OPTIMIZING FLIGHT BOOKING DECISIONS THROUGH MACHINE LEARNING PRICE PREDICTIONS

PROJECT BASED

EXPERIENTIAL LEARNING

PROGRAM

TEAM MEMBERS:

- **L.RESHMA**
- **P. MOHAN RAJ**
- **M.PRAVEEN KUMAR**
- **S. NAGESHWARI**

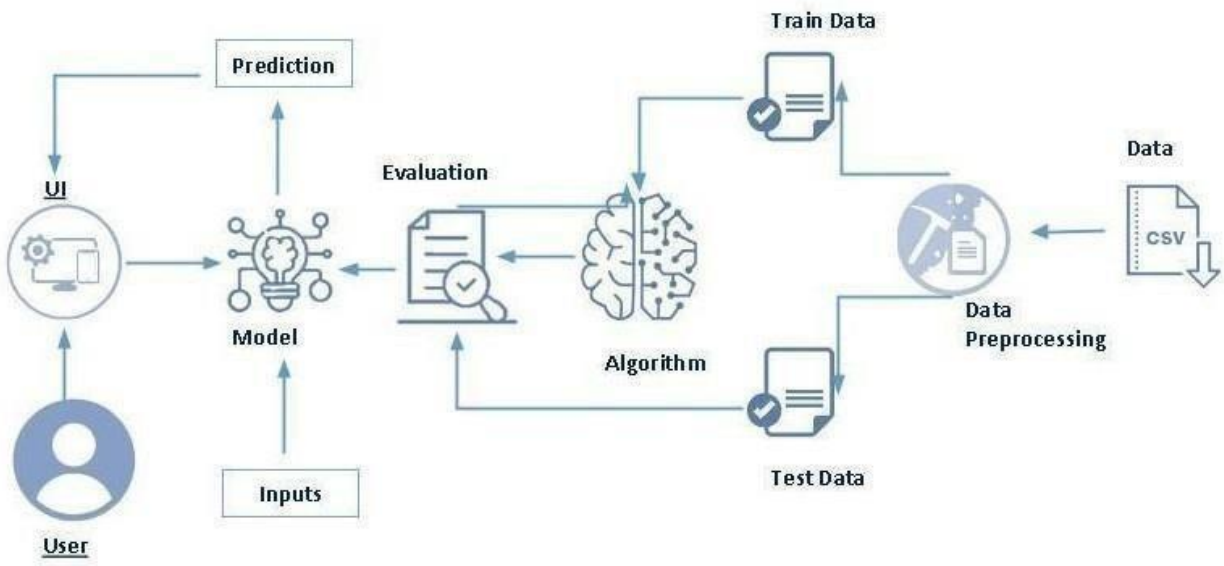
1 INTRODUCTION

1.1 OVERVIEW

Optimizing Flight Booking Decisions through Machine LEARNING Price Predictions

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

Technical Architecture:



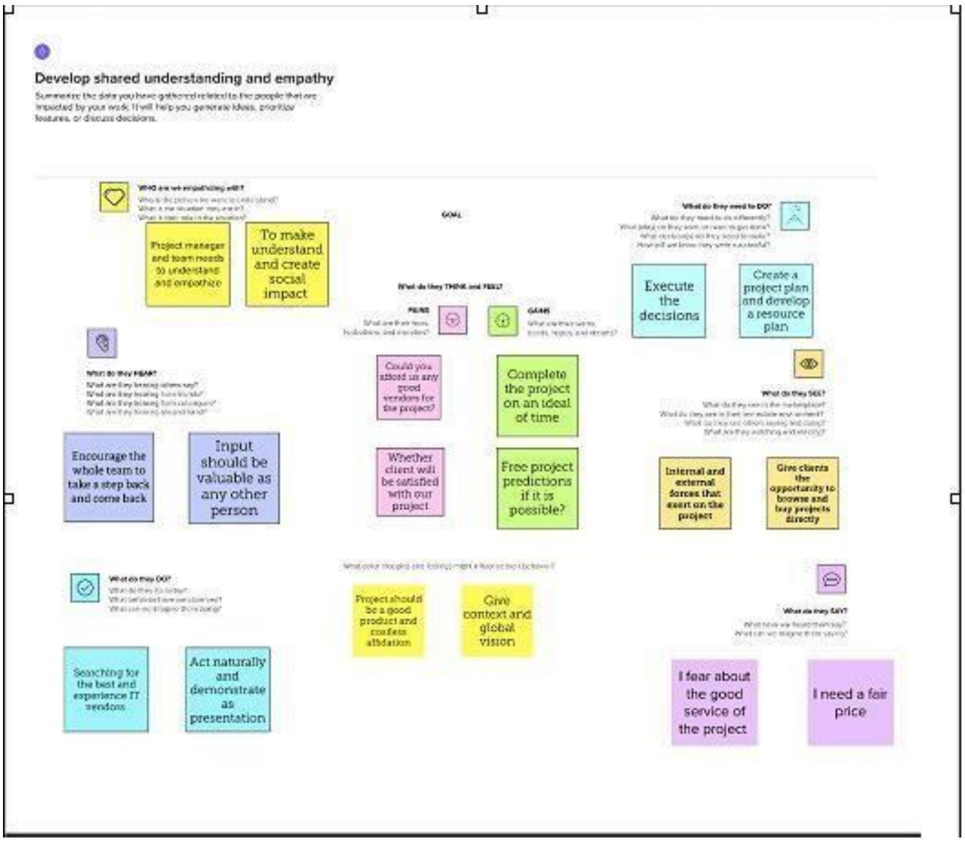
1.2 Purpose

The objective of this article is to predict flight prices given the various parameters. Data used in this article is publicly available at Kaggle. This will be a regression problem since the target or dependent variable is the price (continuous numeric value).

2 PROBLEM DEFINITION & DESIGN THINKING

2.1 Empathy Map Canvas

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



2.2 Ideation & Brainstorming Map

During this activity, our team members gathered and disguised various ideas to solve our projects. Each member contributed six to ten ideas. After gathering all ideas, we assessed the impacts and feasibility of each point.

Finally, we have assigned priority for each point based on these impact values.

Step 1 Team Gathering, Collaboration and Select the Problem Statement

Before you collaborate

A little bit of preparation goes a long way with this session. Here's what you need to do to get going.

10 minutes

- **Team gathering**
  Define who should participate in the session and send an invite. Share relevant information or pre-work ahead.
- **Set the goal**
  Think about the problem you'll be focusing on solving in the brainstorming session.
- **Learn how to use the facilitation tools**
  Use the Facilitation Superpowers to run a happy and productive session.

  Open article →

1 Define your problem statement

What problem are you trying to solve? Frame your problem as a How Might We statement. This will be the focus of your brainstorm.

5 minutes

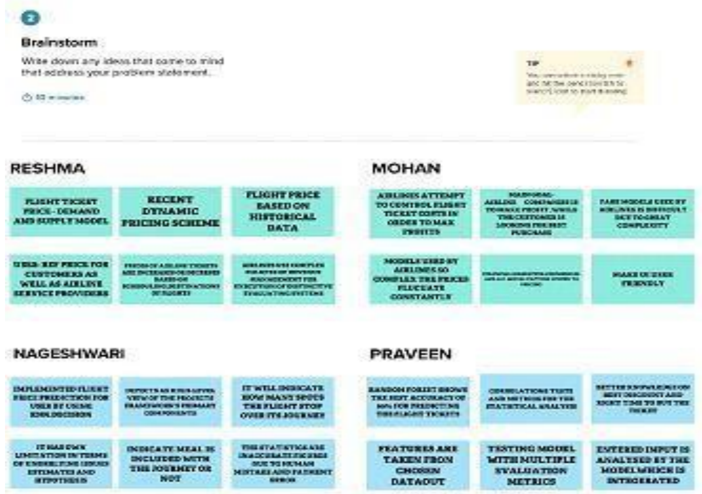
How might we [your problem statement]?

Key rules of brainstorming

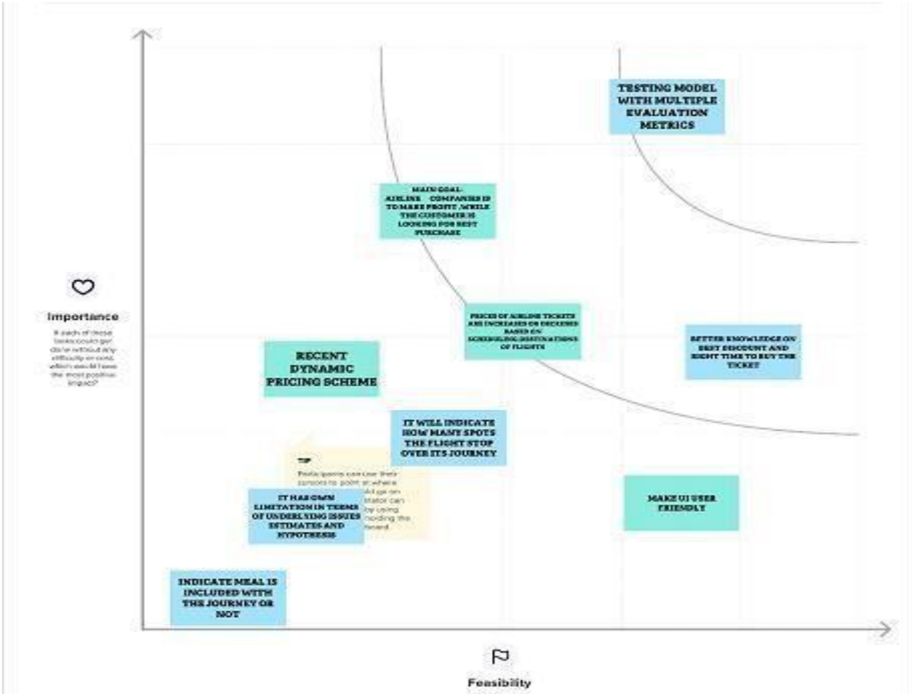
To run a smooth and productive session

- Stay in topic.
- Encourage wild ideas.
- Defer judgment.
- Listen to others.
- Go for volume.
- If possible, be visual.

Step 2 Brainstorm, Idea Listing and Grouping

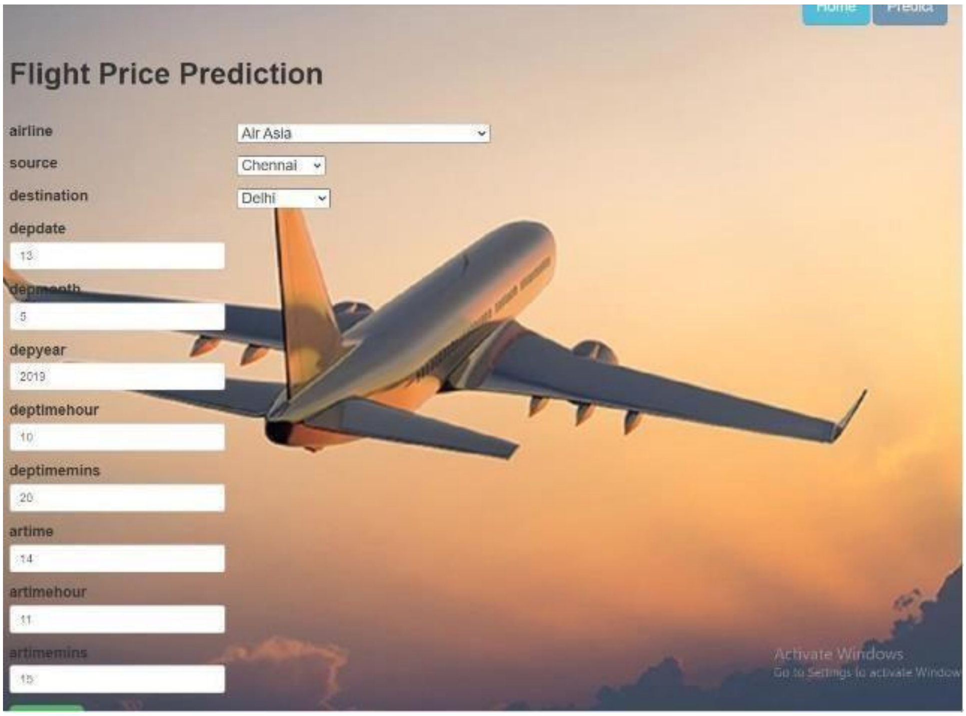


Step 3 Idea Prioritization

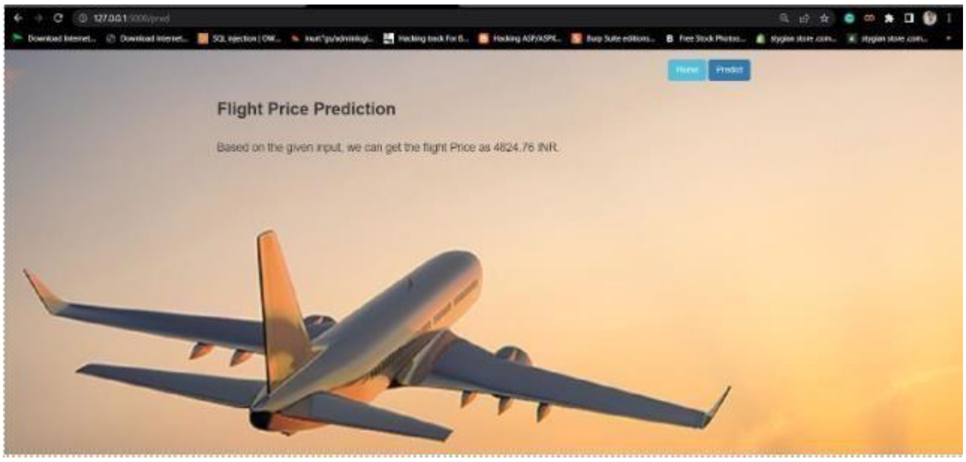


3 RESULT

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



4 Advantages of Flight Booking Decisions through Machine Learning Price Predictions

- Cost Savings: By using machine learning algorithms to predict flight prices, travelers can identify the cheapest time to book their flights. This can result in significant cost savings for both leisure and business travelers.

- Time Savings: Machine learning algorithms can quickly analyze vast amounts of data to predict flight prices, saving travelers the time and effort required to manually research and compare prices across different airlines and travel dates.
- Personalization: Machine learning algorithms can also take into account a traveler's historical booking data, preferences, and travel behavior to provide personalized flight recommendations that are tailored to their specific needs.
- Increased Accuracy: Machine learning algorithms can provide more accurate price predictions than traditional methods of pricing, as they can analyze a wide range of factors that affect airline ticket prices, such as seasonal demand, fuel prices, and competition.
- Better Planning: By predicting flight prices, travelers can plan their trips in advance and avoid last-minute price hikes. This can help reduce the stress and uncertainty associated with travel planning.

Disadvantages of Flight Booking Decisions through Machine Learning Price Predictions

- Data availability: Machine learning models require large amounts of data to make accurate predictions. However, flight data can be complex and difficult to obtain, particularly in regions with limited air traffic.

- Unforeseen events: Flight prices can be significantly impacted by unforeseen events such as natural disasters, geopolitical tensions, and pandemics. These events can be difficult to account for in a machine learning model and may result in inaccurate predictions.

- Limited accuracy: While machine learning models can make accurate predictions based on historical data, there is no guarantee that these predictions will always be correct. There are many factors that can influence flight prices, and it is difficult for a machine learning model to account for all of them.

- Changing pricing models: Airlines can change their pricing models in response to market conditions, which can render a machine learning model obsolete if it is not regularly updated.

- Overreliance on predictions: Relying too heavily on machine learning predictions can lead to missed opportunities or financial losses if the predictions turn out to be inaccurate. It is important to use machine learning predictions in conjunction with other market data and expert analysis.

5 Applications

While flight price prediction through machine learning has several advantages, there are also a few potential disadvantages to consider:

- Data availability: Machine learning models require large amounts of data to make accurate predictions.

However, flight data can be complex and difficult to obtain, particularly in regions with limited air traffic.

- Unforeseen events: Flight prices can be significantly impacted by unforeseen events such as natural disasters, geopolitical tensions, and pandemics. These events can be difficult to account for in a machine learning model and may result in inaccurate predictions.

- Limited accuracy: While machine learning models can make accurate predictions based on historical data, there is no guarantee that these predictions will always be correct. There are many factors that can influence flight prices, and it is difficult for a machine learning model to account for all of them.

- Changing pricing models: Airlines can change their pricing models in response to market conditions, which can render a machine learning model obsolete if it is not regularly updated.

- Overreliance on predictions: Relying too heavily on machine learning predictions can lead to missed opportunities or financial losses if the predictions turn out to be inaccurate.

6 Conclusion

In conclusion, machine learning has proven to be a valuable tool in predicting flight prices. By analyzing historical data and identifying patterns and trends, machine learning algorithms can generate accurate predictions of future flight prices. This can be beneficial for both consumers and airlines, as it can help consumers find the best deals on flights and help airlines optimize their pricing strategies.

However, it's important to note that machine learning models are not perfect, and factors such as unexpected events or changes in market conditions can still impact flight prices. Therefore, while machine learning can provide valuable insights into flight price predictions, it should be used in conjunction with other market data and human expertise to make informed decisions.

7 FUTURE SCOPE

The future scope for flight price prediction through machine learning is vast, as technology continues to advance and more data becomes available. Some potential areas of development include:

- Improved accuracy: As machine learning algorithms become more sophisticated and better at analyzing large amounts of data, we can expect to see even greater accuracy in flight price predictions.

- Real-time price updates: With the availability of real-time data, machine learning algorithms could potentially provide consumers with real-time price updates, allowing them to make more informed decisions about when to book their flights.

- Personalization: Machine learning algorithms could be used to provide personalized flight price predictions based on an individual's past travel history, preferences, and other relevant factors.

- Integration with other travel services: Machine learning could be integrated with other travel services such as hotel bookings and car rentals, to provide a more comprehensive and seamless travel experience.

- Improved transparency: Machine learning algorithms could be used to provide greater transparency in airline pricing, helping consumers understand how prices are determined and making it easier for them to compare prices across different airlines.

8 APPENDIX

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.

```
#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,classificati on_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,Random
ForestRegressor
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.

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

Show 102550100 per page

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 → BOM → COK	09:25	04:25 10 Jun	19h	2 stops	No info	1388
3	IndiGo	12/05/2019	Kolkata	Banglore	CCU → NAG → BLR	18:05	23:30	5h 25m	1 stop	No info	621
4	IndiGo	01/03/2019	Banglore	New Delhi	BLR → NAG → DEL	16:50	21:35	4h 45m	1 stop	No info	1330

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.

```python
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

```
1     [24
      ,
      03,
      201
      9]
2     [1,
      05,
      201
      9]
3     [9,
      06,
      201
      9]
4     [12
      ,
      05,
      201
      9]
5     [01
      ,
      03,
      201
      9]
...
10678    [
         9
         ,
         0
         4
         ,
         2
         0
         1
         9
         ]
10679    [
         2
         7
         ,
```

04,2019]

10680 [27,04,2019]

10681 [01,03,2019]

10682 [9,05,201

```
9
]
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
1    0
2    2
```

```
3      2
4      1
5      1
       ..
10678    0
10679    0
10680    0
10681    0
10682    2
```

Name: Total_Stops, Length: 10682, dtype: object

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

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

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

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

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

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

```
Airline              0
Date_of_Journey      0
Source               0
Destination          0
Route                0
Dep_Time             0
Arrival_Time         0
Duration             0
Total_Stops          0
Additional_Info      0
Price                0
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’.

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

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

categorical = data[column] categorical

index	Airline	Source	Destination	Additional_I	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','Addit ional_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()
```

index	Airline	Source	Destination	Total_ Stops	Additional_ Info	Price	Date	Month	Year	city1	city2	city3	Dep_T ime_H ours	D i
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	

dex	Airline	Source	Destination	Total_ Stops	Additional_ Info	Price	Date	Month	Year	city1	city2	city3	Dep_T ime_H ours	Dep im Mi
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	

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. 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.

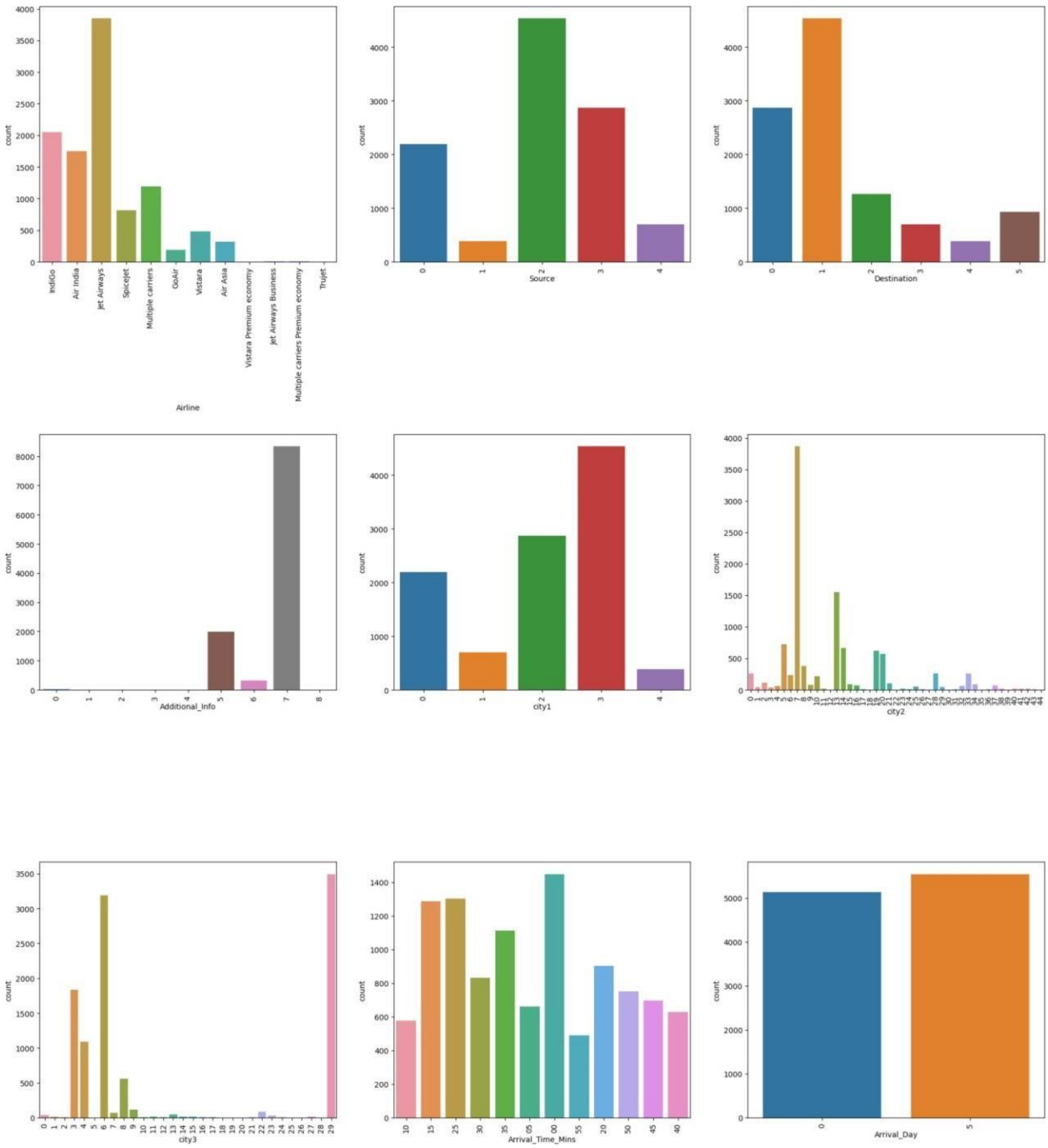
index	Source	Destination	Price	Date	Month
count	10681.0	10681.0	10681.0	10681.0	10
mean	1.952064413444434	1.4360078644321692	9086.443123303061	13.509783728115345	4.70873513715
std	1.177164791209478	1.4748360975189365	4611.075356672832	8.479448759998895	1.164345269867
min	0.0	0.0	1759.0	1.0	
25%	2.0	0.0	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	

categorical = data[column]

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(rotat
ion=90)
plt.tight_layo
ut(pad=3.0)
c=c+1
plt.show()



count
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
Destination
Additional_Info
city1
city2
city3
Arrival_Time_Mins
Arrival_Day

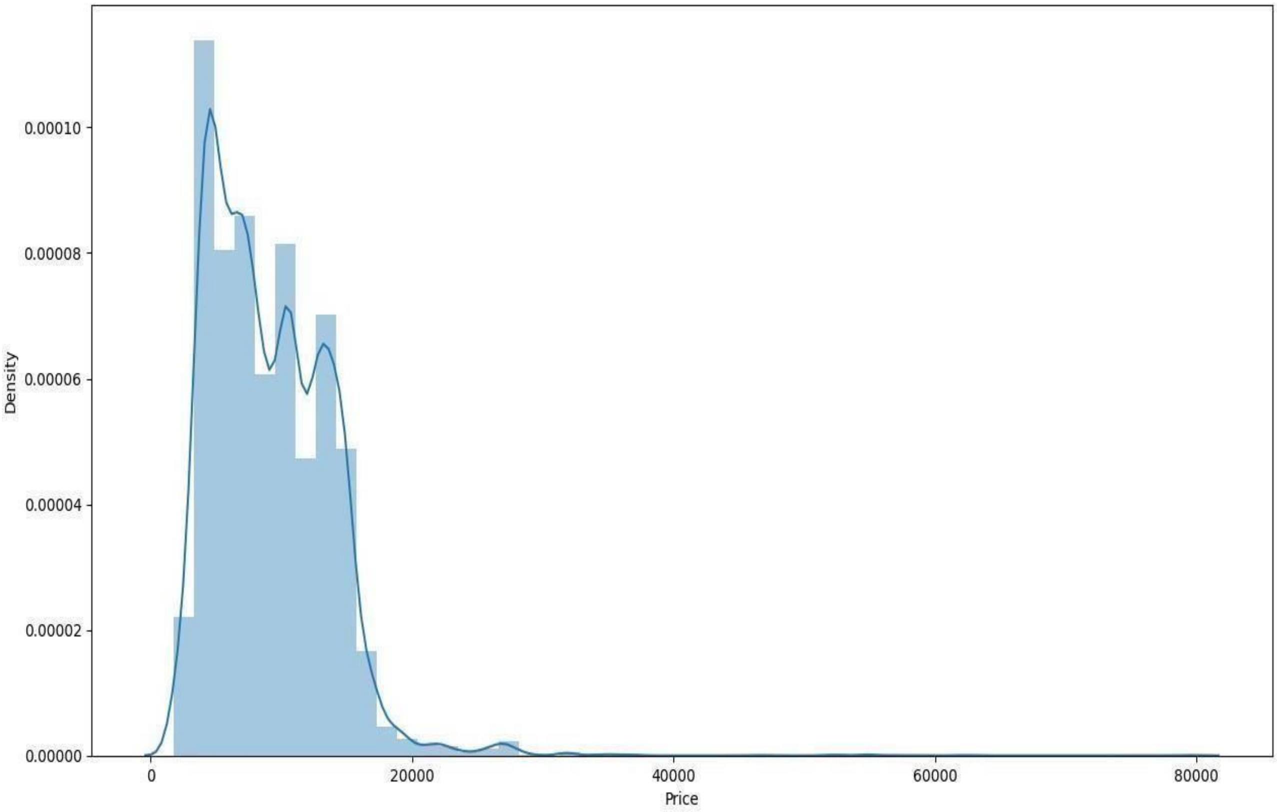
Activity 2.1: We now plot distribution plots to check the distribution in numerical data (Distribution of 'Price' Column)

•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'>

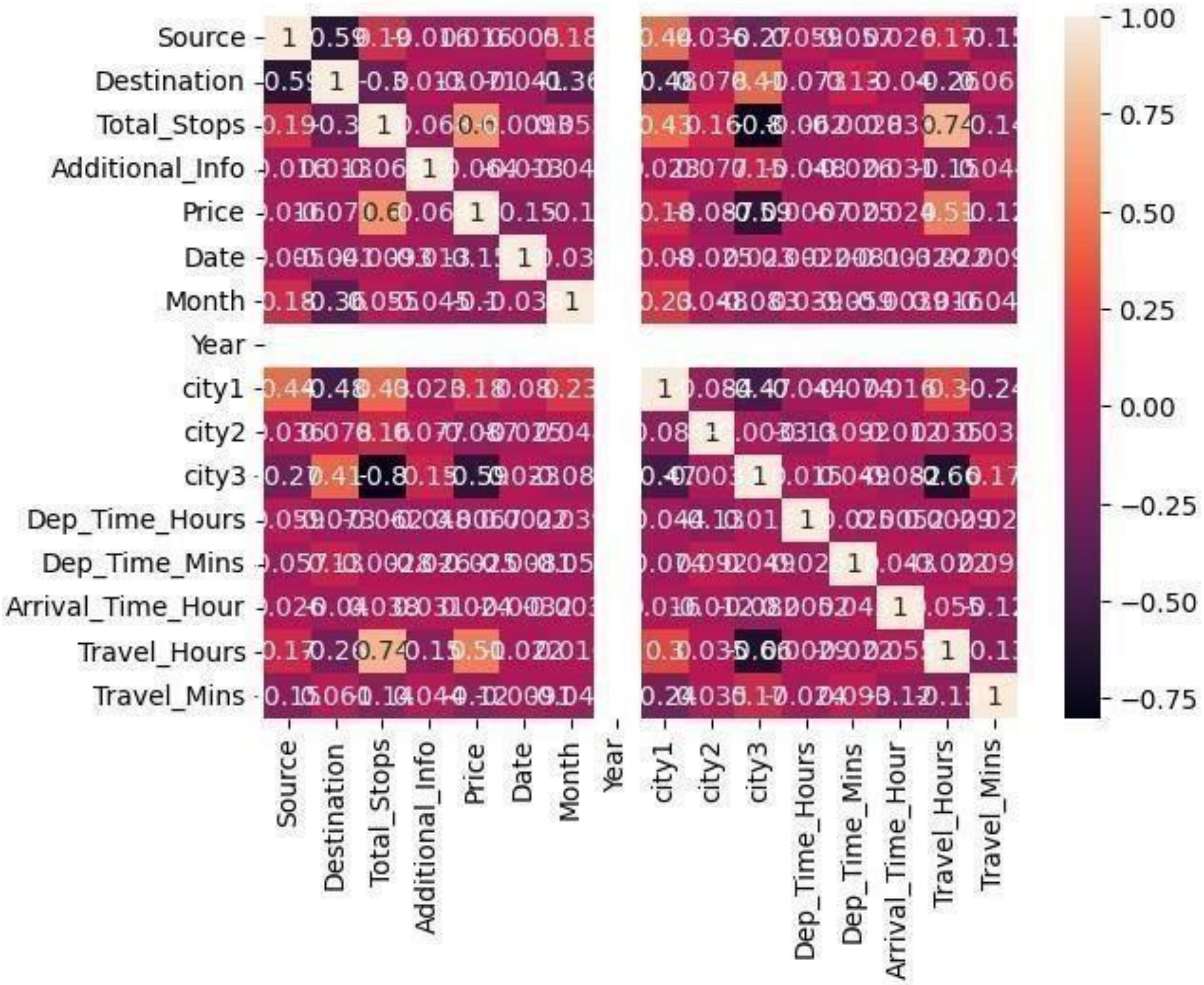


Activity 2.2: Checking the Correlation Using HeatMap

• 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.

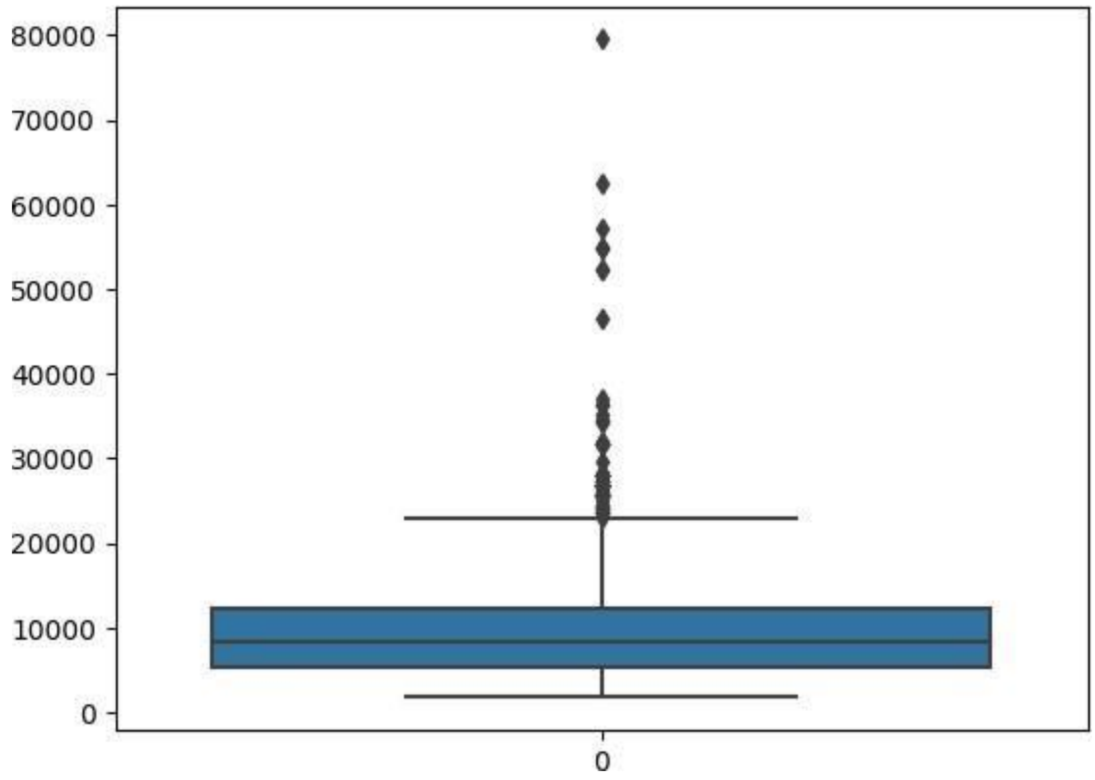
```
sns.heatmap(data.corr(),annot=True)
```



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.

index	Airline	Source	Destination	Date	
0	-0.41093428135292637	-1.6583538810084033	2.4166475116947033	1.2371921421203829	-1.46761
1	-1.2613051152443544	0.8902616302219213	-0.973718431564698	-1.4753753141897006	0.250165
2	0.014251135592787685	0.04072312647847973	-0.2956452429128177	-0.5318735902557585	1.10905
3	-0.41093428135292637	0.8902616302219213	-0.973718431564698	-0.1780604437805302	0.250165
4	-0.41093428135292637	-1.6583538810084033	2.4166475116947033	-1.4753753141897006	-1.46761

- 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()
```

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.

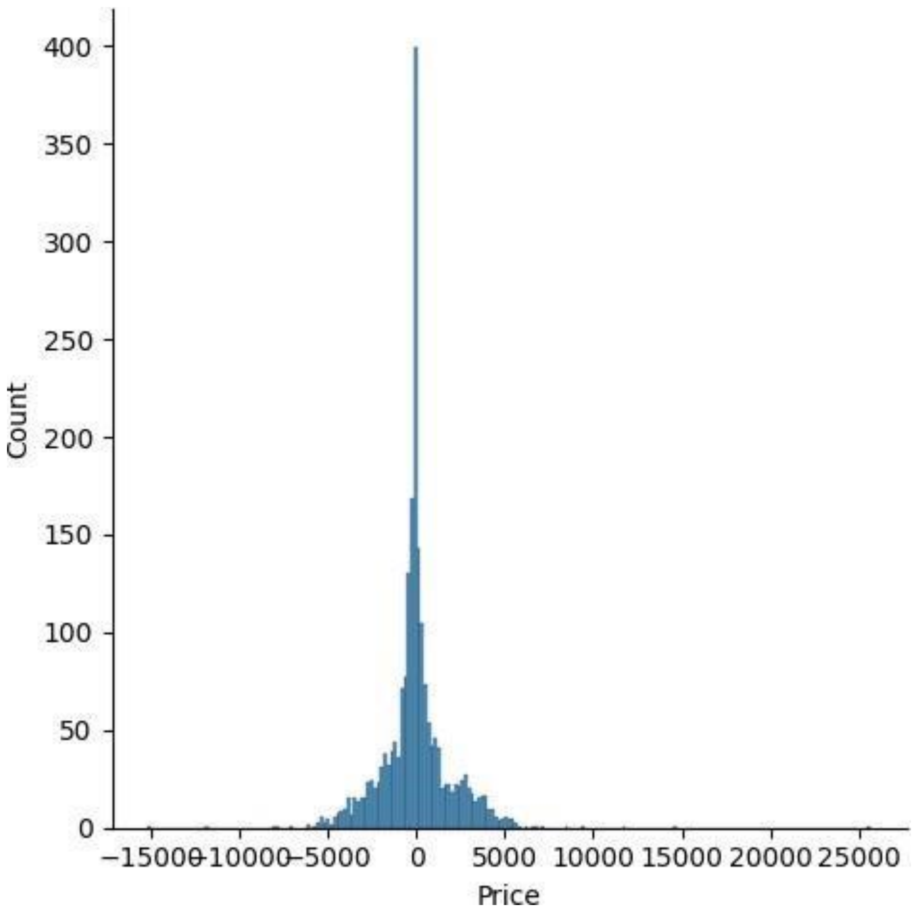
Activity 1: Using Ensemble Techniques

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_squar ed_error
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,prediction
    s)))
    print('MSE:{}'.format(mean_squared_error(y_test,predictions
    ))) print('RMSE:
{}'.format(np.sqrt(mean_squared_error(y_test,predictions))))

        sns.displot(y_testpredictions)

from sklearn.linear_model import LogisticRegression from
sklearn.neighbors import KNeighborsRegressor from sklearn.tree
import DecisionTreeRegressor from sklearn.ensemble import
GradientBoostingRegressor,RandomFore stRegressor
```

```
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,predictio
    ns))))

    sns.disp
    lot(y_te
    st-
    predicti
    ons)
```

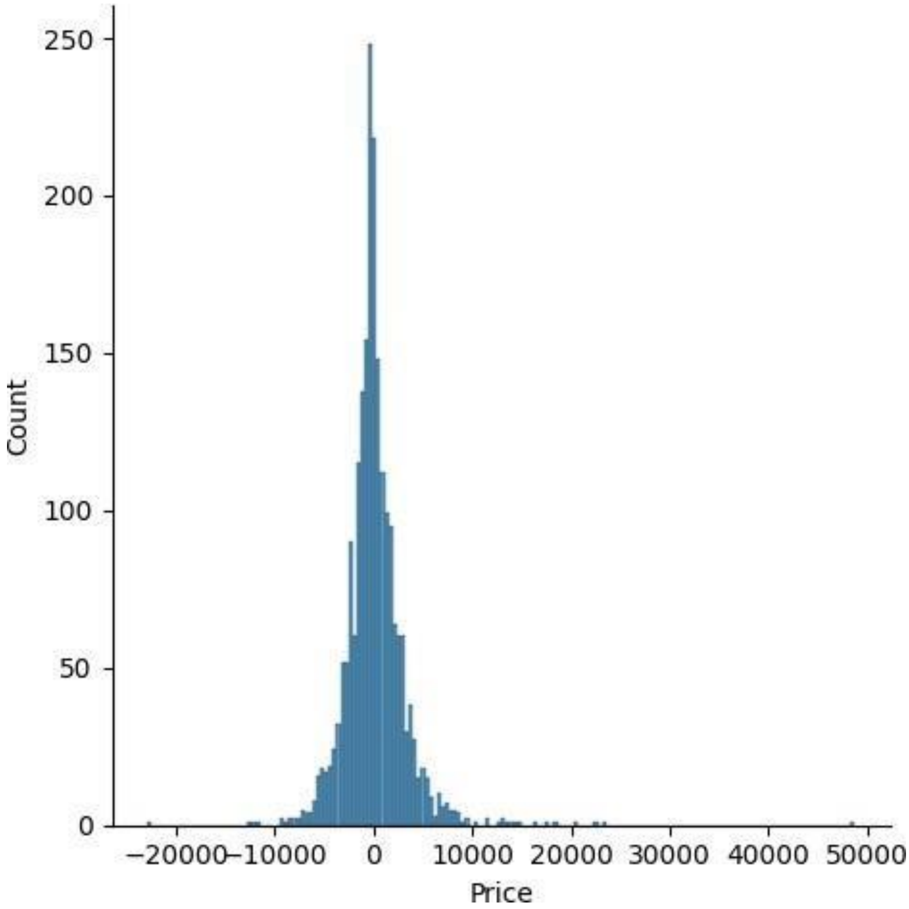
```
predict(RandomForestRegressor())      Model      is:
RandomForestRegressor()
Training score: 0.9520047279248214
    Predictions are: [ 8454.733 13495.64  14811.42  ... 14703.57       5950.2
11696.526]


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]

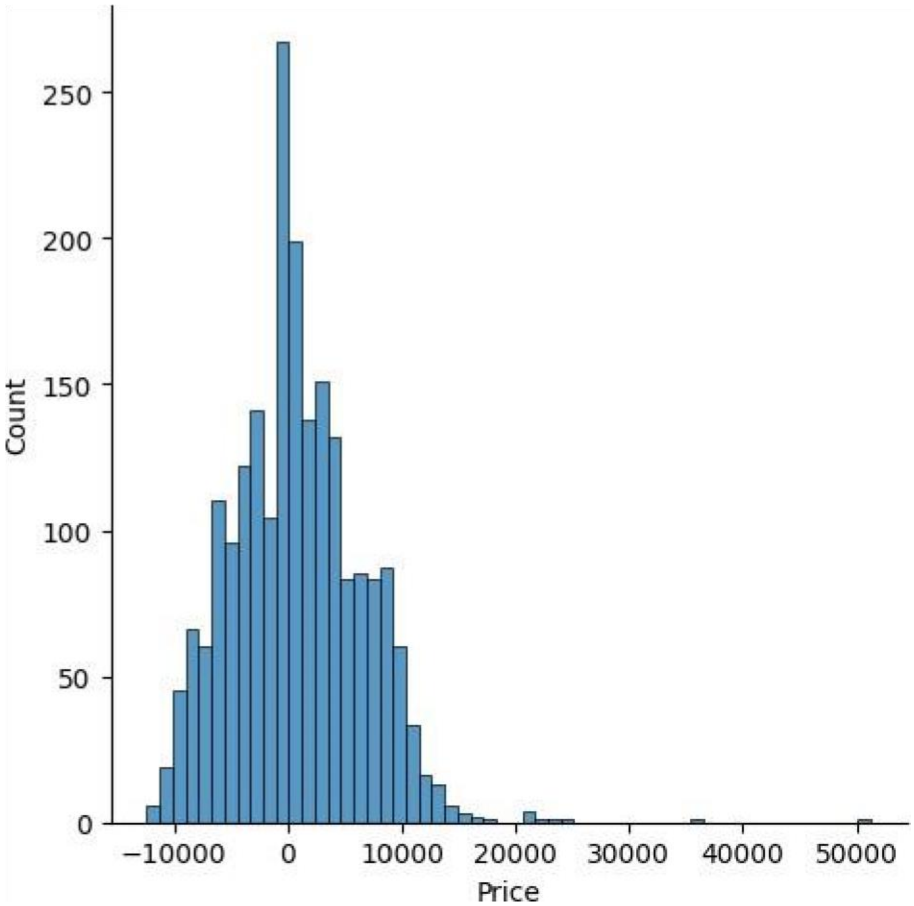
r2 score is: 0.5066647660821886
MAE:1955.9190453907347
MSE:9674509.889021993
RMSE:3110.387417834311



predict(LogisticRegression())

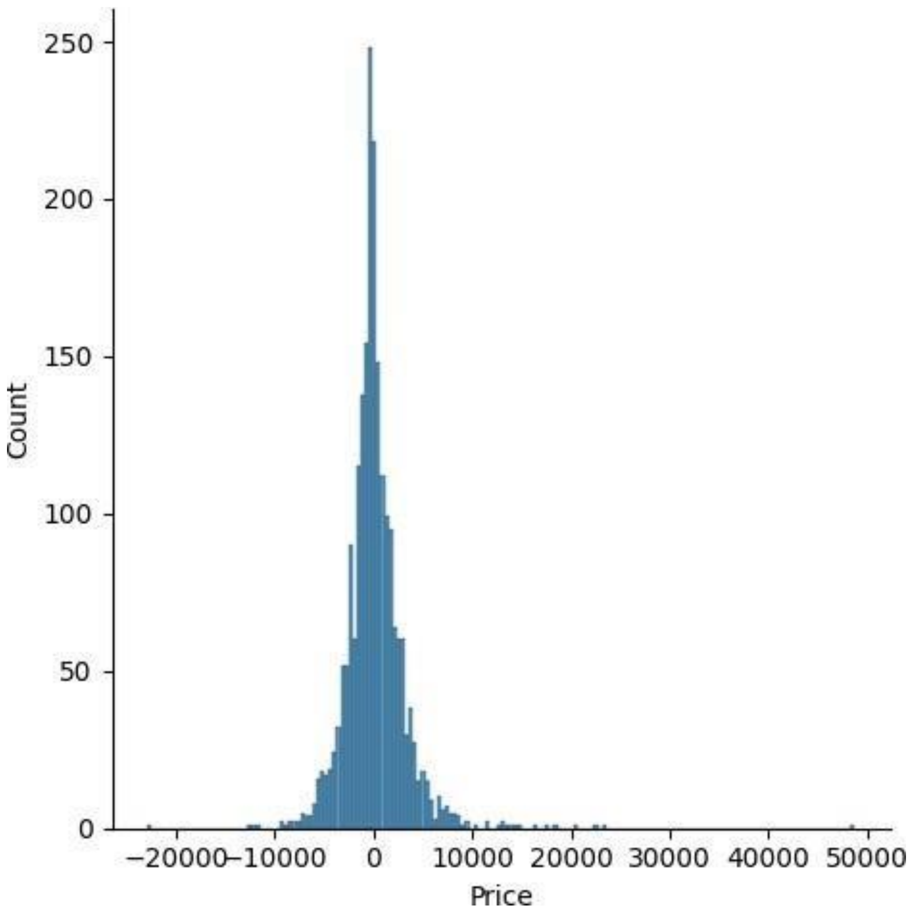
Model is: LogisticRegression()
Training score: 0.04283707865168539
Predictions are: [15129 3943 13941 ... 14714 12898 4174]

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



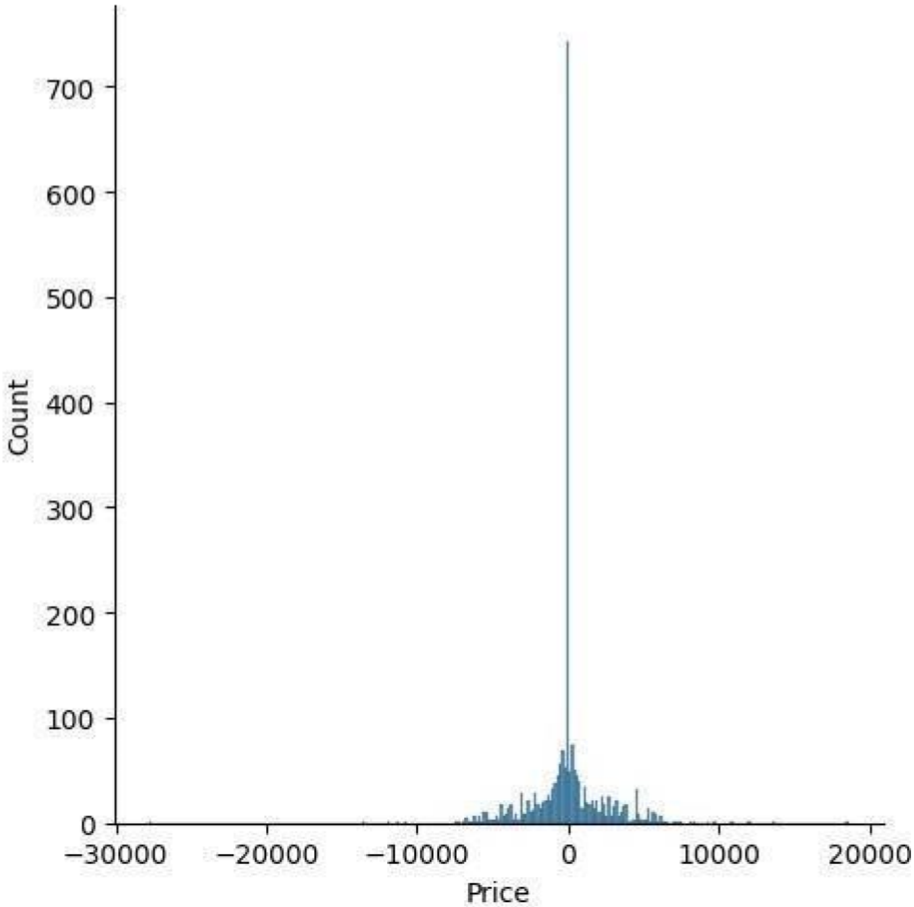
predict(KNeighborsRegressor()) Model is: KNeighborsRegressor()
Training score: 0.7262239247009137
Predictions are: [7629.6 9698.6 12907.8 ... 14293.6 6692.4 5291.8]

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.]

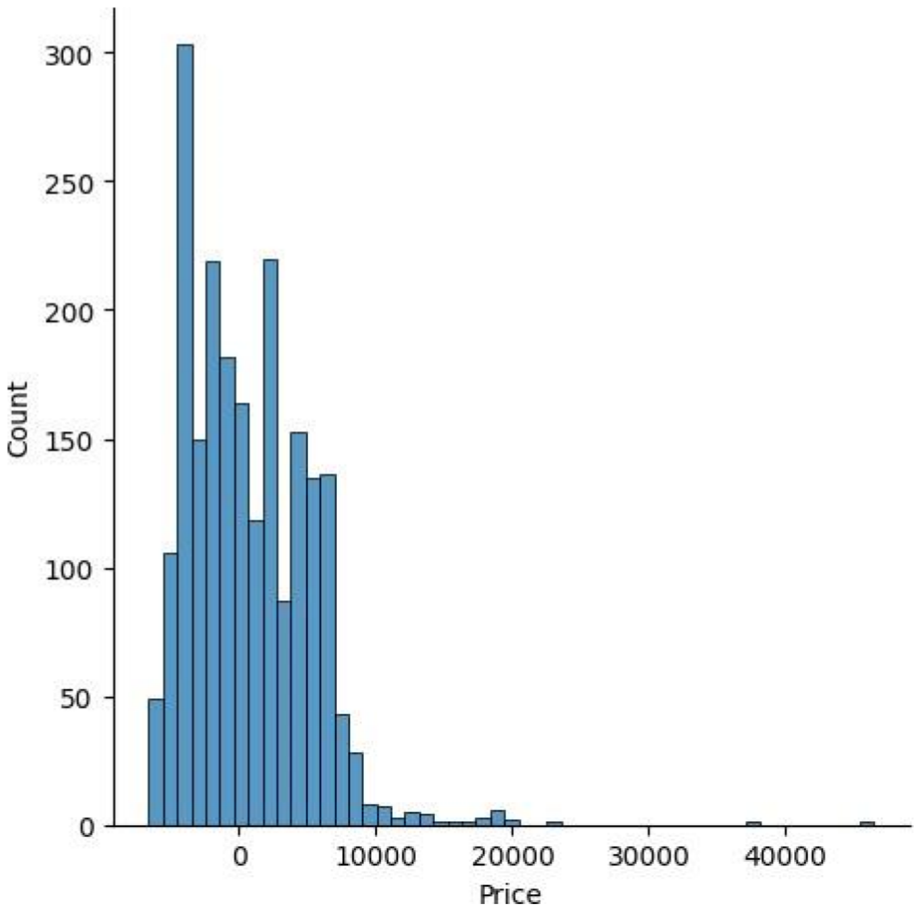
r2 score is: 0.6792060181755581
MAE:1420.1186398377788
MSE:6290903.904942546
RMSE:2508.167439574668



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]

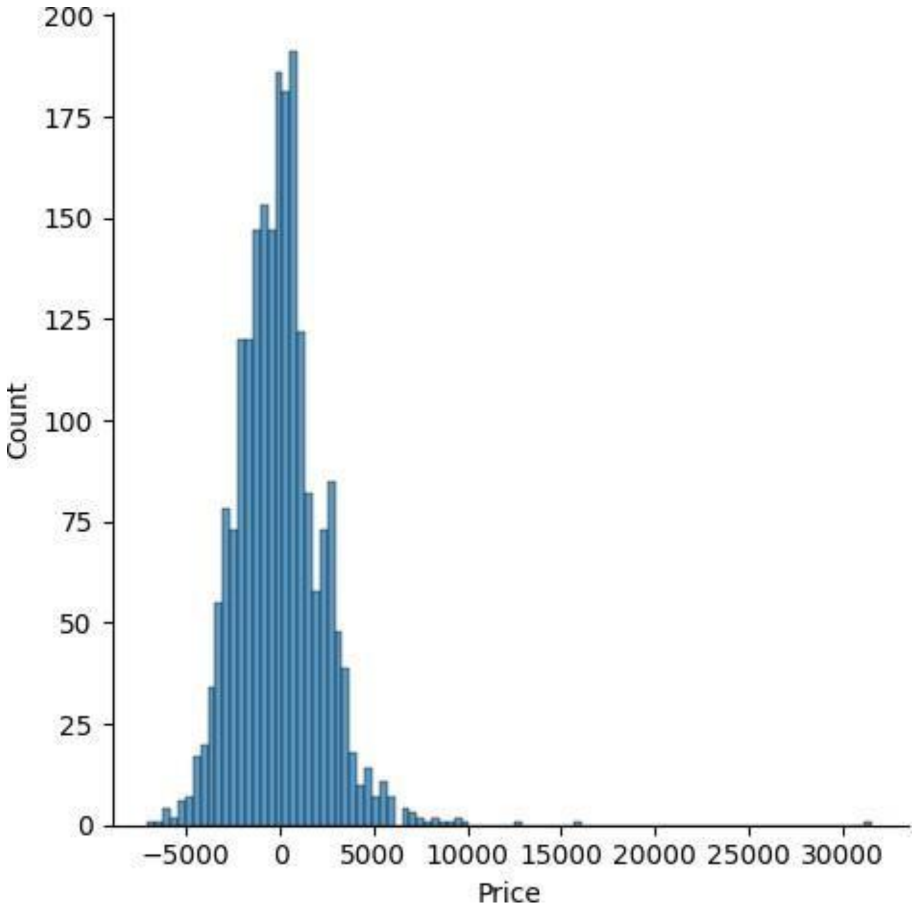
r2 score is: -0.018884049680442 55
MAE:3507.694654077979
MSE:19980741.566174872
RMSE:4469.982278060493



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: -0.018884049680442555
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 kfolds. 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_distributi
ons=ra
ndom_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/crossvalidation-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

```python
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.

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

```python
@app.route("/home")
def home():
```

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



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

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


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

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