 to 10682
Data columns (total 19 columns):
 #   Column          Non-Null     Count        Dtype
---  ------          -----        --------     ----
 0   Airline          10682       non-null     object
 1   Source           10682       non-null     object
2    Destination      10682       non-null     object
3    Total_Stops      10682       non-null     object
4    Additional_Info  10682       non-null     object
5    Price            10682       non-null     int64
6    Date             10682       non-null     object
7    Month            10682       non-null     object
8    Year             10682       non-null     object
9    city1            10682       non-null     object
10   city2            10682       non-null     object
11   city3            10682       non-null     object
12   Dep_Time_Hours   10682       non-null     object
13   Dep_Time_Mins    10682       non-null     object
14   Arrival_Time_Hour 10682      non-null     object
15   Arrival_Time_Mins 10682      non-null     object
16   Arrival_Day      10682       non-null     object
17   Travel_Hours     10682       non-null     object
 18  Travel_Mins      10682       non-null     object
```

dtypes: int64(1), object(18) memory usage: 1.6+ MB

Hence, we try to change the data type of the required columns

```
data.Total_Stops=data.Total_Stops.astype('int64')
data.Date=data.Date.astype('int64')
data.Month=data.Month.astype('int64')
data.Year=data.Year.astype('int64')
data.Dep_Time_Hours=data.Dep_Time_Hours.astype('int64')
data.Dep_Time_Mins=data.Dep_Time_Mins.astype('int64')
data.Arrival_Time_Hour=data.Arrival_Time_Hour.astype('int64')
data.Travel_Mins=data.Travel_Mins.astype('int64')
```

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

ndex	Airline	Source	Destination	Total_Stops	Additional_Info	Price	Date	Month	Year	city1	city2	city3	Dep_Time_Hours
6474	Air India	Mumbai	Hyderabad	2	No info	17327	6	3	2019	BOM	GOI	PNQ	16

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

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

We then convert the 'Travel_hours' column to 'int' data type, and the operation happens successfully. We now have a treated dataset with 10682 rows and 17 columns (16 independent and 1 dependent variable). We create separate lists of categorical columns and numerical columns for plotting and analyzing the data

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

```
#creating list of different types of columns
```

```
categorical = data[column]
categorical
```

index	Airline	Source	Destination	Additional_Info	city1	city2	city3	Arrival_Time_Mins	Arrival_Day
0	IndiGo	Banglore	New Delhi	No info	BLR	DEL	None	10	0
1	Air India	Kolkata	Banglore	No info	CCU	IXR	BBI	15	5
2	Jet Airways	Delhi	Cochin	No info	DEL	LKO	BOM	25	5
3	IndiGo	Kolkata	Banglore	No info	CCU	NAG	BLR	30	0
4	IndiGo	Banglore	New Delhi	No info	BLR	NAG	DEL	35	5
5	SpiceJet	Kolkata	Banglore	No info	CCU	BLR	None	25	5
6	Jet Airways	Banglore	New Delhi	In-flight meal not included	BLR	BOM	DEL	25	5

Activity 2.2: Label Encoding

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

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()

data.airline=le.fit_transform(data.Airline)
data.Source=le.fit_transform(data.Source)
data.Destination=le.fit_transform(data.Destination)
data.Additional_Info=le.fit_transform(data.Additional_Info)
data.city1=le.fit_transform(data.city1)
data.city2=le.fit_transform(data.city2)
data.city3=le.fit_transform(data.city3)
data.head()
```

dex	Airline	Source	Destination	Total_Stops	Additional_Info	Price	Date	Month	Year	city1	city2	city3	Dep_Time_Hours	De Tin _M s
0	IndiGo	0	5	0	7	3897	24	3	2019	0	13	29	22	
1	Air India	3	0	2	7	7662	1	5	2019	2	25	1	5	
2	Jet Airways	2	1	2	7	13882	9	6	2019	3	32	4	9	
3	IndiGo	3	0	1	7	6218	12	5	2019	2	34	3	18	

dex	Airline	Source	Destination	Total_Stops	Additional_Info	Price	Date	Month	Year	city1	city2	city3	Dep_Time_Hours	De Tin _M s
4	IndiGo	0	5	1	7	13302	1	3	2019	0	34	8	16	

Activity 2.3: Output Columns

• Initially in our dataset we have 19 features. So, in that some features are not more important to get output (Price).

• So i removed some unrelated features and I selected important features. So, it makes easy to understand. Now we have only 12 Output Columns.

index	Airline	Source	Destination	Total_Stops	Additional_Info	Price	Date	Month	Year	city1	city2	city3	Dep_Time_Hours	D T _
0	IndiGo	0	5	0	7	3897	24	3	2019	0	13	29	22	
1	Air India	3	0	2	7	7662	1	5	2019	2	25	1	5	
2	Jet Airways	2	1	2	7	13882	9	6	2019	3	32	4	9	
3	IndiGo	3	0	1	7	6218	12	5	2019	2	34	3	18	
4	IndiGo	0	5	1	7	13302	1	3	2019	0	34	8	16	

categorical = data[column]
categorical

index	Airline	Source	Destination	Additional_Info	city1	city2	city3	Arrival_Time_Mins	Arrival_Day
0	IndiGo	0	5	7	0	13	29	10	0
1	Air India	3	0	7	2	25	1	15	5
2	Jet Airways	2	1	7	3	32	4	25	5
3	IndiGo	3	0	7	2	34	3	30	0
4	IndiGo	0	5	7	0	34	8	35	5
5	SpiceJet	3	0	7	2	5	29	25	5
6	Jet Airways	0	5	5	0	7	8	25	5
7	Jet Airways	0	5	7	0	7	8	05	5
8	Jet Airways	0	5	5	0	7	8	25	5
9	Multiple carriers	2	1	7	3	7	6	15	5
10	Air India	2	1	7	3	6	6	00	0

Milestone 3: Exploratory Data Analysis

Activity 1: Descriptive statistical

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

categorical = data[column]

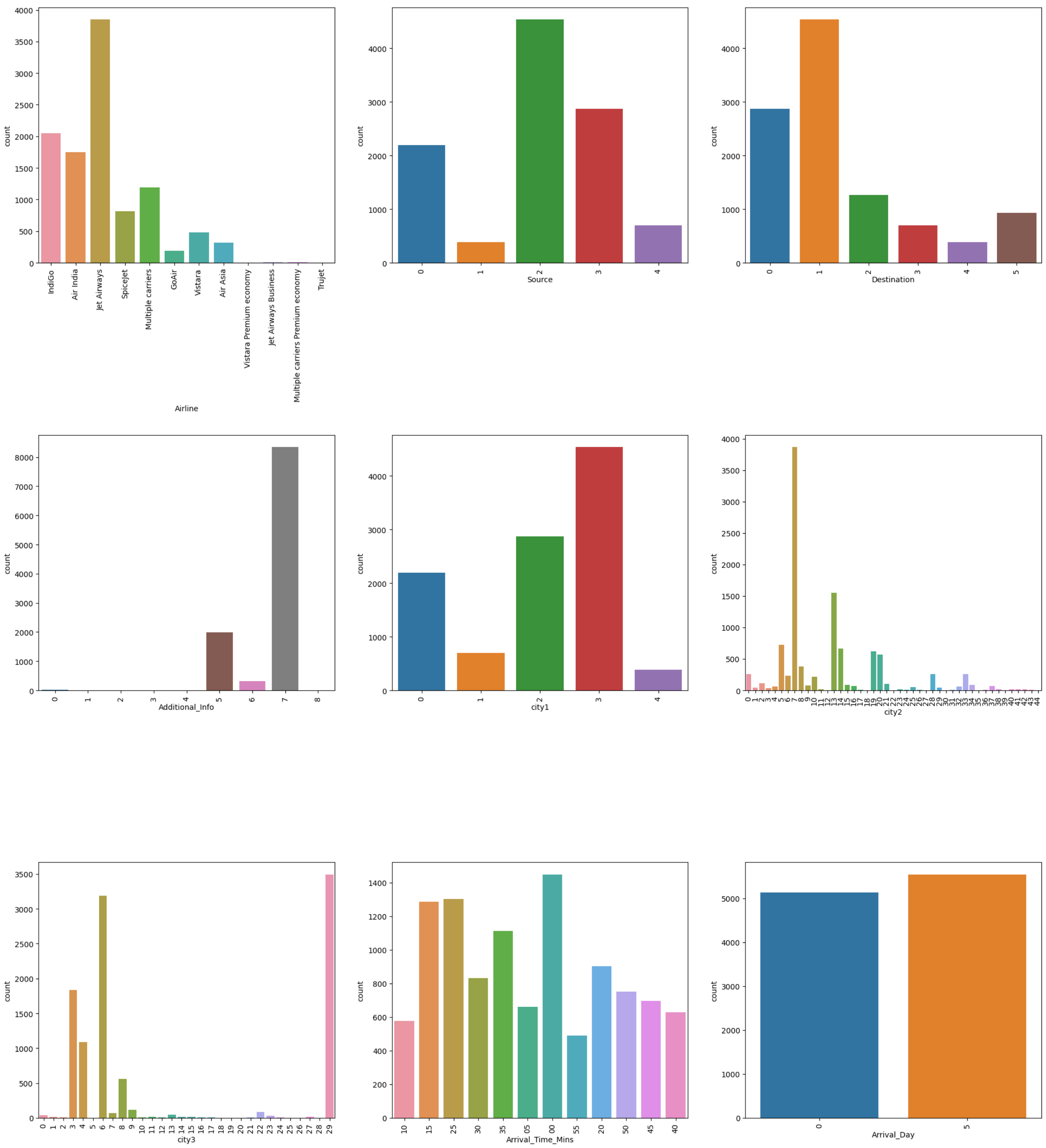
index	Source	Destination	Price	Date	Month
count	10681.0	10681.0	10681.0	10681.0	106
mean	1.952064413444434	1.4360078644321692	9086.443123303061	13.509783728115345	4.7087351371
std	1.177164791209478	1.4748360975189365	4611.075356672832	8.479448759998895	1.1643452698
min	0.0	0.0	1759.0	1.0	
25%	2.0	0.0	5277.0	6.0	
50%	2.0	1.0	8372.0	12.0	
75%	3.0	2.0	12373.0	21.0	
max	4.0	5.0	79512.0	27.0	

Activity 2: Visual Analysis

- Plotting countplots for categorical data

```
#plotting countplots for categorical data

import seaborn as sns
c=1
plt.figure(figsize=(20,45))
for i in categorical:
  plt.subplot(6,3,c)
```

```
    sns.countplot(x = data[i])
    plt.xticks(rotation=90)
    plt.tight_layout(pad=3.0)
    c=c+1
plt.show()
```



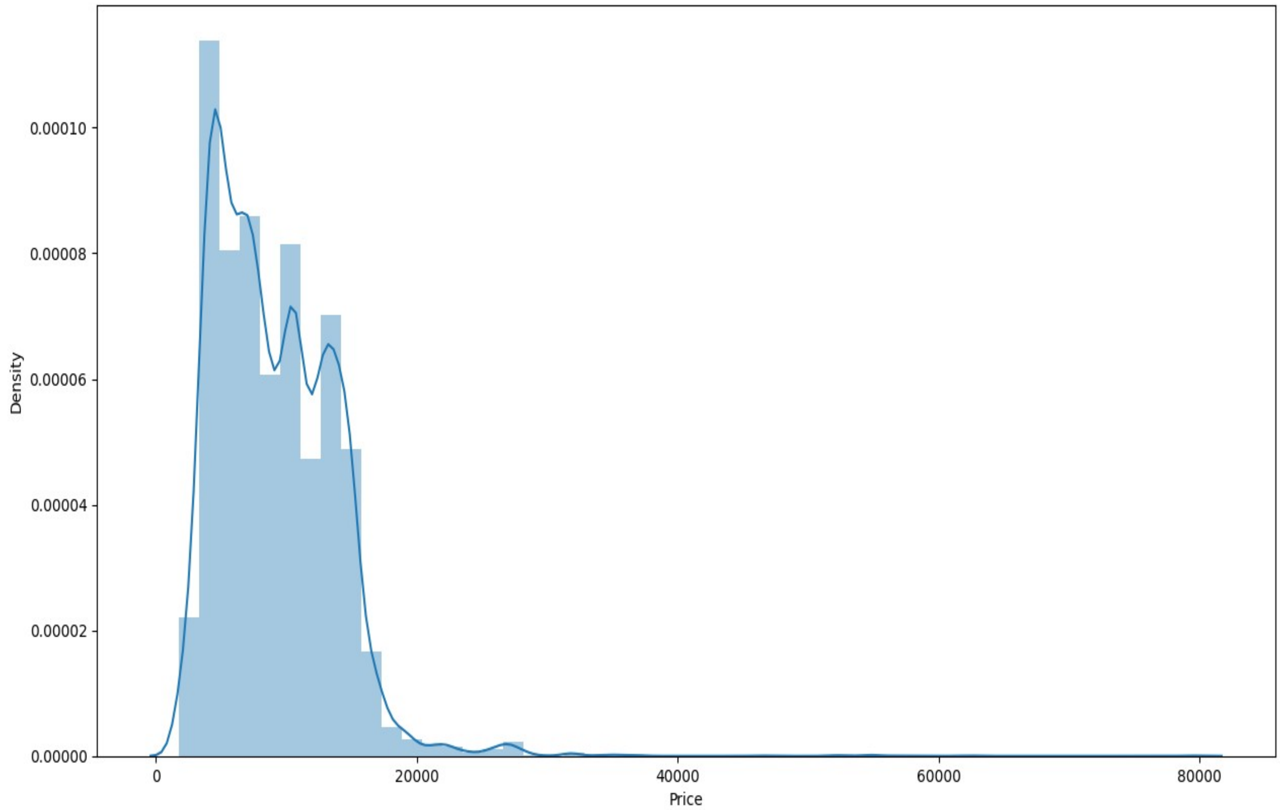
Activity 2.1: We now plot distribution plots to check the distribution in numerical data (Distribution of 'Price' Column)

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

• It estimates the probability of distribution of continous variable across various data.

```
#Distribution of 'PRICE' Column

plt.figure(figsize=(15,8))
sns.distplot(data.Price)
```

<Axes: xlabel='Price', ylabel='Density'>

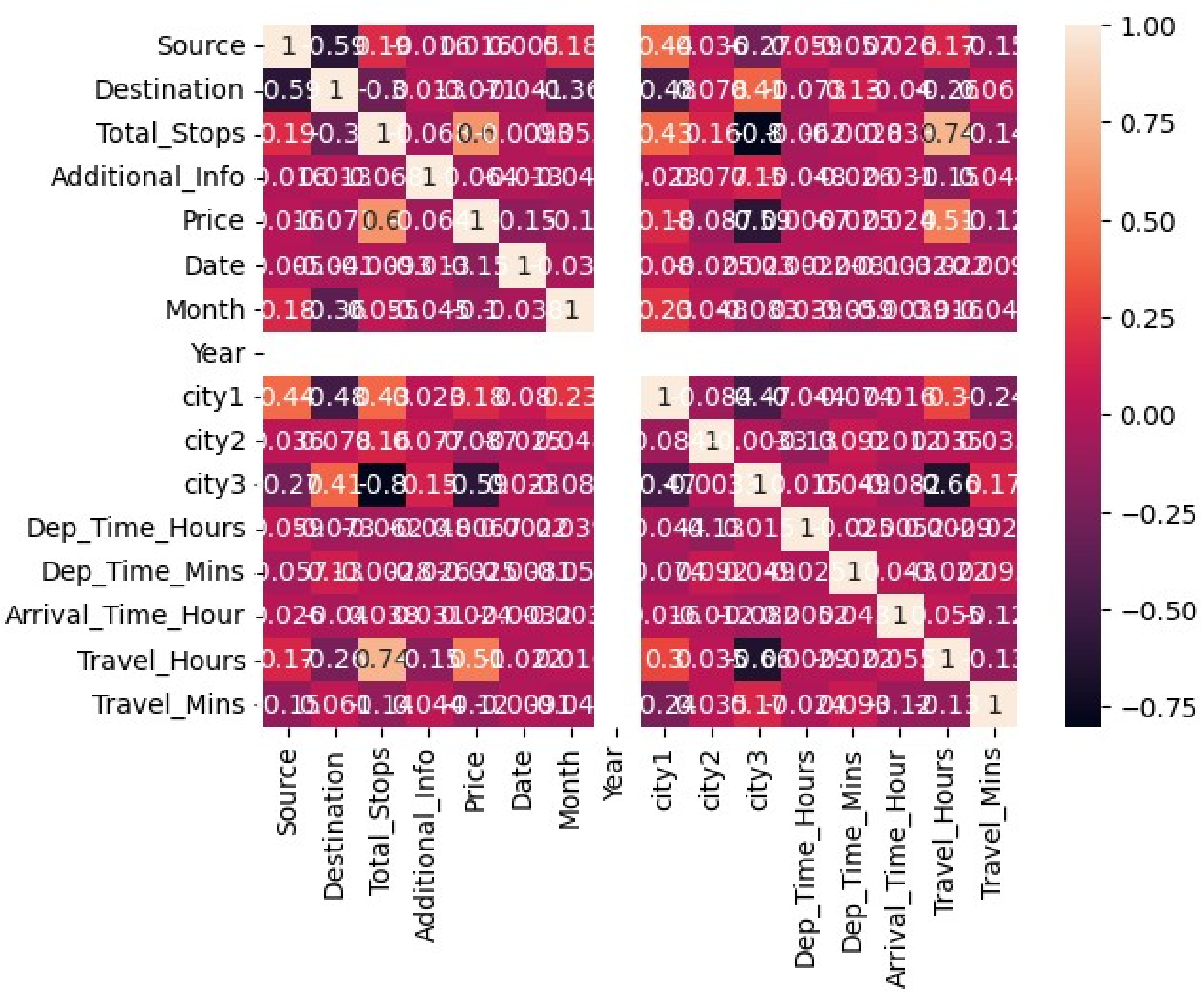


Activity 2.2: Checking the Correlation Using HeatMap

• Here, I 'm finding the correlation using HeatMap. It visualizes the data in 2-D colored maps making use of color variations. It describes the relationship variables in form of colors instead of numbers it will be plotted on both axes

. • So, by this heatmap we found that correlation between 'Arrival_date' and 'Date'. Remaining all columns don't have the any Correlation.

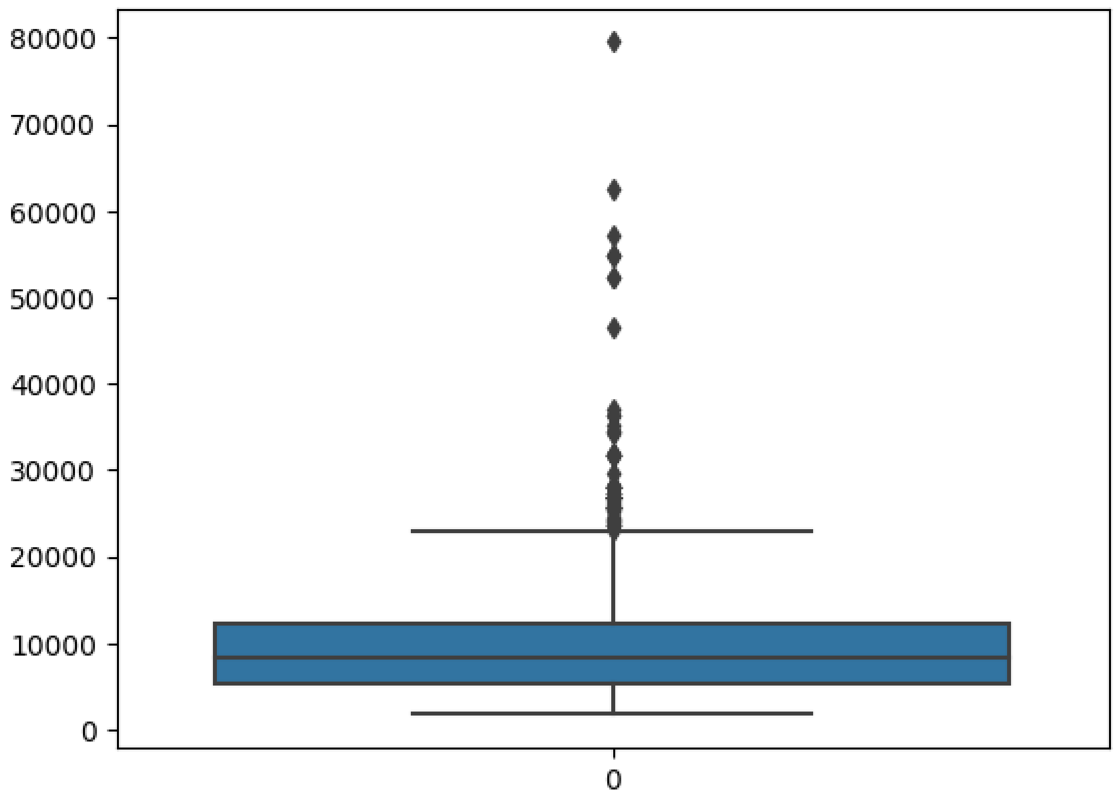
```
sns.heatmap(data.corr(),annot=True)
```



Activity 2.3: Outlier Detection for 'Price' Column

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

```
#detecting the outliers

import seaborn as sns

sns.boxplot(data[‘price’])
```



Scaling the Data

• We are taking two variables 'x' and 'y' to split the dataset as train and test.

• On x variable, drop is passed with dropping the target variable. And on y target variable('Price') is passed. • Scaling the features makes the flow of gradient descent smooth and helps algorithms quickly reach the minima of the cost function.

• Without scaling features, the algorithm maybe biased toward the feature which has values higher in magnitude. it brings every feature in the same range and the model uses every feature wisely.

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

```
x=fdata.drop('Price',axis=1)
y=fdata['Price']


###Scaling the data

from sklearn.preprocessing import StandardScaler
ss=StandardScaler()

xscaled=ss.fit_transform(x)

xscaled=pd.DataFrame(xscaled,columns=x.columns)
xscaled.head()
```

index	Airline	Source	Destination	Date	M
0	-0.41093428135292637	-1.6583538810084033	2.4166475116947033	1.2371921421203829	1.46761
1	-1.2613051152443544	0.8902616302219213	-0.973718431564698	-1.4753753141897006	0.25016
2	0.0142511355927876	0.040723126478479	-	-	1.10905

index	Airline	Source	Destination	Date	M
	85	73	0.2956452429128177	0.5318735902557585	
3	-0.41093428135292637	0.8902616302219213	-0.973718431564698	-0.1780604437805302	0.25016
4	-0.41093428135292637	-1.6583538810084033	2.4166475116947033	-1.4753753141897006	1.46761

Splitting data into train and test

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

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

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test= train_test_split(x,y,test_size=0.20,random_state=123)
x_train.head()
```

index	Airline	Source	Destination	Date	Month	Year	Dep_Time_Hour	Dep_Time_Mins	Arrival_Time_Hour	Arrival_Time_Mins	Arrival_Day
4870	4	2	1	1	6	2019	15	0	12	35	5
1251	4	3	0	12	5	2019	6	30	8	15	5
265	6	2	1	21	3	2019	11	40	1	35	5
1472	8	4	3	21	5	2019	13	15	14	45	5
495	4	3	0	6	5	2019	14	5	9	20	0

Milestone 4: Model Building

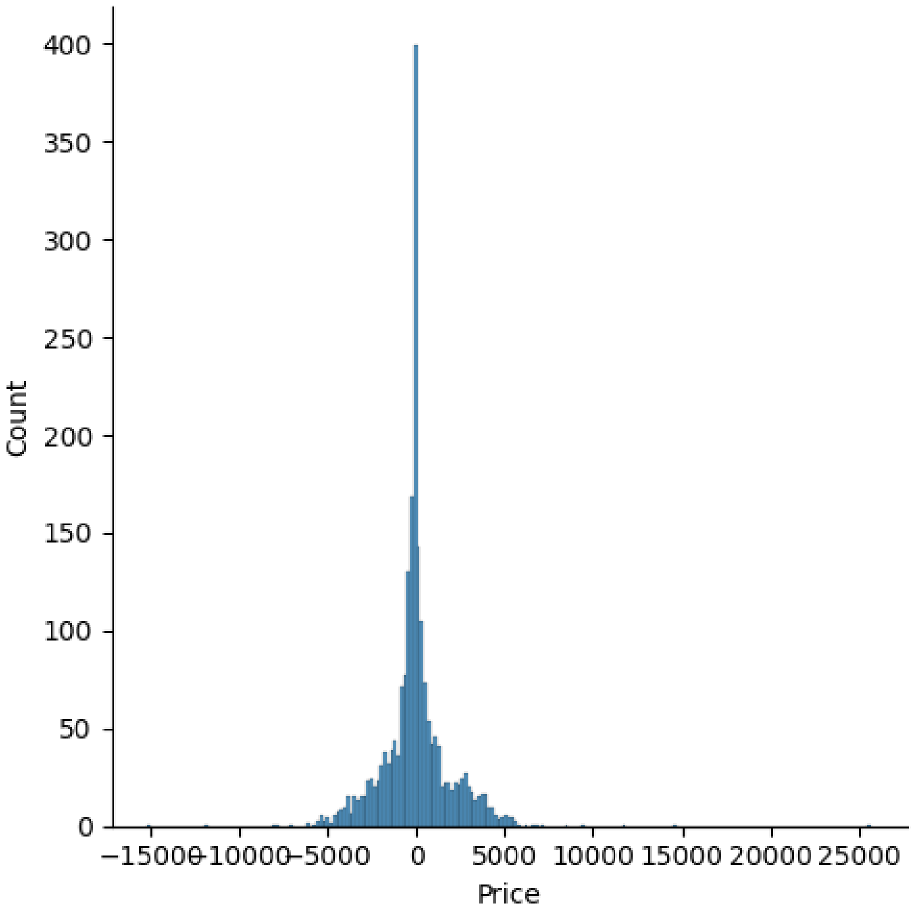
Now our data is cleaned and it's time to build the model. We can train our data on different algorithms. for this project we are applying four regression algorithms. The best model is saved based on its performance.

Activity 1: Using Ensemble Techniques

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

```
from sklearn.metrics import r2_score,mean_absolute_error,mean_squared_er
ror
def predict(ml_model):
    print('Model is: {}'.format(ml_model))
    model= ml_model.fit(x_train,y_train)
    print("Training score: {}".format(model.score (x_train,y_train)))
    predictions = model.predict(x_test)
    print("Predictions are: {}".format(predictions))
    print('\n')
    r2score=r2_score(y_test,predictions)
    print("r2 score is: {}".format(r2score))

    print('MAE:{}'.format(mean_absolute_error(y_test,predictions)))
    print('MSE:{}'.format(mean_squared_error(y_test,predictions)))
    print('RMSE:{}'.format(np.sqrt(mean_squared_error(y_test,predictions))))

    sns.displot(y_test-predictions)
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import GradientBoostingRegressor,RandomForestReg
ressor


from sklearn.metrics import r2_score,mean_absolute_error,mean_squared_er
ror
def predict(ml_model):
    print('Model is: {}'.format(ml_model))
    model= ml_model.fit(x_train,y_train)
    print("Training score: {}".format(model.score (x_train,y_train)))
    predictions = model.predict(x_test)
```

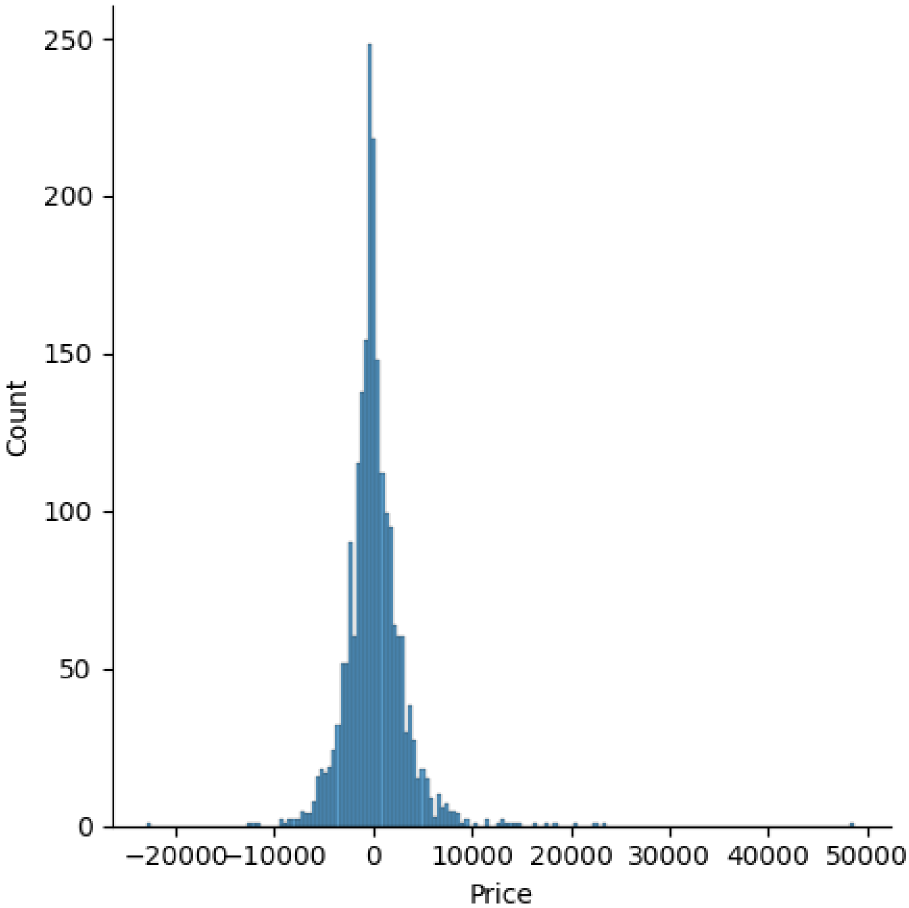
```
    print("Predictions are: {}".format(predictions))
    print('\n')
    r2score=r2_score(y_test,predictions)
    print("r2 score is: {}".format(r2score))

    print('MAE:{}'.format(mean_absolute_error(y_test,predictions)))
    print('MSE:{}'.format(mean_squared_error(y_test,predictions)))
    print('RMSE:{}'.format(np.sqrt(mean_squared_error(y_test,predictions))))

    sns.displot(y_test-predictions)

predict(RandomForestRegressor())
Model is: RandomForestRegressor()
Training score: 0.9520047279248214
Predictions are: [ 8454.733 13495.64  14811.42  ... 14703.57   5950.2   11696.526]


r2 score is: 0.7927034177527617
MAE:1253.0022491967945
MSE:4065172.517650502
RMSE:2016.2272981116246
```



predict(KNeighborsRegressor())

Model is: KNeighborsRegressor()
Training score: 0.7262239247009137
Predictions are: [7629.6 9698.6 12907.8 ... 14293.6 6692.4 5291.8]

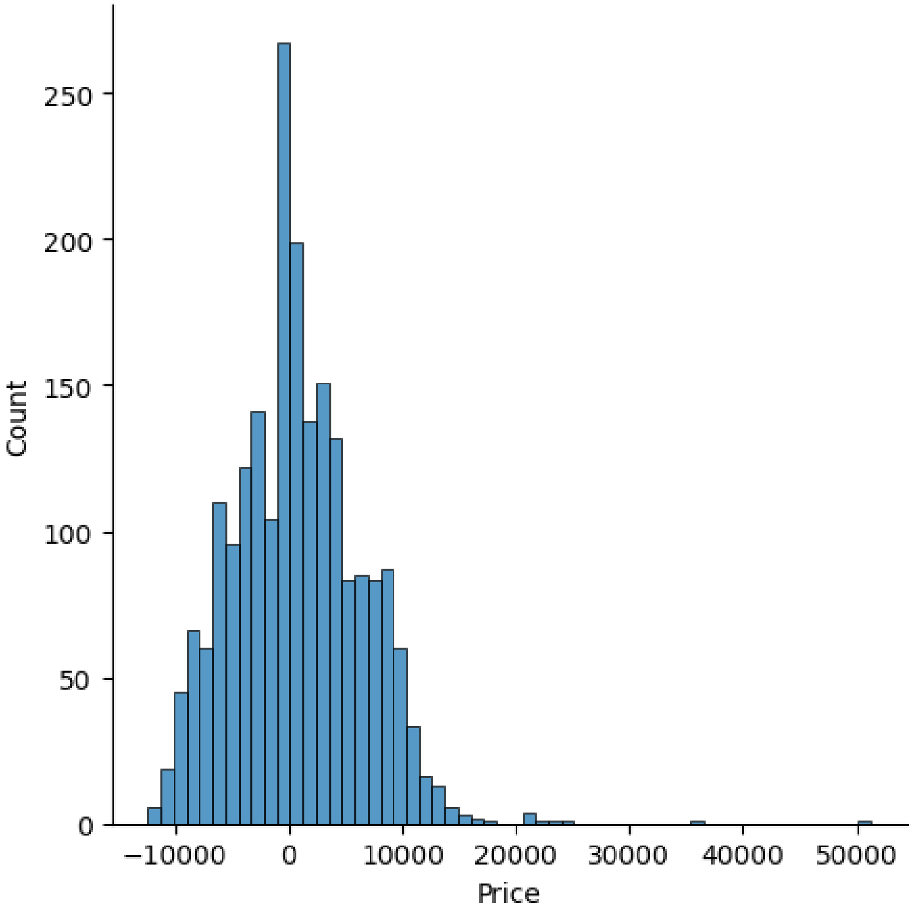
r2 score is: 0.5066647660821886
MAE:1955.9190453907347
MSE:9674509.889021993
RMSE:3110.387417834311



predict(LogisticRegression())

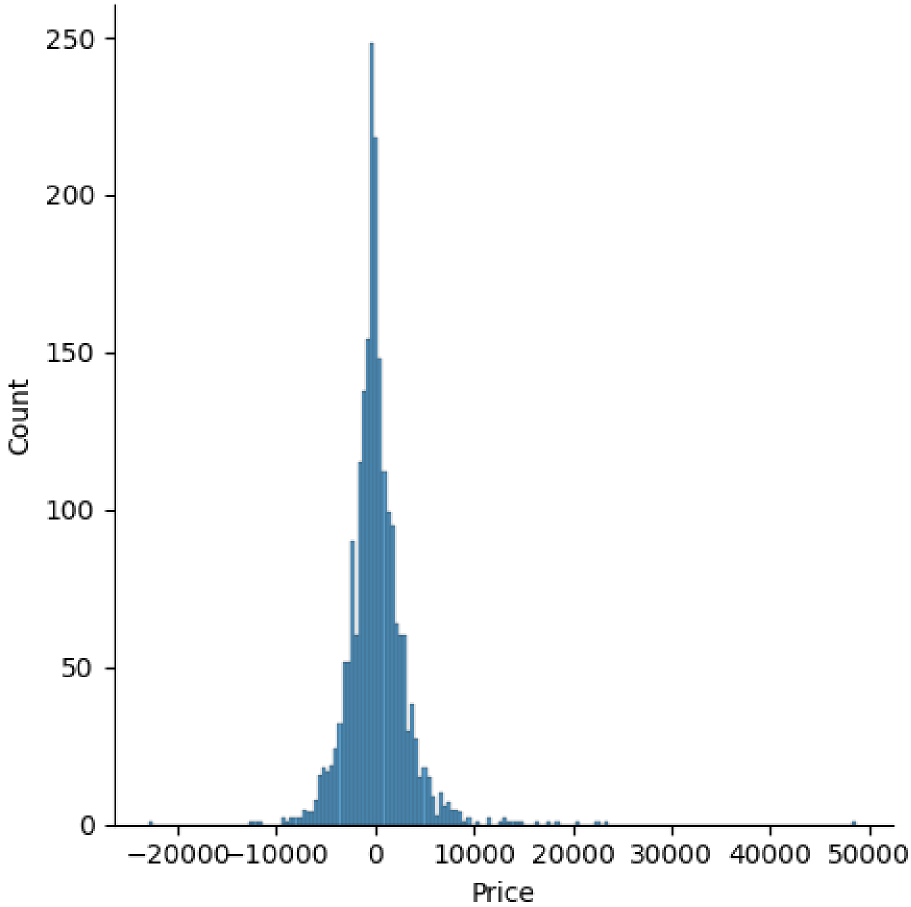
Model is: LogisticRegression()
Training score: 0.042837078651685394
Predictions are: [15129 3943 13941 ... 14714 12898 4174]

r2 score is: -0.6631630011302538
MAE:4383.201684604586
MSE:32615320.770238653
RMSE:5710.982469789122



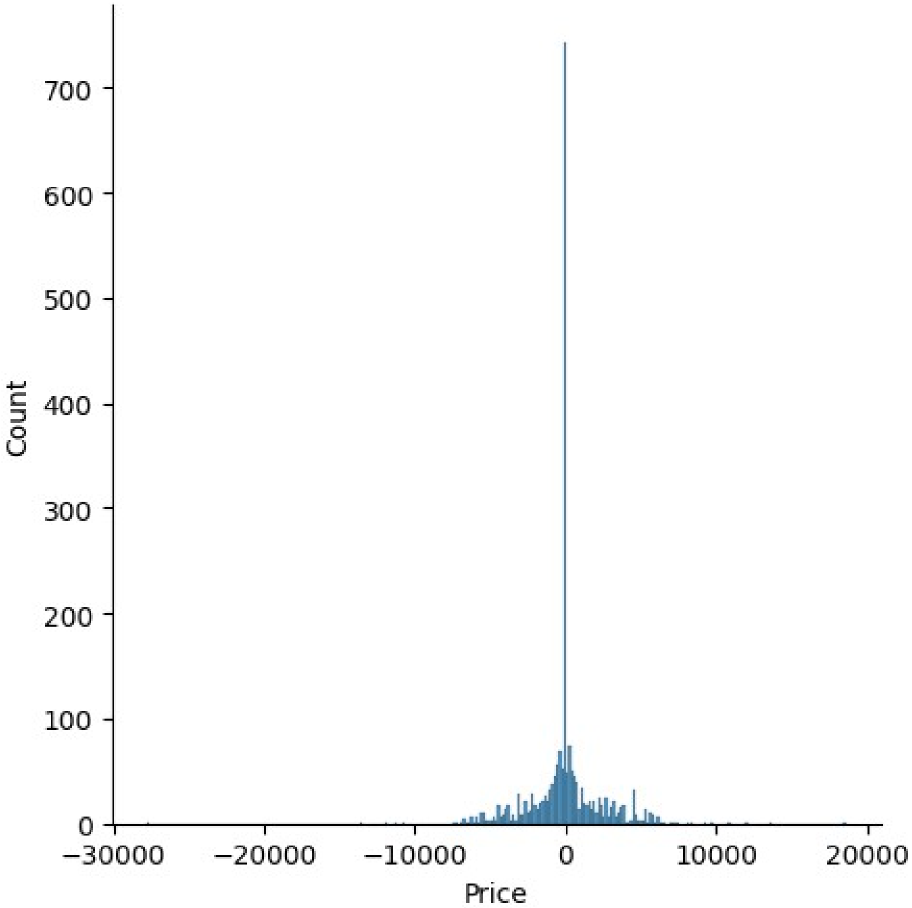
predict(KNeighborsRegressor())
Model is: KNeighborsRegressor()
Training score: 0.7262239247009137
Predictions are: [7629.6 9698.6 12907.8 ... 14293.6 6692.4 5291.8]

r2 score is: 0.5066647660821886
MAE:1955.9190453907347
MSE:9674509.889021993
RMSE:3110.387417834311



predict(DecisionTreeRegressor())
Model is: DecisionTreeRegressor()
Training score: 0.9696975821560773
Predictions are: [7618. 13727. 15129. ... 14924. 6171. 12488.]

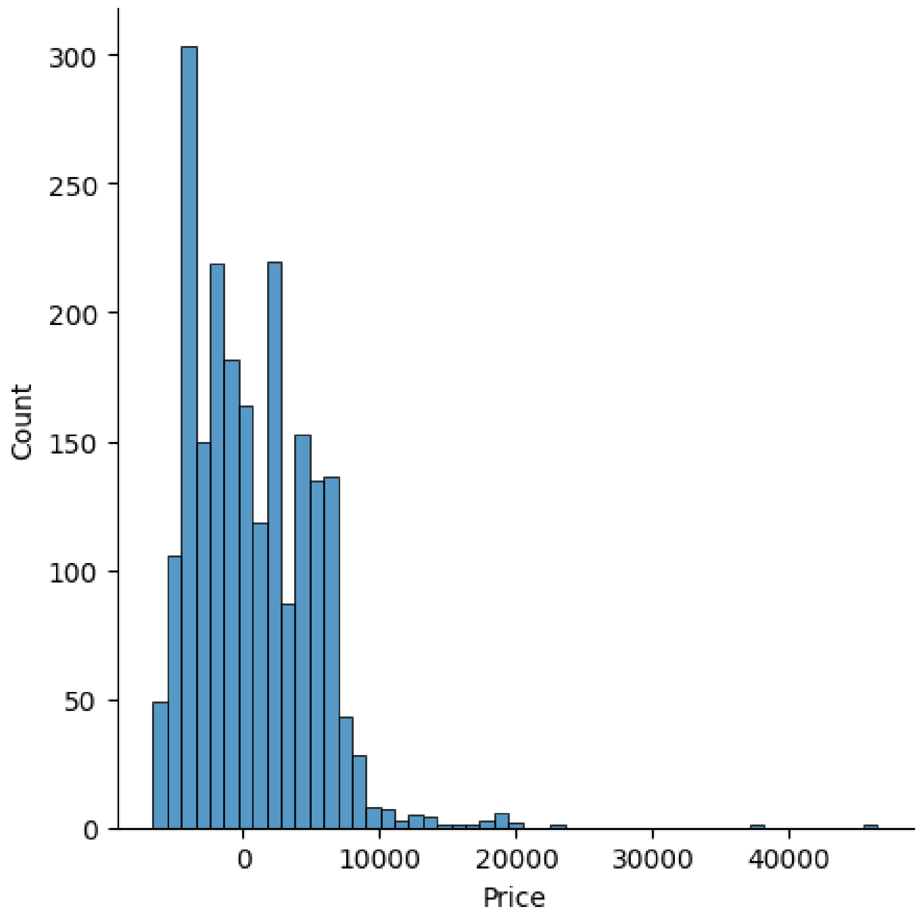
r2 score is: 0.6792060181755581
MAE:1420.1186398377788
MSE:6290903.904942546
RMSE:2508.167439574668




```
from sklearn.svm import SVR
predict(SVR())
```

Model is: SVR()
Training score: -0.025316833905048686
Predictions are: [8372.30814222 8372.10064183 8372.10281517 ...
8372.39456263 8372.21469149
 8372.10552597]

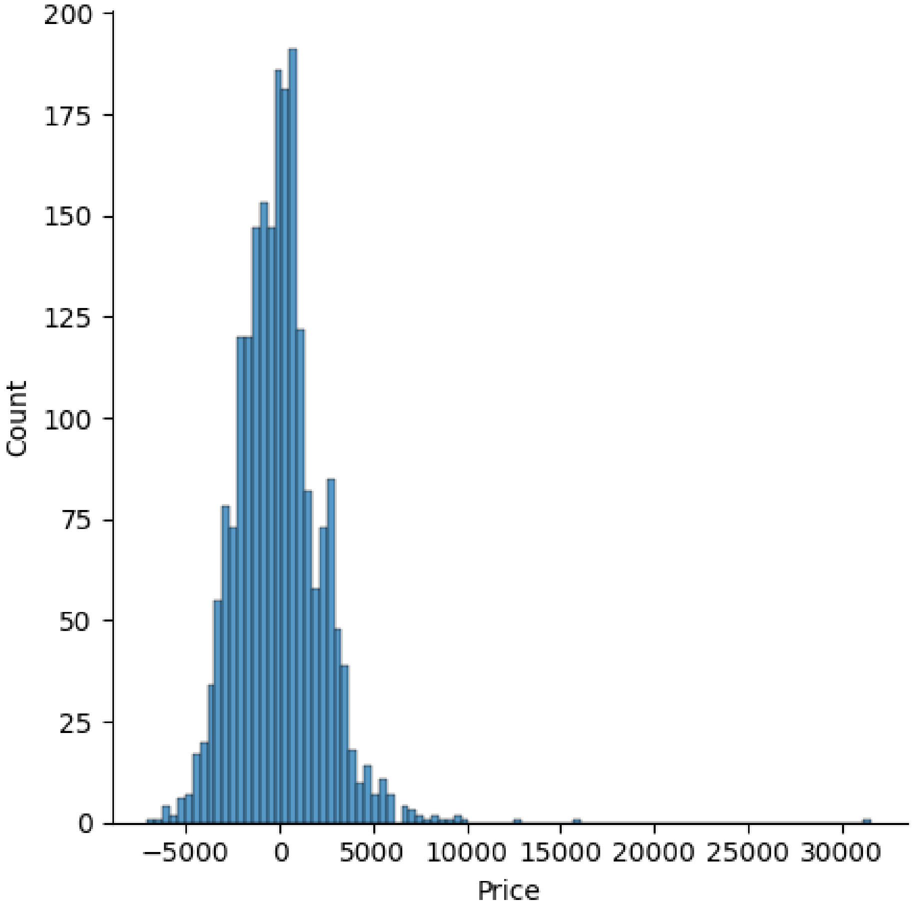
r2 score is: -0.018884049680442555
MAE:3507.694654077979
MSE:19980741.566174872
RMSE:4469.982278060493



predict(GradientBoostingRegressor())

Model is: GradientBoostingRegressor()
Training score: 0.7432674171188071
Predictions are: [10240.2533732 11191.99025903 12475.08997158 ...
12782.76349772
  5187.48765406 11599.11714502]

r2 score is: 0.7311683133344324
MAE:1678.1742821142932
MSE:5271901.604258386
RMSE:2296.0621952069127



Activity 2: Regression Model

KNeighborsRegressor, SVR, DecisionTreeRegressor

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

predict(KNeighborsRegressor())
Model is: KNeighborsRegressor()
Training score: 0.7262239247009137
Predictions are: [7629.6 9698.6 12907.8 ... 14293.6 6692.4 5291.8]

r2 score is: 0.5066647660821886
MAE:1955.9190453907347
MSE:9674509.889021993

RMSE:3110.387417834311

```
from sklearn.svm import SVR
predict(SVR())
```

Model is: SVR()
Training score: -0.025316833905048686
Predictions are: [8372.30814222 8372.10064183 8372.10281517 ...
8372.39456263 8372.21469149
 8372.10552597]

r2 score is: -0.01888404968044255
MAE:3507.694654077979
MSE:19980741.566174872
RMSE:4469.982278060493

```
predict(DecisionTreeRegressor())
```

Model is: DecisionTreeRegressor()
Training score: 0.9696975821560773
Predictions are: [7618. 13727. 15129. ... 14924. 6171. 12488.]

r2 score is: 0.6792060181755581
MAE:1420.1186398377788
MSE:6290903.904942546
RMSE:2508.167439574668

Activity 3: Checking Cross Validation for RandomForestRegressor

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

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

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

Activity 4: Hypertuning the model

RandomSearch CV is a technique used to validate the model with different parameter combinations, by creating a random of parameters and trying all the combinations to compare which combination gave the best results. We apply random search on our model. From sklearn, cross_val_score is used to evaluate the score of the model. On the parameters, we have given rf (model name), x, y, cv (as 3 folds). Our model is performing well

```
from sklearn.model_selection import RandomizedSearchCV
random_grid = {

   'n_estimators' :[100,120,150,180,200,220],
   'max_features' :['auto','sqrt'],
   'max_depth' :[5,10,15,20],
   }

rf=RandomForestRegressor()
rf_random=RandomizedSearchCV(estimator=rf,param_distributions=random_grid,cv=3,verbose=2,n_jobs=-1,)

rf_random.fit(x_train,y_train)


#best parameters

rf_random.best_params_
```

Fitting 3 folds for each of 10 candidates, totalling 30 fits
{'n_estimators': 100, 'max_features': 'auto', 'max_depth': 15}

```
from sklearn.model_selection import RandomizedSearchCV
```

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

rf_res.fit(x_train,y_train)
```

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

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

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

gb_res.fit(x_train,y_train)
```

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

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

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

Accuracy

Checking Train and Test Accuracy by RandomSearchCV using RandomForestRegression Model

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

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

Checking Train and Test Accuracy by RandomSearchCV using KNN Model2

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

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

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

Evaluating Performance Of The Model And Saving The Model

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

Note: To understand cross validation, refer this link. https://towardsdatascience.com/cross-validation-explained-evaluating-estimator-performance-e51e5430ff85.

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

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

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

```
price_list
```

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

2137 rows × 1 columns

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


Milestone 6: Model Deployment

In the Milestone, you will see the model deployment

Activity 1: Save The Best Model

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

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

Activity 2:Integrate With Web Framework

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

This section has the following tasks

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

Activity 2.1: Building Html Pages

For this project create two HTML files namely

- •home.html
- •predict.html
- •submit.html

and save them in the templates folder.

Activity 2.2: Build Python code:

Import the libraries

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

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

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

Render HTML page:

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

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

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

Retrieves the value from UI:

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




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

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


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

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

Main Function:

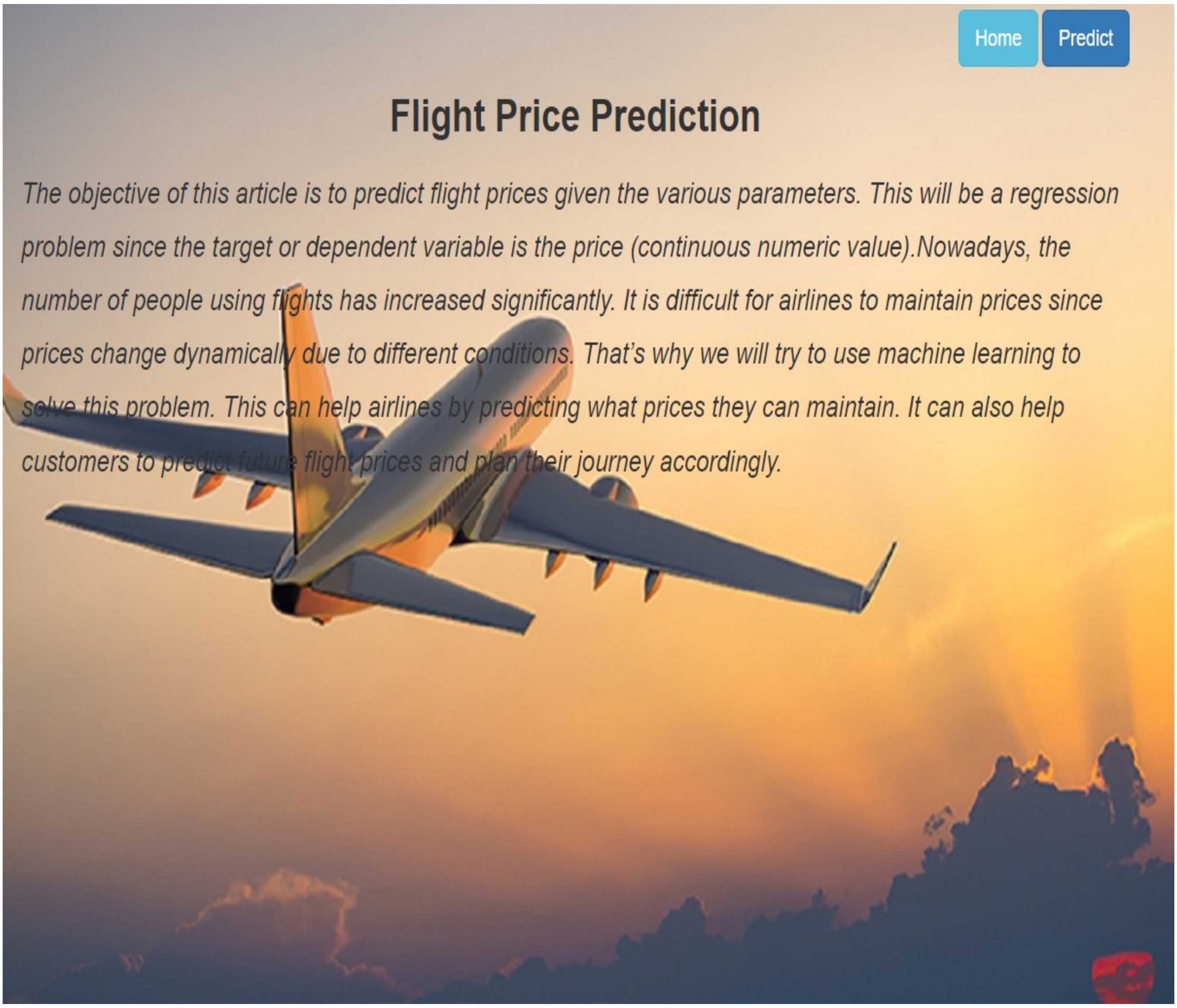
```
if __name__ == "__main__":
    app.run(debug=False)
```

Run The Web Application

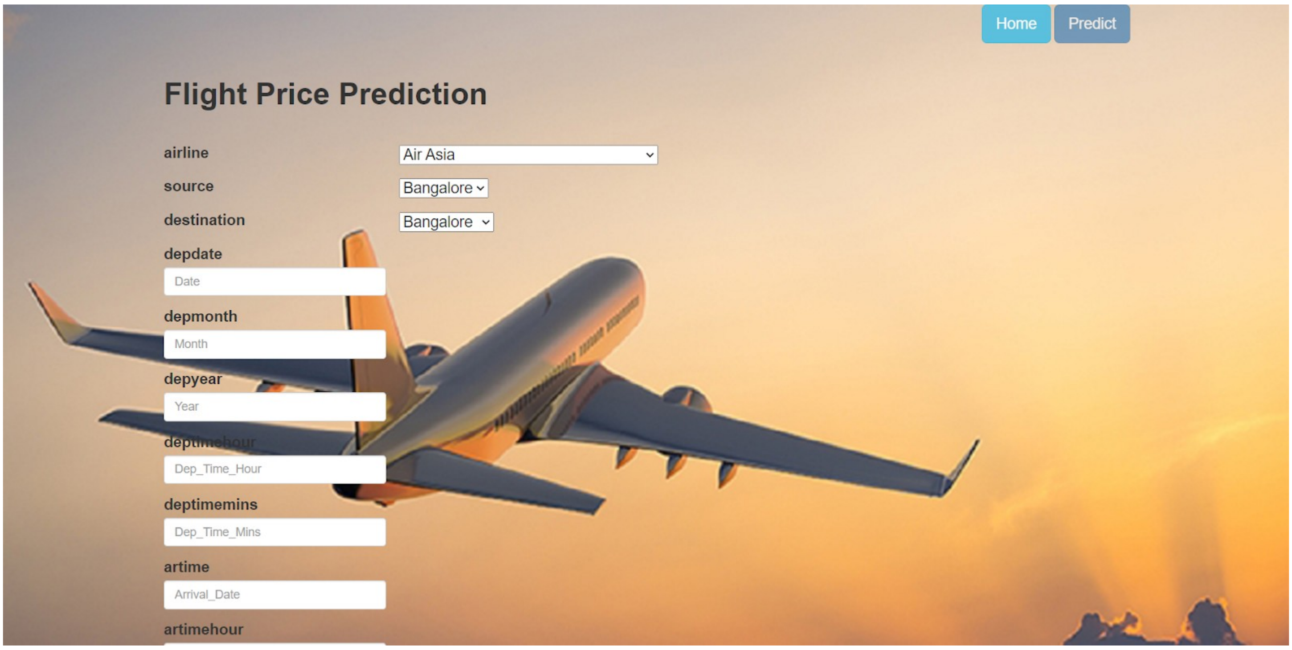
- Open anaconda prompt from the start menu
- Navigate to the folder where your python script is.
- Now type “python app.py” command
- Navigate to the localhost where you can view your web page.
- Click on the predict button from the top left corner, enter the inputs, click on the submit button, and see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

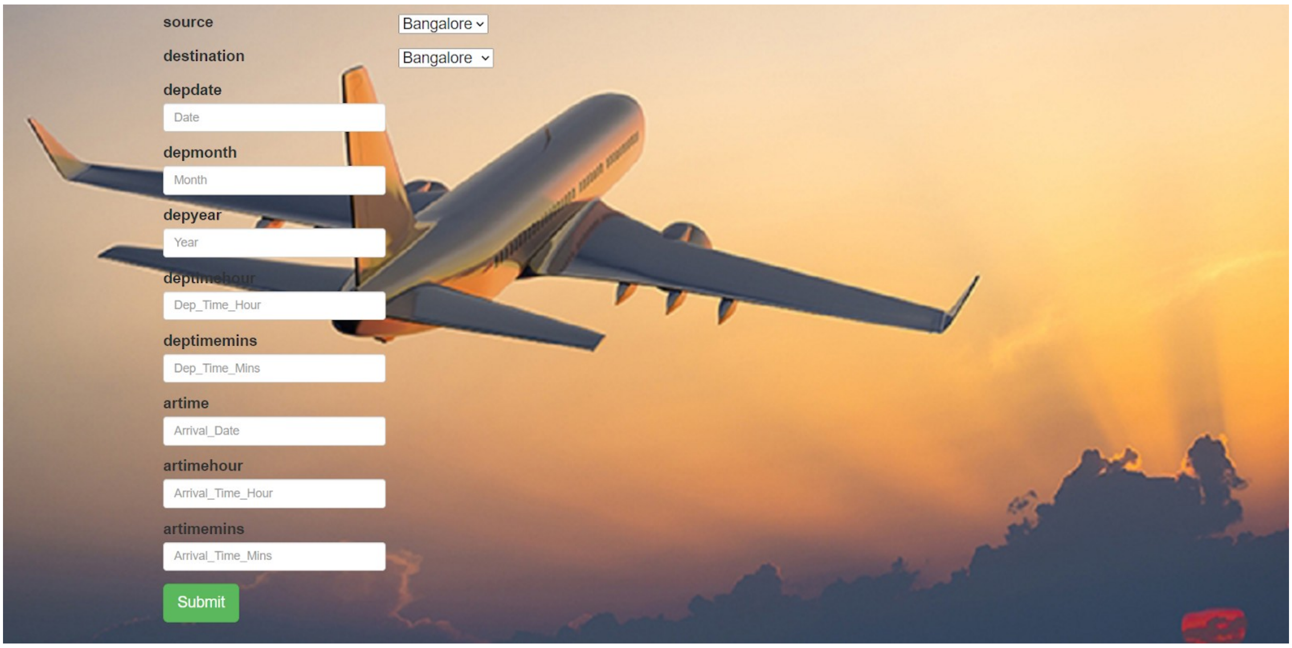
```
* Serving Flask app "app" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a p
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

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

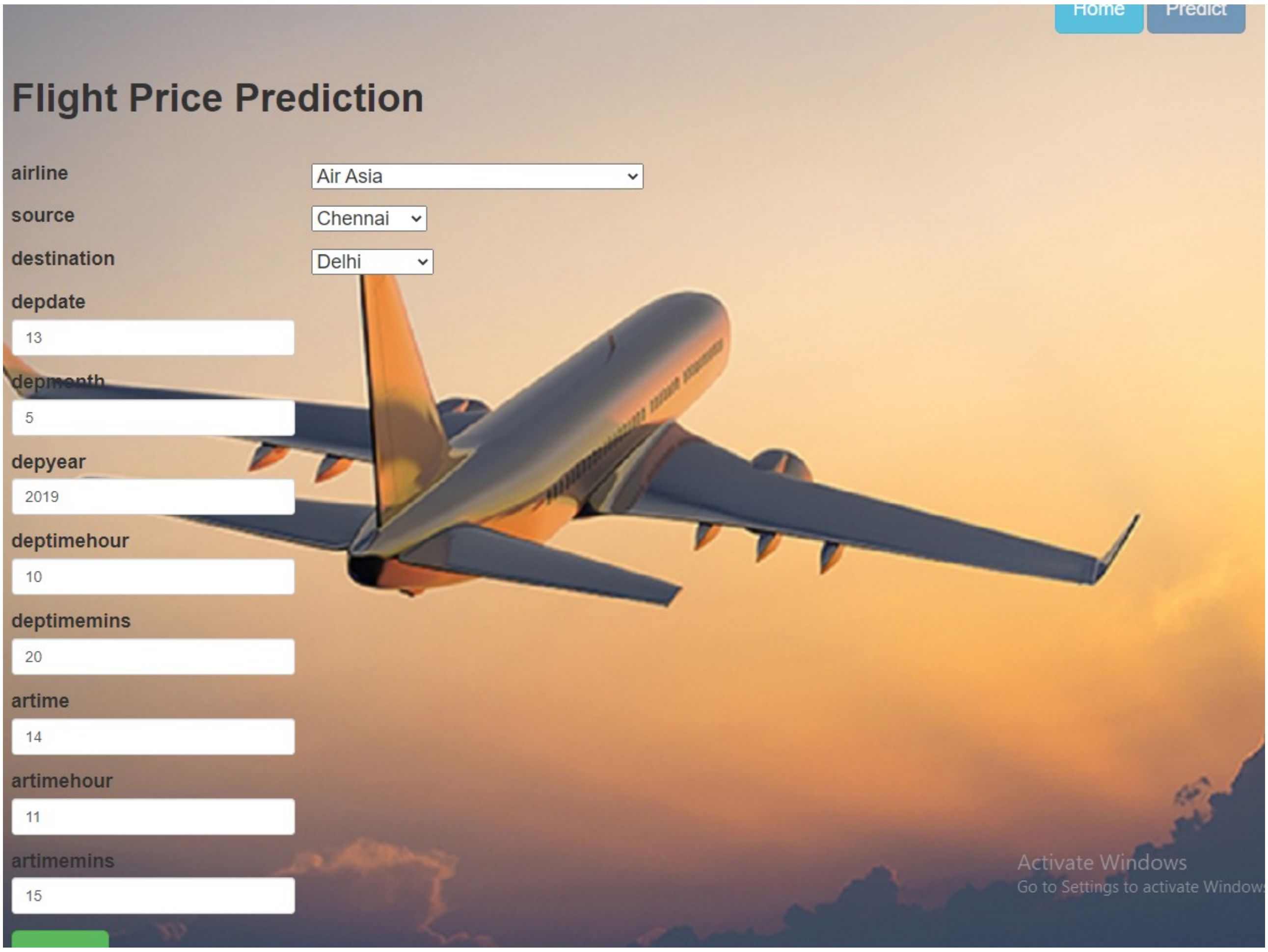


Now, when you click on Predict button you will get redirected to the prediction page.

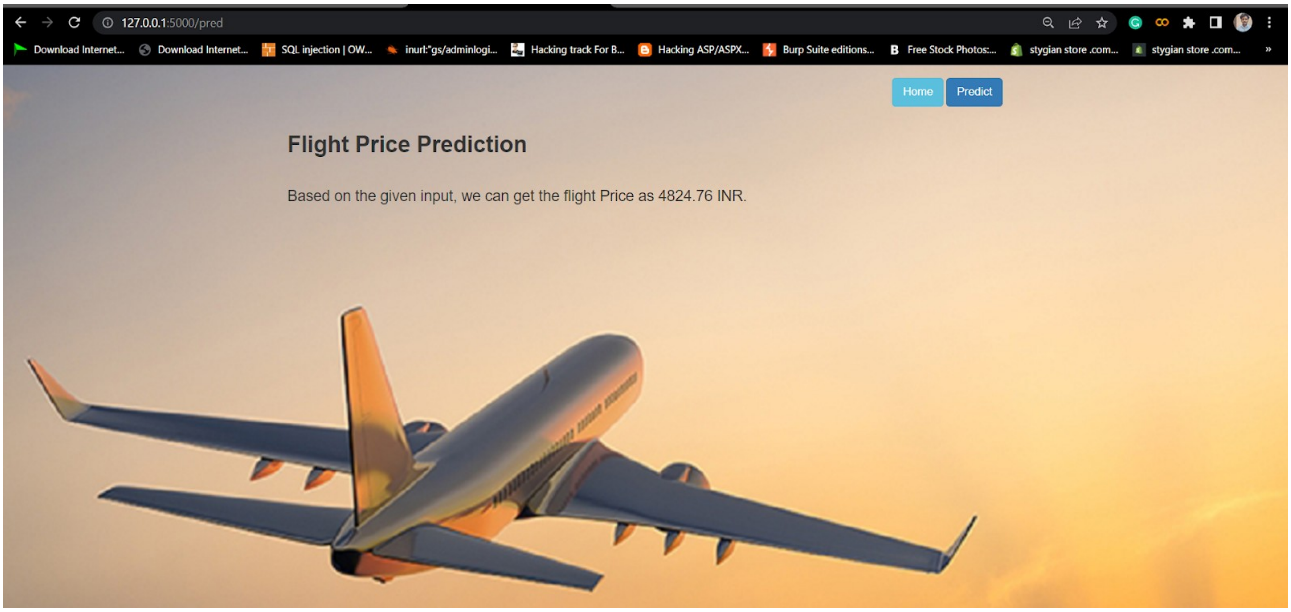




Input 1- Now, the user will give inputs to get the predicted result after clicking onto the submit button.



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



Milestone 7: Project Demonstration & Documentation

Below mentioned deliverables to be submitted along with other deliverables

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

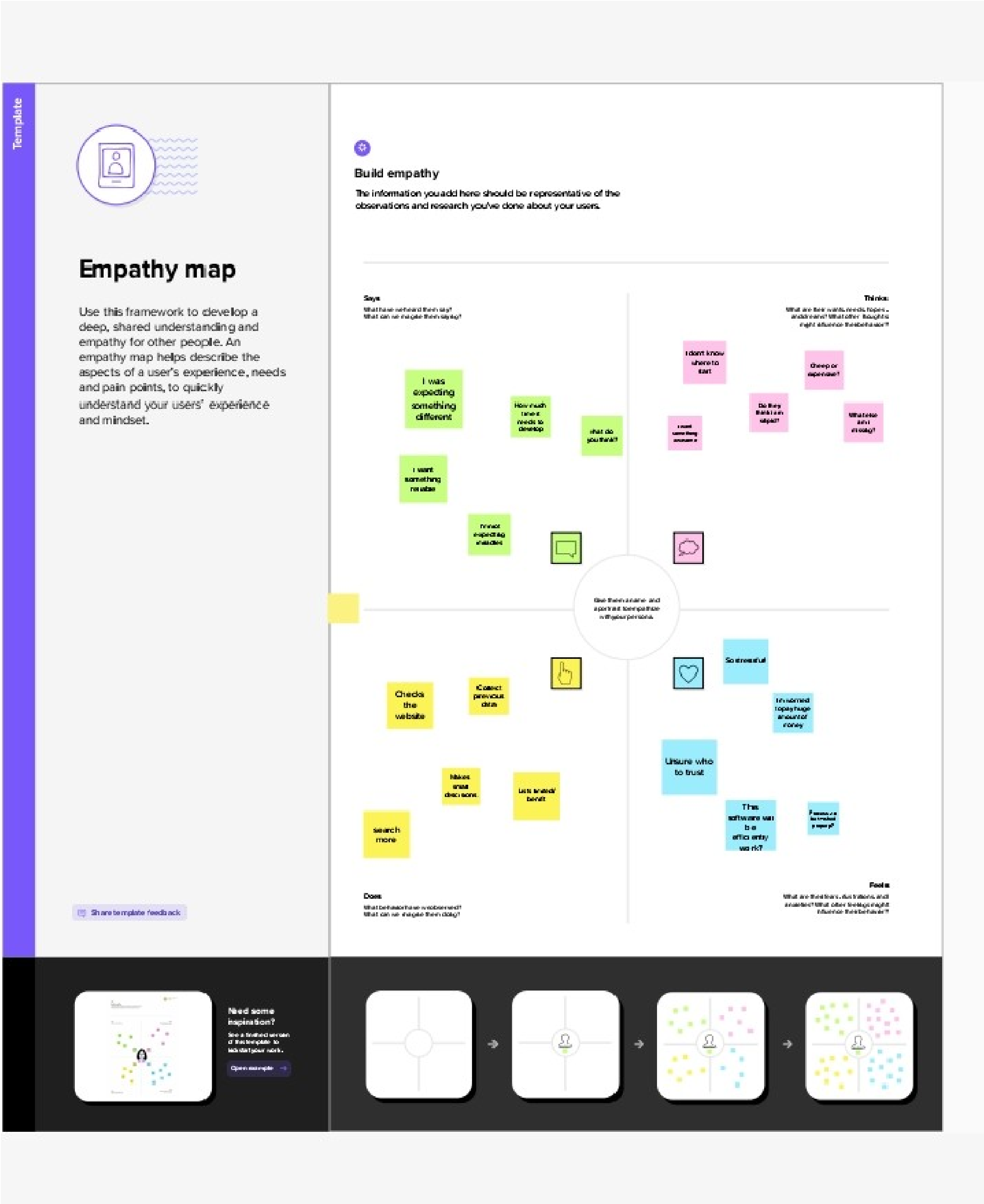
Activity 2:- Project Documentation-Step by step project development procedure Create document as per the template provided

1.2 PURPOSE

With consideration of some features like arrival time, departure time as well as time to purchase the ticket using these factors prices can be predict. due to this factors there may be change in airline fare prices and also detect how factors are related to being change of Flight ticket.

2. PROBLEM DEFINITION & DESIGN THINKING

2.1 Empathy Map



2.2 Ideation & Brainstorming Map

Ideation Phase

Empathize &Discover

Empathy Map Canvas:

In the idea phase, we have empathized as our client Indian airlines and we have acquired the details, which are represented in the empathy map given

Date	14 March 2023
Team ID	NM2023TMID19165
Project Name	Optimizing Flight Booking Decisions through Machine Learning Price Predictions
Maximum Mark	5 Marks

below.

Empathy map canvas

Use this framework to empathize with a customer, user, or any person who is affected by a team's work. Document and discuss your observations and note your assumptions to gain more empathy for the people you serve.

Originally created by Dave Gray at

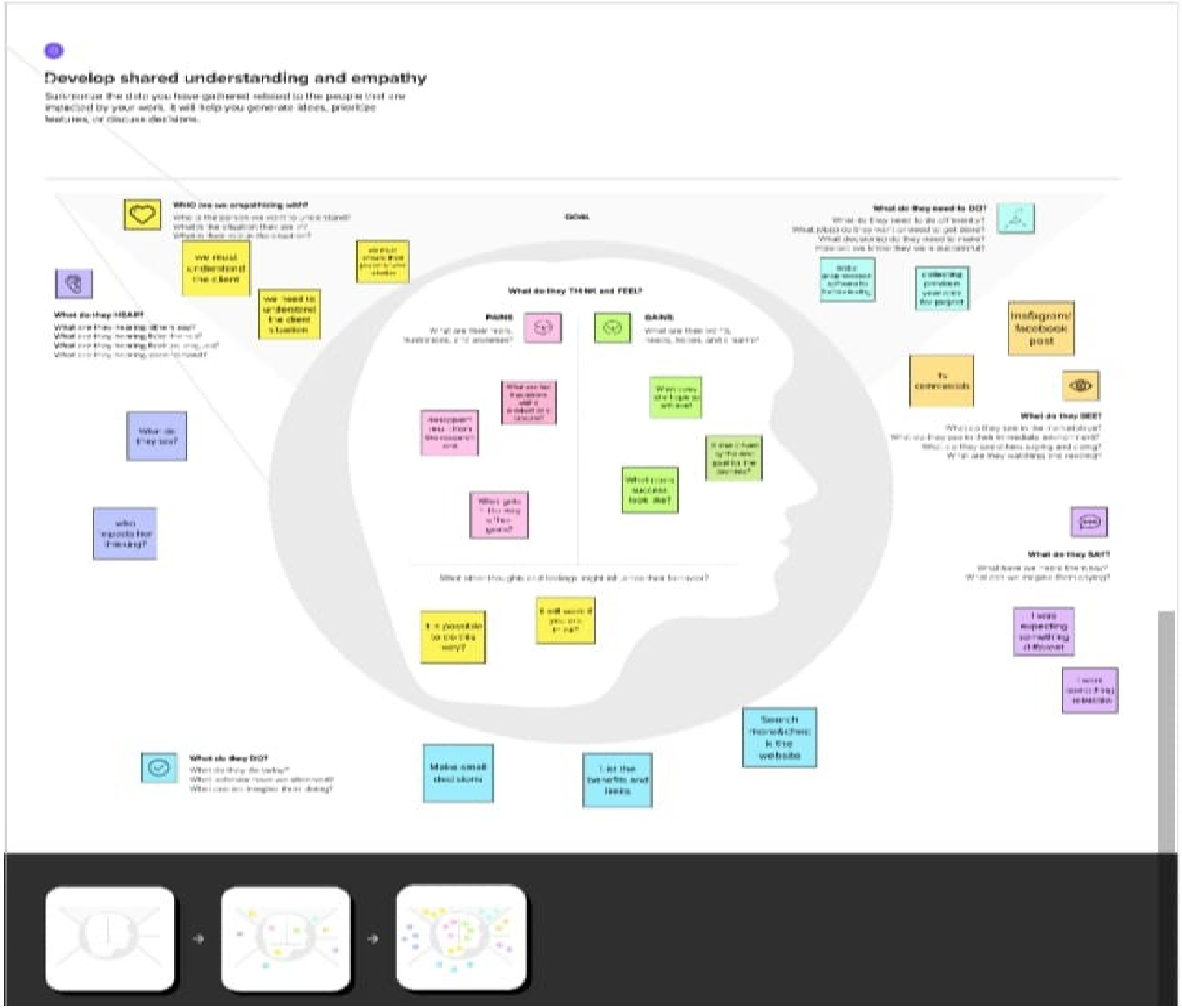
XPLANE

Share template feedback

Need some inspiration?

See a finished version of this template to kickstart your work.

Open example


Develop shared understanding and empathy
WHO are we empathizing with?
GOAL
What do they need to DO?
we must understand the client
we need to understand the client situation
What do they HEAR?
What do they THINK and FEEL?
PAINS
GAINS
Instagram/ facebook post
What do they SEE?
What do they SAY?
I was expecting something different
What do they DO?
Make small decisions
List the benefits and links
Search more&check the website

3. RESULT


Flight Price Prediction
Optimize Flight Booking
airline :
Source :
Destination :
Date:
Month :
Year :
Dep_Time_Hour :
Dep_Time_Mins :
Arrival_Time_Hour :
Arrival_Time_Mins :
Arrival_Day :
Predict
gettyimages
spooh
155150766


Flight Price Prediction
Prediction Results!!
{{prediction_results}}
gettyimages
spooh

4 . Advantages & Disadvantages

Advantage:

- The prediction will help a traveller to decide a specific airline as per his\her budget.
- Airline corporations are using complex startegies and methods to assign airfare prices in a dynamic fashion.
- Due to the high complexity of the pricing models applied by the airlines.
- User can login with valid credentials in orders to access the application .
- A traveller can access this module to get the future price prediction of individual airlines.

Disadvantages:

- Improper data will result in incorrect fare predictions.
- It needs active internet connection .
- It is based on historical data.
- Flight price prediction apps are not suitable for business travel.
- Result has shown light GBM regression outperforms other conventional regressors with extensive experiment on a large real world dataset.
- Flight delays only irritate air passengers and disrupts their schedules.

5 . APPLICATIONS

A Flight price prediction application which predicts fares of flight for a particular date based on various parameters like Source, Destination, Stops & Airline.Data used in this Project is publicly available at Kaggle. The dataset goes through Data Cleaning, Data Wrangling , Exploratory Data Analysis which gives insights about the data and later uses Machine

Learning techniques to train the data for prediction.
It is a regression problem which is solved using RandomForestRegressor ML Algorithm which generates accurate results for price prediction. A web application is created using Flask through which user can interact and get accurate prediction of flight fares.

6. CONCLUSION:

We further proceed to test the object that we saved using joblib, and create a dataframe of predicted values –

```
model = joblib.load('flight_price.obj')
pred = model.predict(x_test)

#Creating a dataframe with actual and predicted values

predicted_values = pd.DataFrame({'Actual':y_test,'Predicted':pred})
```

We receive the following metrics as our final metrics –

```
R2 score for test data is  0.8721948958355091
R2 for train data 0.9144685033282565
Mean absolute error is  974.5192099594759
Mean squared error is  2627378.1594185294
Root mean squared error is  1620.918924381639
```

We have achieved an r2_score value of 87%, meaning that we are actually able to predict values quite near to the actual prices, for majority of the rows. A glimpse of our resulting dataframe is attached below.

	Actual	Predicted
8161	10703	10588.869238
6423	13587	10804.367668
3102	12819	14140.445786
5797	8610	8960.063030
7180	14714	14131.448592
...	...	...
2216	3210	3738.333383
5327	1965	2274.839825
5663	8479	8443.645208
6160	11467	13593.043350
3625	9316	10546.197245

These are the predictions on the training data, but we also had a test file for which we need to predict the outputs.

We load the test file, apply all the data modeling processes and operations on our test data similar to what we did with the train data, and then make the final prediction using the saved model object.

```
#Loading the saved object
flight_price = joblib.load('flight_price.obj')
```

```
prices = flight_price.predict(testset)
```

```
#Printing the final result
prices
```

```
array([14137.44026214,  5760.86187326, 11554.97811437, ...,
       17203.89074664, 14118.47632041,  9920.10389661])
```

```
#Storing predicted data to dataframe
price_list = pd.DataFrame({'Price':prices})
```

```
price_list
```

	Price
0	14137.440262
1	5760.861873
2	11554.978114
3	10418.727189
4	3373.438961
...	...
2665	8198.451471
2666	4728.137735
2667	17203.890747
2668	14118.476320
2669	9920.103897

2670 rows × 1 columns

Hence, at the end, we were successfully able to train our regression model ‘Gradient Boosting Regressor’ to predict the flights of prices with an r2_score of 87%, and have achieved the required task successfully.

7. FUTURE SCOPE :

- More routes can be added and the same analysis can be expanded to major airports and travel routes in India.
- The analysis can be done by increasing the data points and increasing the historical data used.
- That will train the model better giving better accuracies and more savings.
- More rules can be added in the Rule based learning based on our understanding of the industry,
- also incorporating the offer periods given by the airlines.

- Developing a more user friendly interface for various routes giving more flexibility to the user

8. APPENDIX:

A . SOURCE CODE

```
app = Flask(__name__)

model = pickle.load(open('flightprice.pkl', 'rb'))


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


@app.route('/getdata', methods=['POST'])
def pred():
    Airline = request.form['airline']
    print(Airline)
    Source = request.form['Source']
    print(Source)
    Destination = request.form['Destination']
    print(Destination)
    Date = request.form['date']
    print(Date)
    Month = request.form['month']
    print(Month)
    Year = request.form['year']
```

```
    print(Year)
    Dep_Time_Hour = request.form['Dep_Time_Hour']
    print(Dep_Time_Hour)
    Dep_Time_Mins = request.form['Dep_Time_Mins']
    print(Dep_Time_Mins)
    Arrival_Time_Hour = request.form['Arrival_Time_Hour']
    print(Arrival_Time_Hour)
    Arrival_TimeMins = request.form['Arrival_Time_Mins']
    print(Arrival_TimeMins)
    Arrival_Day = request.form['Arrival_Day']
    print(Arrival_Day)



    inp_features = [[int(Airline), int(Source), int(Destination) ,int(Date),int(Month), int(Year), int(Dep_Time_Hour),
                     int(Dep_Time_Mins),
                     int(Arrival_Time_Hour),int(Arrival_TimeMins),int(Arrival_Day)]]
    print(inp_features)
    prediction = model.predict(inp_features)
    print(type(prediction))
    t = prediction[0]
    print(t)
    if t > 0.5:
        prediction_text = 'No change in price'
    else:
```

```python
        prediction_text = 'Price will increase'
    print(prediction_text)
    return render_template('prediction.html', prediction_results=prediction_text)


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

HTML CODINGS

File Edit Format View Help

```html
<!DOCTYPE html>
<html>
<head>
<title>Flight Price Prediction</title>
</head>

<body style="background-image: url('https://media.gettyimages.com/id/155150766/photo/passenger-jet-airplane-flying-above-clouds.jpg?s=612x612&w=gi&k=20&c=1BiIwCoCK-XY9smFkY3h4VmqWrJPZdsY-1VtfCwX1Cs='); background-size: cover; background-repeat: no-repeat; background-position: center;"><text='black'>
<h1>
<b>
<i>
<font size=15>
Flight Price Prediction
</font>
</i>
</b>
</h1>
<div style="background-color:white">
<hr>
<hr></div>
<h2> Optimize Flight Booking</h2>
<h4>

<form action="/getdata" method="post">


 <p>  airline : <input type='text' name='airline' placeholder='  ' required='required' /></p>
 <p> Source : <input type='text' name='Source' placeholder='  ' required='required' /></p>
  <p> Destination : <input type='text' name='Destination' placeholder='  ' required='required' /></p>
  <p> Date: <input type='text' name='date' placeholder='  ' required='required' /></p>
  <p>Month : <input type='text' name='month' placeholder='  ' required='required' /></p>
  <p>Year : <input type='text' name='year' placeholder='  ' required='required' /></p>
  <p> Dep_Time_Hour : <input type='text' name='Dep_Time_Hour' placeholder='  ' required='required' /></p>
  <p> Dep_Time_Mins : <input type='text' name='Dep_Time_Mins' placeholder='  ' required='required' /></p>
   <p>  Arrival_Time_Hour : <input type='text' name='Arrival_Time_Hour' placeholder='  ' required='required' /></p>
  <p>  Arrival_Time_Mins : <input type='text' name='Arrival_Time_Mins' placeholder='  ' required='required' /></p>
   <p>  Arrival_Day : <input type='text' name='Arrival_Day' placeholder='  ' required='required' /></p>




  <button type="submit" class="btn btn-primary btn-block btn-large">Predict</button>
</form>
</h4>
<h2>
</h2>
</body>
</html>
```

THANK

YOU!