OPTIMIZING FLIGHT BOOKING DECISIONS THROUGH MACHINE LEARNING PRICE PREDICTIONS

1. INTRODUCTION:

1.1 OVERVIEW:

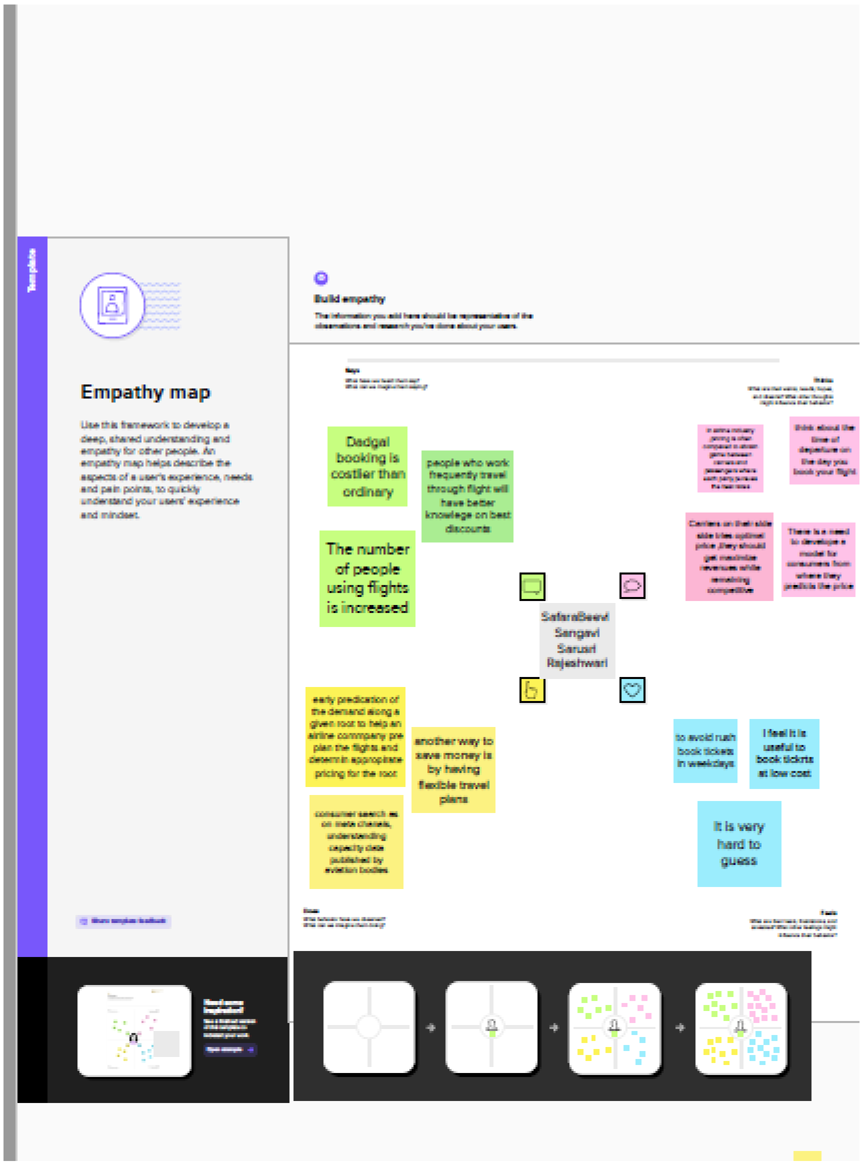
Recently the airline organizations are giving more attention to the complex tactics and processes to finalize the ticket costs in dynamic manner. Also, with the explosive growth of the net and ecommerce, air passengers today will check transportation and availability of any airlines round the world simply. Once satisfying with associate degree of transportation, these customers should buy their desired tickets online through official airline or agent websites to assist the shoppers to shop for the foremost inexpensive transportation, there are variety of prediction models to predict the transportation costs. Social media these days is an integral part of people's daily routines and therefore this resource as a result, is abundant in user opinions. The analysis of some specific opinions will inform corporations on the amount of satisfaction within customers. Airline price ticket costs modification terribly dynamically and for a similar flight day by day. It is terribly tough for a customer to buy an air ticket within the lowest value since the value changes dynamically. We addressed the matter regarding the market section level airfare ticket cost forecasting by usage of publicly obtainable datasets and completely unique machine learning model to forecast market section level price cost of airline ticket. The purpose of this study is to raise and analyze the options that influence transportation.

1.2 PURPOSE:

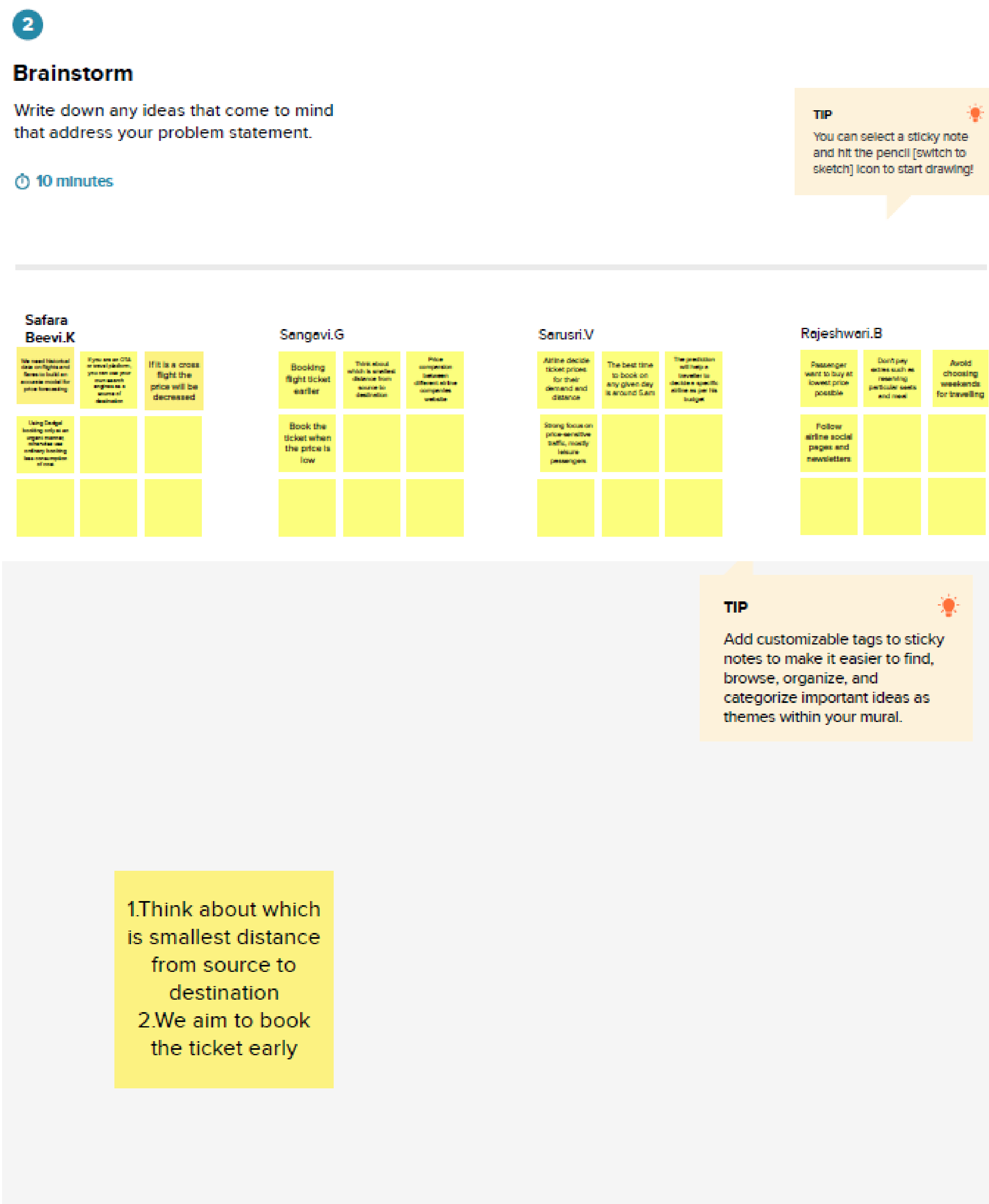
A person who already has reserved a ticket for a flight realizes how powerfully the price of the ticket switches. Airline utilizes progressed techniques considered Revenue Management to accomplish a characteristic esteeming technique. The most affordable ticket available changes over a course of time. The expense of the booking may be far and wide. This esteeming technique normally alters the cost according to the different times in a day namely forenoon, evening, or night. Expenses for the flight may similarly alter according to the different seasons in a year like summers, rainy and winters, also during the period of festivals. The buyers would be looking for the cheapest ticket while the outrageous objective of the transporter would be generating more and more revenue. Travelers for the most part attempt to buy the ticket ahead of their departure day. The reason would be their belief that the prices might be the highest when they would make a booking much nearer to the day of their flight but conventionally this isn't verifiable. The buyer might wrap up paying more than they should for a comparable seat.

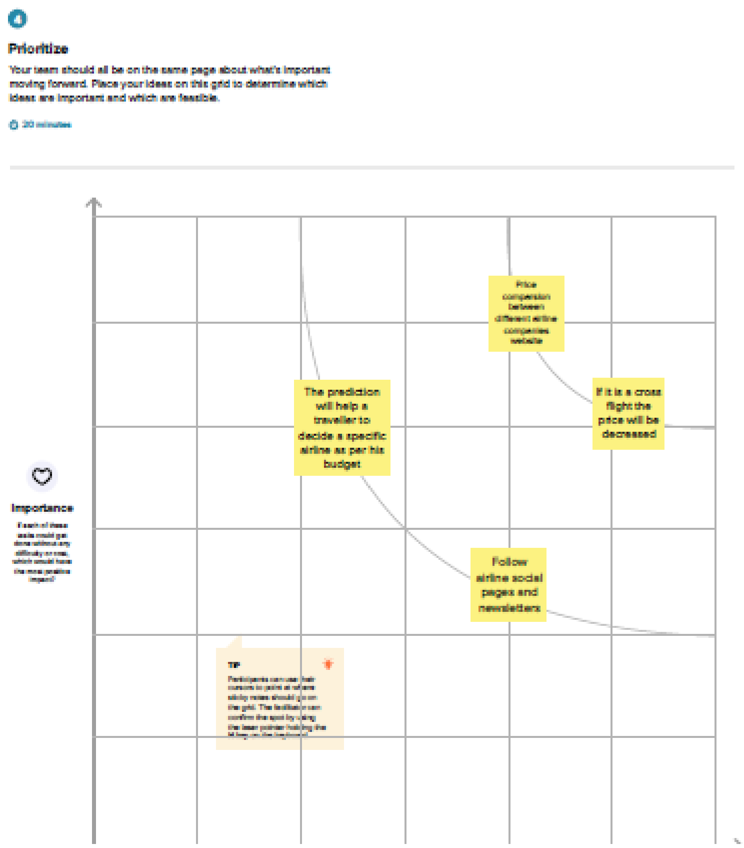
2. PROBLEM DEFINITION & DESIGN THINKING:

2.1 EMPATHY MAP:



2.2 IDEATION & BRAINSTORMING MAP:





3. RESULT:

For the selected test dataset, output of the model is plotted across the test dataset. Graph shows the comparative study of original values and predicted results. By the analysis of the results obtained from the algorithm such as SVM, Decision Tree, KNN, Bagging Tree, Random Forest and Linear regression gives the predicted values of the fare to purchase the flight ticket at the right time. Table I gives the values for R- Square.

The graph is plotted between the days left until departure verses the fare of the flight. The blue color line denotes the actual value of the flight ticket whereas the red color line shows the predicted value of the flight tickets. Decision Tree algorithm has more accuracy compared to other algorithms for the given dataset. The plot between Days remaining for the departure vs. Actual and predicted values evaluated by the Random Algorithm. It gives the highest R-Square value with maximum accuracy in the regression analysis.

4. ADVANTAGES:

The simple task of booking flight tickets has become a science in its own right. People, these days, prefer booking tickets themselves instead of buying from the travel agents in order to save more on their flight tickets. The travelers have a myriad of variables to take into account when plotting air travel – from the distance of journey or the type of services offered – but the other side of the story is that the airlines companies try their best to ensure maximum revenue while ensuring offering best prices on tickets. The airlines companies use the most sophistic software to adjust fares dynamically that consider the performances of its routes and services around the world.

5. DISADVANTAGES:

Reliable internet access is required to check reservations and add bookings that are made over the phone. However, services like can be run on mobile internet connections. Given the industry's transition to online tools, it's a good idea to invest in the best internet service possible for your region.

Choosing an online booking software that doesn't meet your needs can be a real detriment to your business. It's important to do your due diligence upfront. Fortunately, a little bit of research now will save you immeasurable time & frustration in the future.

6. APPLICATIONS:

Airline tickets are important documents that confirm a passenger has a seat on a flight. The ticket includes important information about the passenger and the flight that they will take. The ticket is exchanged for a boarding pass during the check-in process, and this gives the passengers permission to board the plane.

Flight tickets can be purchased in travel class packages, and these packages may vary among companies. Economy and business classes are some of the most common packages.

7. CONCLUSION:

Presently, there are many fields where expectation based administrations are utilized, for example, stock value indicator apparatuses utilized by stock dealers and administration like Z estimate which gives the assessed worth of house costs. In this manner, there is necessity for administration like this in the flight business which can help the clients in booking tickets. There are many investigates works that have been done on this utilizing different procedures and more examination is expected to work on the exactness of the expectation by utilizing various calculations. More precise information with better elements can be likewise be utilized to get more exact outcomes.

8. FUTURE SCOPE:

Later on, our system can be stretched out to incorporate air ticket exchange data, which can give more insight concerning a particular schedule, like time and date of takeoff and appearance, seat area, covered auxiliary items, and so forth By joining such information with the current market fragment and macroeconomic highlights in the current structure, it is feasible to construct an all the more impressive and thorough airfare value forecast model on the day by day or even hourly level. Moreover, airfare cost in a market portion can be impacted by an unexpected inundation of enormous volume of travelers brought about by some exceptional occasions. Accordingly, occasions data will likewise be gathered from different sources, which incorporate social stages and news offices, as to supplement our expectation model. Also, we will explore other progressed ML models, for example, Deep Learning models, while attempting to work on the current models by tuning their hyper-boundaries to arrive at the best engineering for airfare value expectation.

9. APPENDIX:

Milestone 2: Data collection & Preparation

Importing the libraries:

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

Read the dataset

```
data=pd.read_csv('/content/Data_Train.csv')
data.head()
```

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR ? DEL	22:20	01:10 22 Mar	2h 50m	non-stop	No info	3897
1	Air India	1/05/2019	Kolkata	Banglore	CCU ? IXR ? BBI ? BLR	05:50	13:15	7h 25m	2 stops	No info	7662
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL ? LKO ? BOM ? COK	09:25	04:25 10 Jun	19h	2 stops	No info	13882
3	IndiGo	12/05/2019	Kolkata	Banglore	CCU ? NAG ? BLR	18:05	23:30	5h 25m	1 stop	No info	6218
4	IndiGo	01/03/2019	Banglore	New Delhi	BLR ? NAG ? DEL	16:50	21:35	4h 45m	1 stop	No info	13302

Data Preparation

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

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

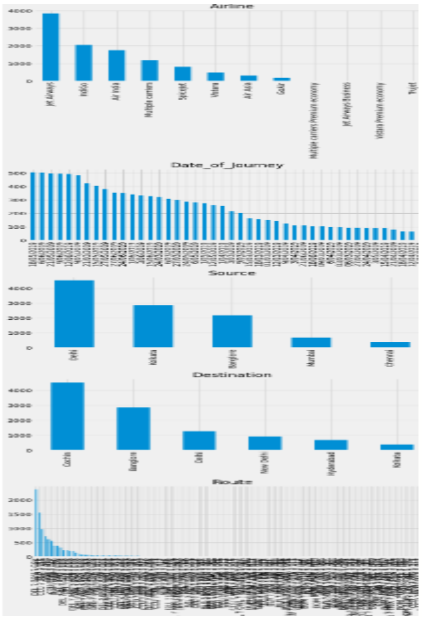
```
for i in category:
    print(i,data[i].unique())
```

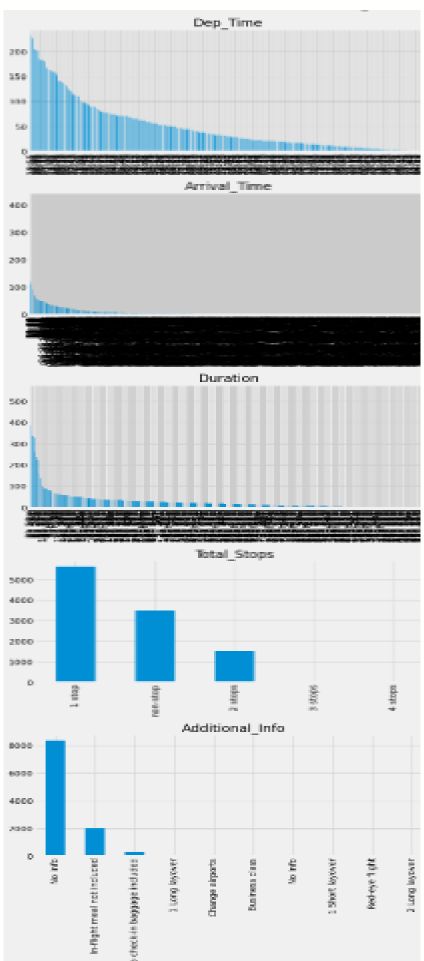
```
Airline ['IndiGo' 'Air India' 'Jet Airways' 'SpiceJet' 'Multiple carriers' 'GoAir'
 'Vistara' 'Air Asia' 'Vistara Premium economy' 'Jet Airways Business'
 'Multiple carriers Premium economy' 'Trujet']
Source ['Banglore' 'Kolkata' 'Delhi' 'Chennai' 'Mumbai']
Destination ['New Delhi' 'Banglore' 'Cochin'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

```
category_cols=data.select_dtypes(include=['object']).columns
category_cols
```

```
Index(['Airline', 'Date_of_Journey', 'Source', 'Destination', 'Route',
       'Dep_Time', 'Arrival_Time', 'Duration', 'Total_Stops',
       'Additional_Info'],
      dtype='object')
```

```
for column in category_cols:
    plt.figure(figsize=(20,4))
    plt.subplot(121)
    data[column].value_counts().plot(kind='bar')
    plt.title(column)
```


Airline
Date_of_Journey
Source
Destination
Route


Dep_Time
Arrival_Time
Duration
Total_Stops
1 stop
non-stop
2 stops
3 stops
4 stops
Additional_Info
No info
In-flight meal not included
No check-in baggage included
1 Long layover
Change airports
Business class
No Info
1 Short layover
Red-eye flight
2 Long layover

```
data.Route=data.Route.str.split('->')
data.Route
```

```
0                       [BLR ? DEL]
1           [CCU ? IXR ? BBI ? BLR]
2           [DEL ? LKO ? BOM ? COK]
3                 [CCU ? NAG ? BLR]
4                 [BLR ? NAG ? DEL]
                        ...
10678                   [CCU ? BLR]
10679                   [CCU ? BLR]
10680                   [BLR ? DEL]
10681                   [BLR ? DEL]
10682       [DEL ? GOI ? BOM ? COK]
Name: Route, Length: 10683, dtype: object
```

```
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]
```

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

```
0         [24, 03, 2019]
1          [1, 05, 2019]
2          [9, 06, 2019]
3         [12, 05, 2019]
4         [01, 03, 2019]
               ...
10678      [9, 04, 2019]
10679     [27, 04, 2019]
10680     [27, 04, 2019]
10681     [01, 03, 2019]
10682      [9, 05, 2019]
Name: Date_of_Journey, Length: 10683, dtype: object
```

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

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

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

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

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

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

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

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]

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

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)

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

```
Airline                  0
Date_of_Journey          0
Source                   0
Destination              0
Route                    1
Dep_Time                 0
Arrival_Time             0
Duration                 0
Total_Stops              1
Additional_Info          0
Price                    0
City1                    1
City2                10683
City3                10683
City4                10683
City5                10683
City6                10683
Date                     0
Month                    0
Year                     0
Dep_Time_Hour            0
Dep_Time_Mins            0
Arrival_date          6348
Time_of_Arrival          0
Arrival_Time_Hour        0
Arrival_Time_Mins        0
Travel_Hours             0
Travel_Mins           1032
dtype: int64
```

```
data.drop(['City4','City5','City6'],axis=1,inplace=True)
data.drop(['Date_of_Journey','Route','Dep_Time','Arrival_Time','Duration']
,axis=1,inplace=True)
data.drop(['Time_of_Arrival'],axis=1,inplace=True)

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

```
Airline                  0
Source                   0
Destination              0
Total_Stops              1
Additional_Info          0
Price                    0
City1                    1
City2                10683
City3                10683
Date                     0
Month                    0
Year                     0
Dep_Time_Hour            0
Dep_Time_Mins            0
Arrival_date          6348
Arrival_Time_Hour        0
Arrival_Time_Mins        0
Travel_Hours             0
Travel_Mins           1032
dtype: int64
```

Replacing missing values

```
data['City3'].fillna('None',inplace=True)

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

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

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

```
Airline                0
Source                 0
Destination            0
Total_Stops            1
Additional_Info        0
Price                  0
City1                  1
City2              10683
City3                  0
Date                   0
Month                  0
Year                   0
Dep_Time_Hour          0
Dep_Time_Mins          0
Arrival_date           0
Arrival_Time_Hour      0
Arrival_Time_Mins      0
Travel_Hours           0
Travel_Mins            0
dtype: int64
```

```
data.Date=data.Date.astype('int64')
data.Month=data.Month.astype('int64')
data.Year=data.Year.astype('int64')
data.Dep_Time_Hour=data.Dep_Time_Hour.astype('int64')
data.Dep_Time_Hour=data.Dep_Time_Hour.astype('int64')
data.Dep_Time_Mins=data.Dep_Time_Mins.astype('int64')
data.Arrival_date=data.Arrival_date.astype('int64')
data.Arrival_Time_Hour=data.Arrival_Time_Hour.astype('int64')
data.Arrival_Time_Mins=data.Arrival_Time_Mins.astype('int64')
data.Travel_Mins=data.Travel_Mins.astype('int64')

data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10683 entries, 0 to 10682
Data columns (total 19 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Airline            10683 non-null  object 
 1   Source             10683 non-null  object 
 2   Destination        10683 non-null  object 
 3   Total_Stops        10682 non-null  object 
 4   Additional_Info    10683 non-null  object 
 5   Price              10683 non-null  int64  
 6   City1              10682 non-null  object 
 7   City2              0 non-null      float64
 8   City3              10683 non-null  object 
 9   Date               10683 non-null  int64  
 10  Month              10683 non-null  int64  
 11  Year               10683 non-null  int64  
 12  Dep_Time_Hour      10683 non-null  int64  
 13  Dep_Time_Mins      10683 non-null  int64  
 14  Arrival_date       10683 non-null  int64  
 15  Arrival_Time_Hour  10683 non-null  int64  
 16  Arrival_Time_Mins  10683 non-null  int64  
 17  Travel_Hours       10683 non-null  object 
 18  Travel_Mins        10683 non-null  int64  
dtypes: float64(1), int64(10), object(8)
memory usage: 1.5+ MB
```

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

	Airline	Source	Destination	Total_Stops	Additional_Info	Price	City1	City2	City3	Date	Month	Year	Dep_Time_Hour	Dep_Time_Mins	Arrival_date	Arrival_Time_Hour	Arrival_Time_Mins	Tra
6474	Air India	Mumbai	Hyderabad	2	No info	17327	BOM ? GOI ? PNQ ? HYD	NaN	None	6	3	2019	16	50	6	16	55	

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

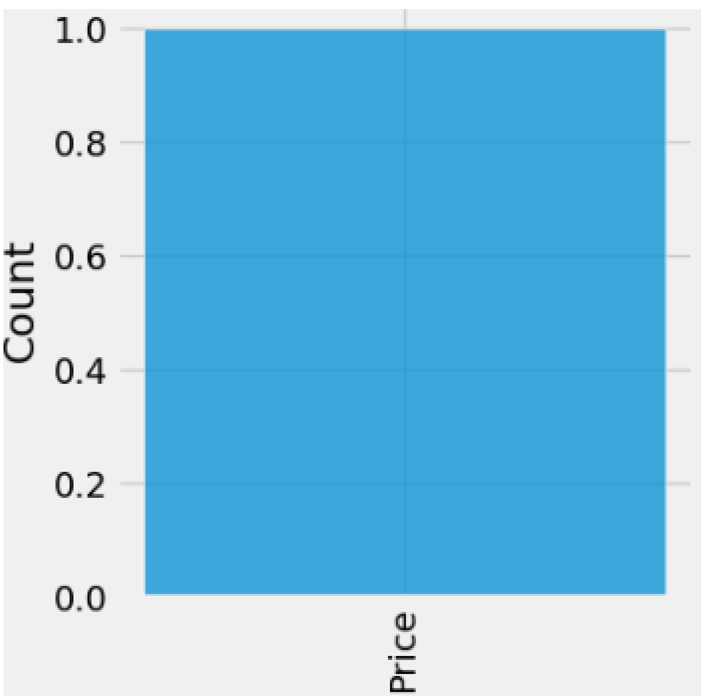
```
data.Travel_Hours=data.Travel_Hours.astype('int64')
```

```
categorical=['Airline','Source','Destination','Additional_Info','City1']
numerical=['Total_Stops','Date','Month','Year','Dep_Time_Hour','Dep_Time_M
ins','Arrival_Time_Mins','Travel_Hours','Travel_Mins']
```

Exploratory data analysis:

Visual analysis:

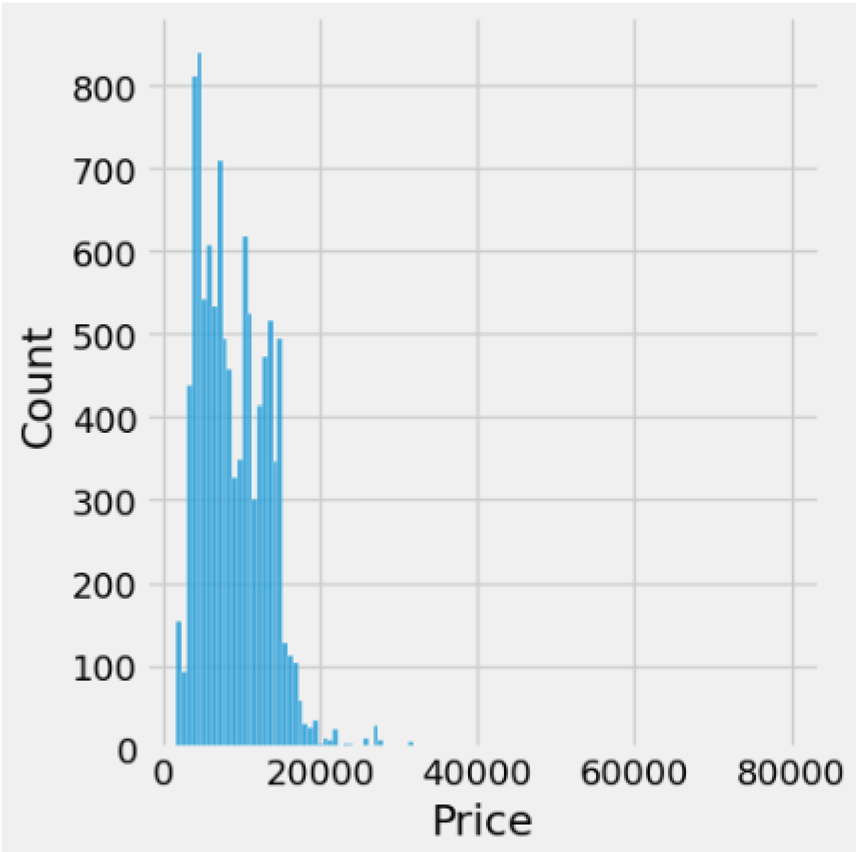
```
import seaborn as sns
c=1

plt.figure(figsize=(20,45))
for i in categorical:
    plt.subplot(6,3,c)
    sns.displot('Price')
    plt.xticks(rotation=90)
    plt.tight_layout(pad=3.0)
    c=c+1

plt.show()
```



Plot distribution to check the distribution in numerical data:

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

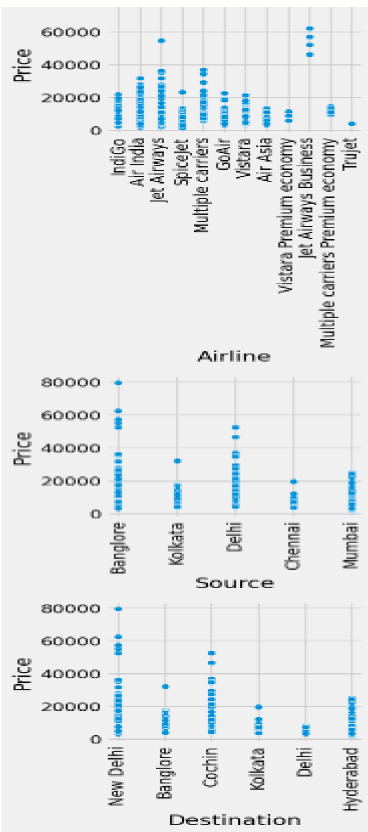
```
<seaborn.axisgrid.FacetGrid at 0x7f52483e42b0>
<Figure size 1500x800 with 0 Axes>
```

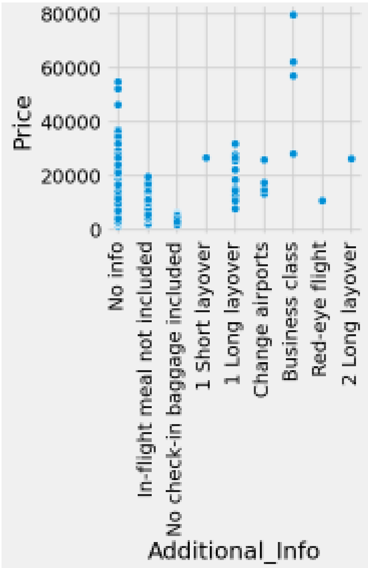


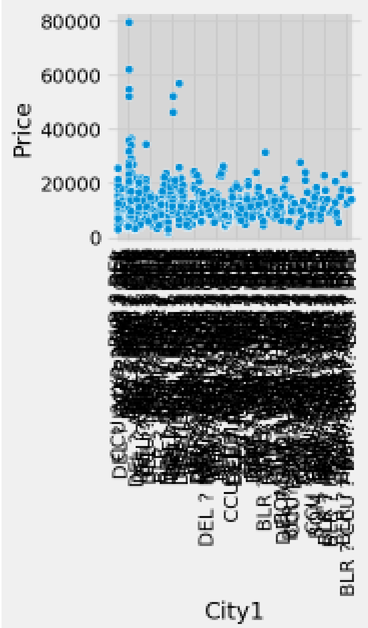
```
data.columns
```

```
Index(['Airline', 'Source', 'Destination', 'Total_Stops', 'Additional_Info',
       'Price', 'City1', 'City2', 'City3', 'Date', 'Month', 'Year',
       'Dep_Time_Hour', 'Dep_Time_Mins', 'Arrival_date', 'Arrival_Time_Hour',
       'Arrival_Time_Mins', 'Travel_Hours', 'Travel_Mins'],
      dtype='object')
```

```
import seaborn as sns
c=1

for i in categorical:
    plt.figure(figsize = (10,20))

    plt.subplot(6,3,c)

    sns.scatterplot(x=data[i],y=data.Price)
    plt.xticks(rotation=90)
    #plt.tight_layout(pad=3.0)
    c=c+1
    plt.show()
```








```
data[data.Price>50000]
data.head()
pd.set_option('display.max_columns',25)
data.head()
```

	Airline	Source	Destination	Total_Stops	Additional_Info	Price	City1	City2	City3	Date	Month	Year	Dep_Time_Hour	Dep_Time_Mins	Arrival_date	Arrival_Time_Hour	Arrival_Time_Mins	Travel_Hours	Travel_Mins
0	IndiGo	Banglore	New Delhi	non-stop	No info	3897	BLR ? DEL	NaN	None	24	3	2019	22	20	22	1	10	2	50
1	Air India	Kolkata	Banglore	2	No info	7662	CCU ? IXR ? BBI ? BLR	NaN	None	1	5	2019	5	50	1	13	15	7	25
2	Jet Airways	Delhi	Cochin	2	No info	13882	DEL ? LKO ? BOM ? COK	NaN	None	9	6	2019	9	25	10	4	25	19	0
3	IndiGo	Kolkata	Banglore	1	No info	6218	CCU ? NAG ? BLR	NaN	None	12	5	2019	18	5	12	23	30	5	25
4	IndiGo	Banglore	New Delhi	1	No info	13302	BLR ? NAG ? DEL	NaN	None	1	3	2019	16	50	1	21	35	4	45

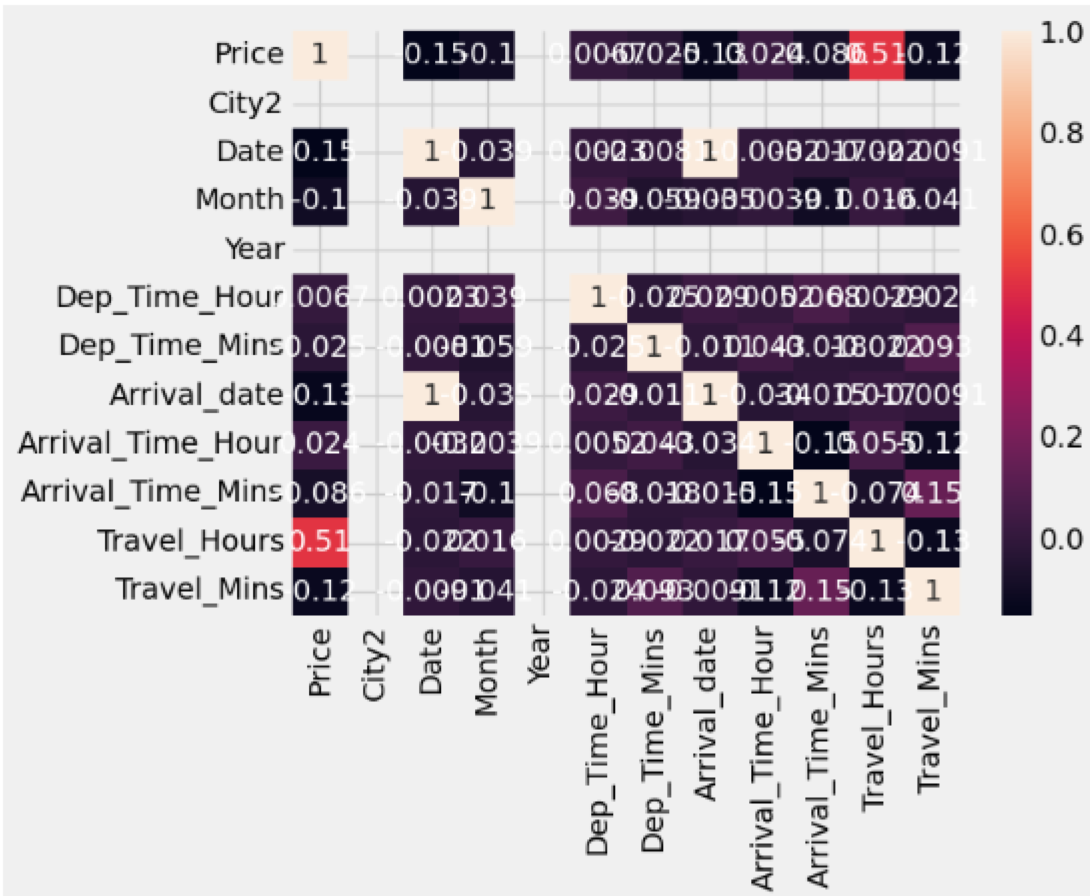
```
data['Year'].max()
```

```
2019
```

Checking the correlation using heatmap:

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

```
<Axes: >
```



```
data.info()
```

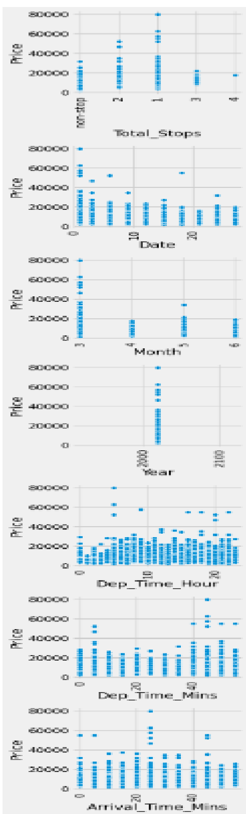
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10682 entries, 0 to 10682
Data columns (total 19 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Airline            10682 non-null  object 
 1   Source             10682 non-null  object 
 2   Destination        10682 non-null  object 
 3   Total_Stops        10681 non-null  object 
 4   Additional_Info    10682 non-null  object 
 5   Price              10682 non-null  int64  
 6   City1              10681 non-null  object 
 7   City2              0 non-null      float64
 8   City3              10682 non-null  object 
 9   Date               10682 non-null  int64  
 10  Month              10682 non-null  int64  
 11  Year               10682 non-null  int64  
 12  Dep_Time_Hour      10682 non-null  int64  
 13  Dep_Time_Mins      10682 non-null  int64  
 14  Arrival_date       10682 non-null  int64  
 15  Arrival_Time_Hour  10682 non-null  int64  
 16  Arrival_Time_Mins  10682 non-null  int64  
 17  Travel_Hours       10682 non-null  int64  
 18  Travel_Mins        10682 non-null  int64  
dtypes: float64(1), int64(11), object(7)
memory usage: 1.6+ MB
```

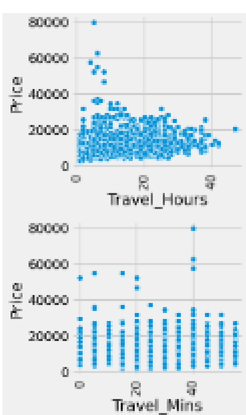
```
data
```

	Airline	Source	Destination	Total_Stops	Additional_Info	Price	City1	City2	City3	Date	Month	Year	Dep_Time_Hour	Dep_Time_Mins	Arrival_date	Arrival_Time_Hour	Arrival_Time_Mins	Travel_Hours	Travel_Mins
0	IndiGo	Banglore	New Delhi	non-stop	No info	3897	BLR ? DEL	NaN	None	24	3	2019	22	20	22	1	10	2	50
1	Air India	Kolkata	Banglore	2	No info	7662	CCU ? IXR ? BBI ? BLR	NaN	None	1	5	2019	5	50	1	13	15	7	25
2	Jet Airways	Delhi	Cochin	2	No info	13882	DEL ? LKO ? BOM ? COK	NaN	None	9	6	2019	9	25	10	4	25	19	0
3	IndiGo	Kolkata	Banglore	1	No info	6218	CCU ? NAG ? BLR	NaN	None	12	5	2019	18	5	12	23	30	5	25
4	IndiGo	Banglore	New Delhi	1	No info	13302	BLR ? NAG ? DEL	NaN	None	1	3	2019	16	50	1	21	35	4	45
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
10678	Air Asia	Kolkata	Banglore	non-stop	No info	4107	CCU ? BLR	NaN	None	9	4	2019	19	55	9	22	25	2	30
10679	Air India	Kolkata	Banglore	non-stop	No info	4145	CCU ? BLR	NaN	None	27	4	2019	20	45	27	23	20	2	35
10680	Jet Airways	Banglore	Delhi	non-stop	No info	7229	BLR ? DEL	NaN	None	27	4	2019	8	20	27	11	20	3	0
10681	Vistara	Banglore	New Delhi	non-stop	No info	12648	BLR ? DEL	NaN	None	1	3	2019	11	30	1	14	10	2	40
10682	Air India	Delhi	Cochin	2	No info	11753	DEL ? GOI ? BOM ? COK	NaN	None	9	5	2019	10	55	9	19	15	8	20

10682 rows × 19 columns

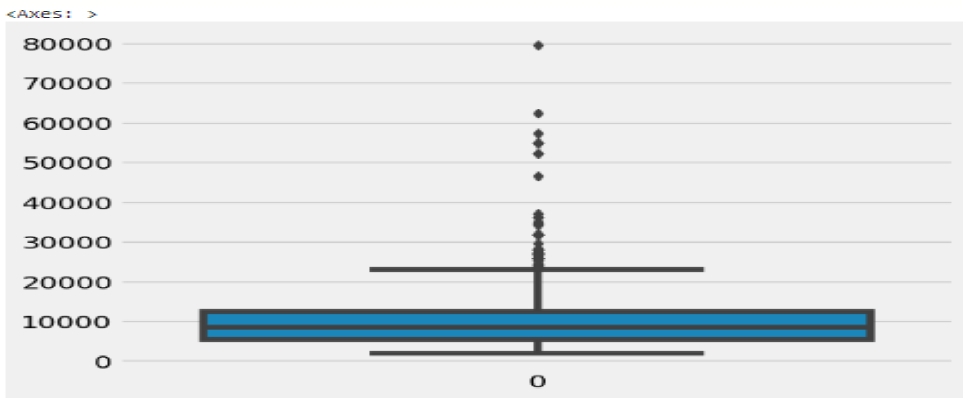
```python
c=1

for i in numerical:
  plt.figure(figsize=(10,20))
  plt.subplot(6,3,c)
  sns.scatterplot(x = data[i], y=data.Price)
  plt.xticks(rotation=90)
  #plt.tight_layout(pad=3.0)
  c=c+1
  plt.show()
```





Outlier detection for 'Price' columns:

```python
import seaborn as sns
sns.boxplot(data['Price'])
```




```python
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.Total_Stops=le.fit_transform(data.Total_Stops)
data.City1=le.fit_transform(data.City1)
data.City2=le.fit_transform(data.City2)
data.City3=le.fit_transform(data.City3)
data.Additional_Info=le.fit_transform(data.Additional_Info)
data.head()
```

	Airline	Source	Destination	Total_Stops	Additional_Info	Price	City1	City2	City3	Date	Month	Year	Dep_Time_Hour	Dep_Time_Mins	Arrival_date	Arrival_Time_Hour	Arrival_Time_Mins	Travel_Hours	Travel_Mins
0	3	0	5	4	7	3897	18	0	0	24	3	2019	22	20	22	1	10	2	50
1	1	3	0	1	7	7662	84	0	0	1	5	2019	5	50	1	13	15	7	25
2	4	2	1	1	7	13882	118	0	0	9	6	2019	9	25	10	4	25	19	0
3	3	3	0	0	7	6218	91	0	0	12	5	2019	18	5	12	23	30	5	25
4	3	0	5	0	7	13302	29	0	0	1	3	2019	16	50	1	21	35	4	45

```python
data.head()
```

	Airline	Source	Destination	Total_Stops	Additional_Info	Price	City1	City2	City3	Date	Month	Year	Dep_Time_Hour	Dep_Time_Mins	Arrival_date	Arrival_Time_Hour	Arrival_Time_Mins	Travel_Hours	Travel_Mins
0	3	0	5	4	7	3897	18	0	0	24	3	2019	22	20	22	1	10	2	50
1	1	3	0	1	7	7662	84	0	0	1	5	2019	5	50	1	13	15	7	25
2	4	2	1	1	7	13882	118	0	0	9	6	2019	9	25	10	4	25	19	0
3	3	3	0	0	7	6218	91	0	0	12	5	2019	18	5	12	23	30	5	25
4	3	0	5	0	7	13302	29	0	0	1	3	2019	16	50	1	21	35	4	45

```
data = data[['Airline','Source','Destination','Date','Month','Year','Dep_T
ime_Hour','Dep_Time_Mins','Arrival_date','Arrival_Time_Hour','Arrival_Time
_Mins','Price']]

data.head()
```

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

Scaling the data:

```
from sklearn.preprocessing import StandardScaler
ss=StandardScaler()


data1 = ss.fit_transform(data)
```

```
data1 = pd.DataFrame(data1,columns=data.columns)
data.head()
```

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

```
y = data1['Price']
x = data1.drop(columns=['Price'],axis=1)
```

Splitting the data into train and test:

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

```
x_train.head()
```

	Airline	Source	Destination	Date	Month	Year	Dep_Time_Hour	Dep_Time_Mins	Arrival_date	Arrival_Time_Hour	Arrival_Time_Mins
10004	0.864716	0.040721	-0.29563	1.591104	0.250153	0.0	-0.781129	0.297937	1.546321	0.823940	-0.587017
3684	0.014369	0.040721	-0.29563	-0.531796	0.250153	0.0	-0.259258	0.297937	-0.461621	-0.196605	0.624852
1034	1.715063	0.040721	-0.29563	1.237288	-0.608777	0.0	0.436570	1.097240	1.191978	1.261317	-1.192952
3909	0.864716	0.040721	-0.29563	0.883471	-1.467707	0.0	-0.085301	1.363674	0.955750	-1.800319	0.624852
3088	-1.261152	0.040721	-0.29563	1.237288	1.109082	0.0	0.784483	-0.501367	1.310092	0.823940	-0.587017

```
x_train.shape
```

```
(8545, 11)
```

Model Building

Using ensemble techniques:

```
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegres
sor, AdaBoostRegressor
```

```
rfr=RandomForestRegressor()
gb=GradientBoostingRegressor()
ad=AdaBoostRegressor()
```

```
from sklearn.metrics import r2_score,mean_absolute_error,mean_squared_erro
r

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

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

```
RandomForestRegressor()
R2 score is 0.8519520591893709
R2 for train data 0.9497597215143748
Mean Absolute Error is 0.25295968202767016
Mean Squared Error is 0.1472174215284212
Root Mean Squared Error is 0.38368922519197907
GradientBoostingRegressor()
R2 score is 0.7652955778741217
R2 for train data 0.7338510043179753
Mean Absolute Error is 0.364940138086792
Mean Squared Error is 0.23338777734765506
Root Mean Squared Error is 0.48310224316148137
AdaBoostRegressor()
R2 score is 0.15074658319703382
R2 for train data 0.14424297207423298
Mean Absolute Error is 0.7630438227023815
Mean Squared Error is 0.8444892753074892
Root Mean Squared Error is 0.9189609759437498
```

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Dense, Activation, Dropout
from tensorflow.keras.optimizers import Adam

model=keras.Sequential()
model.add(Dense(7,activation ='relu',input_dim=11))

model.add(Dense(7,activation='relu'))

model.add(Dense(1,activation='linear'))

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 7)                 84

 dense_4 (Dense)             (None, 7)                 56

 dense_5 (Dense)             (None, 1)                 8

=================================================================
Total params: 148
Trainable params: 148
Non-trainable params: 0
_________________________________________________________________
```

```python
model.compile(loss = 'mse', optimizer = 'rmsprop',metrics = ['mae'])

model.fit(x_train, y_train, batch_size = 20, epochs = 10)
```

```
Epoch 1/10
428/428 [==============================] - 1s 2ms/step - loss: 1.1009 - mae: 0.8037
Epoch 2/10
428/428 [==============================] - 1s 2ms/step - loss: 0.8595 - mae: 0.7138
Epoch 3/10
428/428 [==============================] - 1s 2ms/step - loss: 0.7723 - mae: 0.6689
Epoch 4/10
428/428 [==============================] - 1s 2ms/step - loss: 0.7171 - mae: 0.6377
Epoch 5/10
428/428 [==============================] - 1s 2ms/step - loss: 0.6828 - mae: 0.6195
Epoch 6/10
428/428 [==============================] - 1s 2ms/step - loss: 0.6612 - mae: 0.6072
Epoch 7/10
428/428 [==============================] - 1s 3ms/step - loss: 0.6436 - mae: 0.5989
Epoch 8/10
428/428 [==============================] - 1s 3ms/step - loss: 0.6289 - mae: 0.5911
Epoch 9/10
428/428 [==============================] - 1s 2ms/step - loss: 0.6194 - mae: 0.5852
Epoch 10/10
428/428 [==============================] - 1s 2ms/step - loss: 0.6098 - mae: 0.5807
<keras.callbacks.History at 0x7f5246bcf700>
```

```python
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() 0.7880290883663809
RandomForestRegressor() 0.7902280513773019
RandomForestRegressor() 0.8012364592314392
```

```python
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
▸ estimator: RandomForestRegressor
  ▸ RandomForestRegressor
```

```
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
estimator: GradientBoostingRegressor
GradientBoostingRegressor
```

```
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 0.9186119083927532
test accuracy 0.7785052329121148
```

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

```
RandomForestRegressor(max_features='sqrt', n_estimators=10) 0.7261268905747631
RandomForestRegressor(max_features='sqrt', n_estimators=10) 0.7292504145204649
RandomForestRegressor(max_features='sqrt', n_estimators=10) 0.7277272322018159
```

```
gb=GradientBoostingRegressor(n_estimators=10,max_features='sqrt',max_depth=None)
gb.fit(x_train,y_train)
y_train_pred=gb.predict(x_train)
y_test_pred=gb.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 0.6364865173543217
test accuracy 0.18074819829450883
```

Regression model:

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

from sklearn.metrics import r2_score,mean_absolute_error,mean_squared_erro
r

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

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

```
KNeighborsRegressor()
R2 score is 0.7337698733752689
R2 score for train data 0.7878038246704271
Mean Absolute Error is 0.35917333406078
Mean Squared Error is 0.26473662896136735
Root Mean Squared Error is 0.5145256348923417
SVR()
R2 score is 0.6248128034087579
R2 score for train data 0.5957444387377182
Mean Absolute Error is 0.4162888458787714
Mean Squared Error is 0.3730824715981053
Root Mean Squared Error is 0.6108047737191526
```

```
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 0.8797607060998262
test accuracy 0.7013502693959782
```

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

```
KNeighborsRegressor(n_jobs=-1, n_neighbors=2) 0.6301907440142309
KNeighborsRegressor(n_jobs=-1, n_neighbors=2) 0.6458609920294707
KNeighborsRegressor(n_jobs=-1, n_neighbors=2) 0.6646580886084823
```

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

```
predicted_values
```

	Actual	Predicted
6075	1.641563	1.681688
3544	-0.895161	-0.895161
9290	0.021842	-0.217169
5032	-1.133955	-1.190563
2483	0.826714	1.516636
...	...	...
9796	-0.364002	0.976150
9870	-0.968253	-0.614942
10062	-0.354459	-0.636414
8802	-0.439479	-0.466156
8617	1.007382	1.104331

2137 rows × 2 columns

```
prices=rfr.predict(x_test)
```

Evaluating performance of the model and saving the model:

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

price_list
```

	Price
0	1.068566
1	-0.583796
2	-0.027109
3	-1.138250
4	1.413260
...	...
2132	0.811749
2133	-0.887288
2134	-0.564233
2135	0.030952
2136	0.711200

2137 rows × 1 columns

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