Milestone 2

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, GradientBoostingClassifier
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/Copy of Data_Train.csv")
data.head()
```

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Dur
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR ? DEL	22:20	01:10 22 Mar	2
1	Air India	1/05/2019	Kolkata	Banglore	CCU ? IXR ? BBI ? BLR	05:50	13:15	7

```
data.shape
```

```
(10683, 11)
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10683 entries, 0 to 10682
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   Airline          10683 non-null  object
 1   Date_of_Journey  10683 non-null  object
 2   Source           10683 non-null  object
 3   Destination      10683 non-null  object
 4   Route            10682 non-null  object
 5   Dep_Time         10683 non-null  object
 6   Arrival_Time     10683 non-null  object
 7   Duration         10683 non-null  object
 8   Total_Stops      10682 non-null  object
 9   Additional_Info  10683 non-null  object
 10  Price            10683 non-null  int64 
dtypes: int64(1), object(10)
memory usage: 918.2+ KB
```

```
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
dtype: int64
```

Data Preparation

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

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

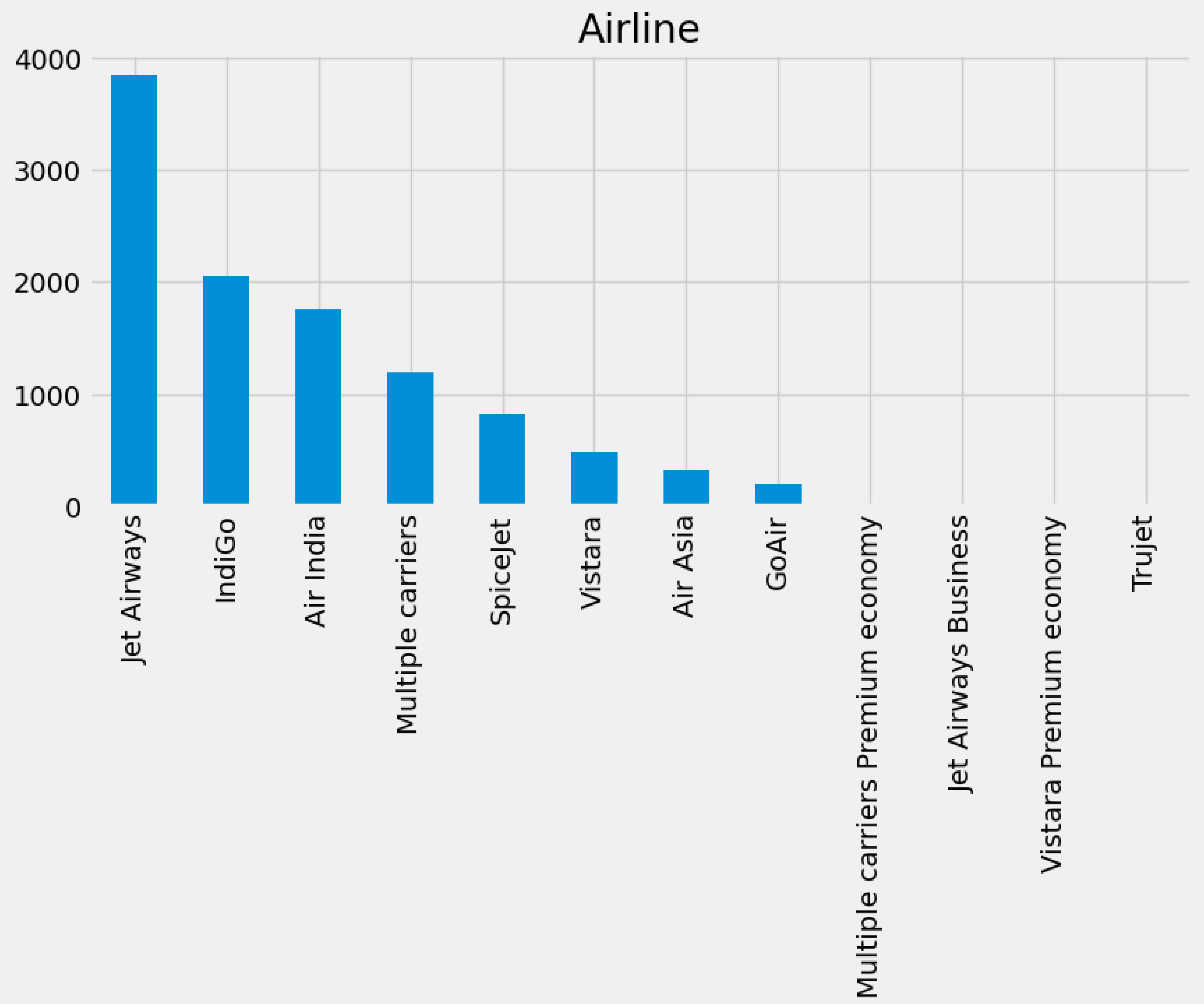
```
for i in category:
    print(i, data[i].unique())
```

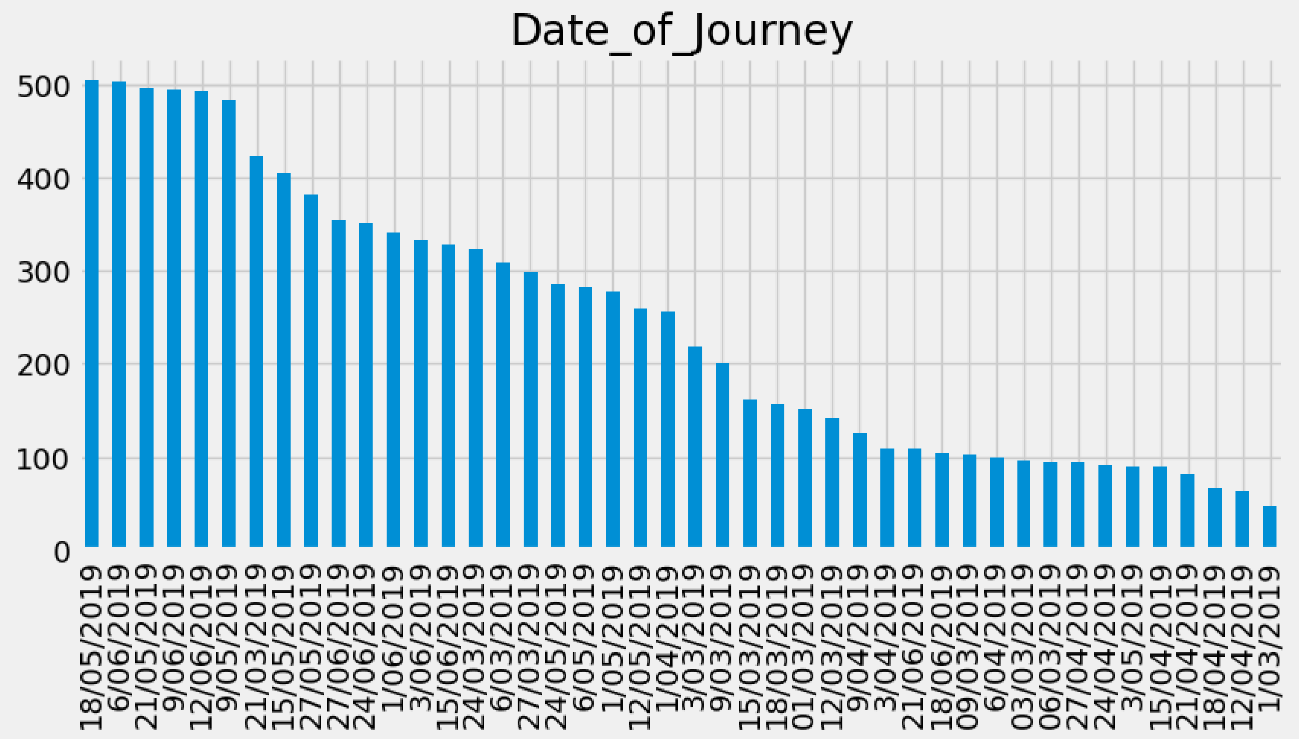
```
Airline ['IndiGo' 'Air India' 'Jet Airways' 'SpiceJet' 'Multiple carriers' 'GoAir'
 'Vistara' 'Air Asia' 'Vistara Premium economy' 'Jet Airways Business'
 'Multiple carriers Premium economy' 'Trujet']
Source ['Banglore' 'Kolkata' 'Delhi' 'Chennai' 'Mumbai']
Destination ['New Delhi' 'Banglore' 'Cochin'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

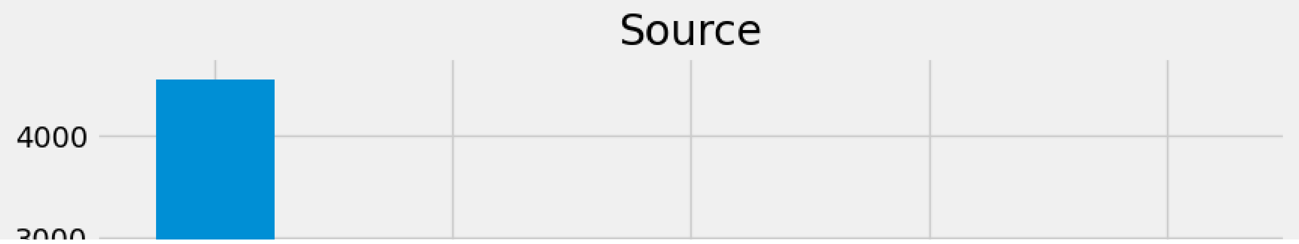
```
category_cols=data.select_dtypes(include=['object']).columns
category_cols
```

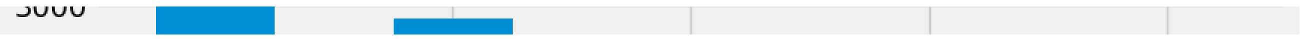
```
Index(['Airline', 'Date_of_Journey', 'Source', 'Destination', 'Route',
       'Dep_Time', 'Arrival_Time', 'Duration', 'Total_Stops',
       'Additional_Info'],
      dtype='object')
```

```
#plotting a barchart for each of the categorical value
for column in category_cols:
  plt.figure(figsize=(20,4))
  plt.subplot(121)
  data[column].value_counts().plot(kind='bar')
  plt.title(column)
```


Airline
4000
3000
2000
1000
0
Jet Airways
IndiGo
Air India
Multiple carriers
SpiceJet
Vistara
Air Asia
GoAir
Multiple carriers Premium economy
Jet Airways Business
Vistara Premium economy
Trujet


Date_of_Journey
500
400
300
200
100
0
18/05/2019
6/06/2019
21/05/2019
9/06/2019
12/06/2019
9/05/2019
21/03/2019
15/05/2019
27/05/2019
27/06/2019
24/06/2019
1/06/2019
3/06/2019
15/06/2019
24/03/2019
6/03/2019
27/03/2019
24/05/2019
6/05/2019
1/05/2019
12/05/2019
1/04/2019
3/03/2019
9/03/2019
15/03/2019
18/03/2019
01/03/2019
12/03/2019
9/04/2019
3/04/2019
21/06/2019
18/06/2019
09/03/2019
6/04/2019
03/03/2019
06/03/2019
27/04/2019
24/04/2019
3/05/2019
15/04/2019
21/04/2019
18/04/2019
12/04/2019
1/03/2019


Source
4000
3000



```
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=
data.drop(['Time_of_Arrival'],axis=1,inplace=True)
```

Replacing Missing Values

```
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
```

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

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

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

```
#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_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_Hours=data.Travel_Hours.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  City
```

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

```
      6474   Air India   Mumbai    Hyderabad           2         No info   17327     ?    NaN
                                                                                  HYD
```

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

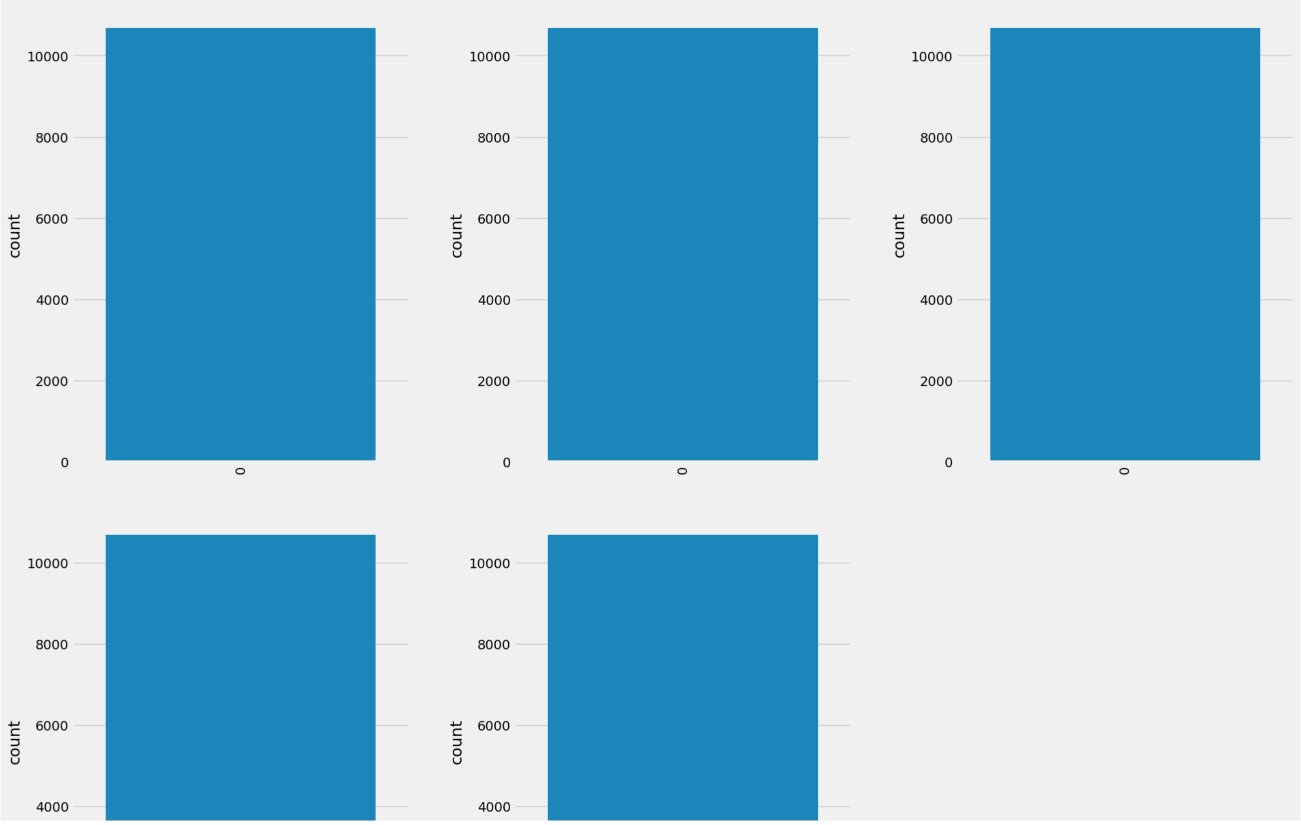
```
categorical=['Airline','Source','Destination','Additional_Info','City1']
numerical=['Total_Stops','Date','Month','Year','Dep_Time_Hour','Dep_Time_Mins','Arrival_da
```

visual Analysis

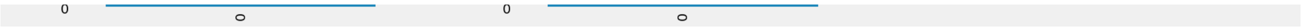
sample visualization

```
import numpy as np
from matplotlib import pyplot as plt

ys=200+np.random.randn(100)
x=[x for x in range(len(ys))]

plt.plot(x,ys,'-')
plt.fill_between(x,ys,195,where=(ys>195),facecolor='g',alpha=0.6)

plt.title("sample visualization")
plt.show()
```

```
#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(data['Price'])
  plt.xticks(rotation=90)
  plt.tight_layout(pad=3.0)
  c=c+1

plt.show()
```

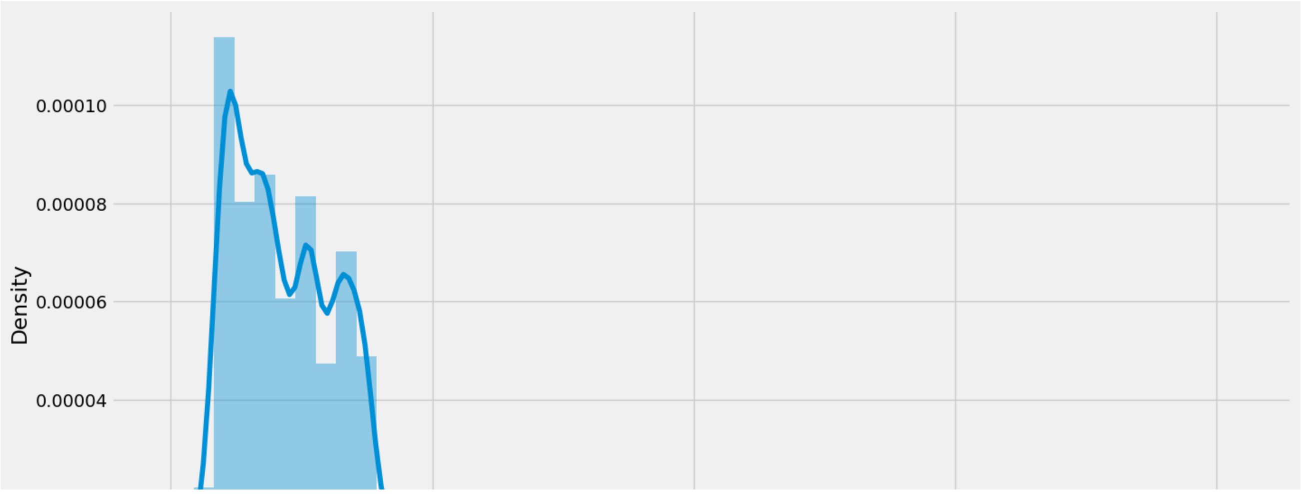


Distribution of price column



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

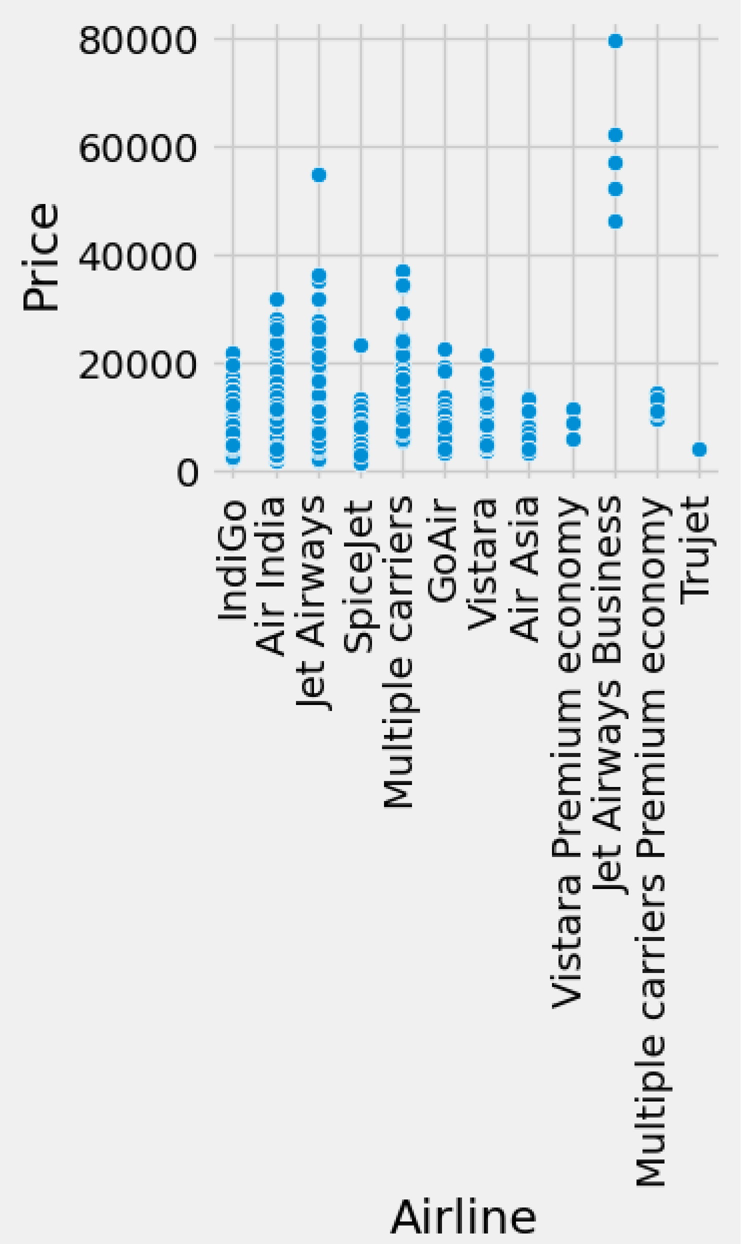
```
<Axes: xlabel='Price', ylabel='Density'>
```

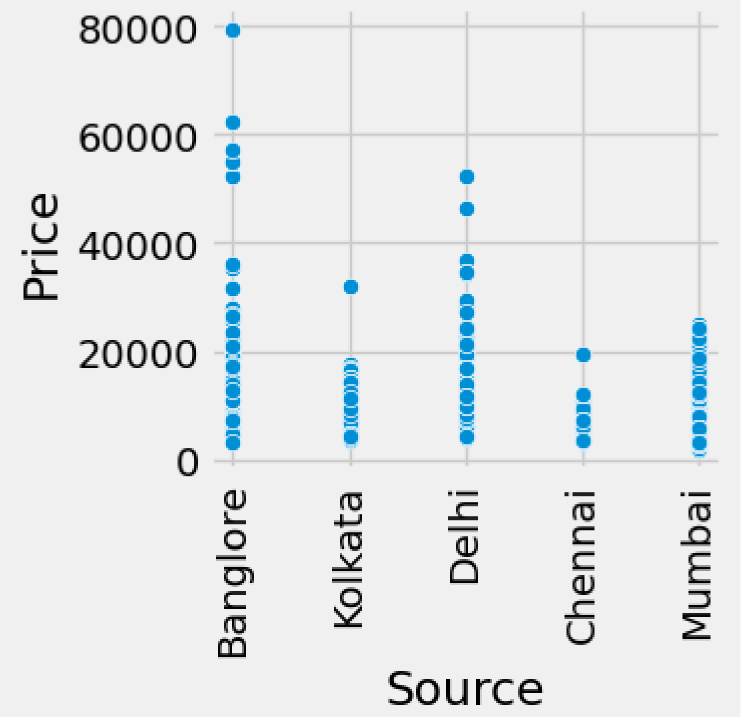


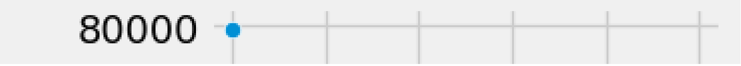
```
data.columns
```

```
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')
```

```
# Checking the relation of price with categorical data

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()
```


Price
80000
60000
40000
20000
0
IndiGo
Air India
Jet Airways
SpiceJet
Multiple carriers
GoAir
Vistara
Air Asia
Vistara Premium economy
Jet Airways Business
Multiple carriers Premium economy
Trujet
Airline


Price
80000
60000
40000
20000
0
Banglore
Kolkata
Delhi
Chennai
Mumbai
Source


80000

```
# Checking flight with high prices
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
0	IndiGo	Banglore	New Delhi	non-stop	No info	3897	BLR ? DEL	NaN
1	Air India	Kolkata	Banglore	2	No info	7662	CCU ? IXR ? BBI ? BLR	NaN
2	Jet Airways	Delhi	Cochin	2	No info	13882	DEL ? LKO ? BOM ? COK	NaN
3	IndiGo	Kolkata	Banglore	1	No info	6218	CCU ? NAG ? BLR	NaN
4	IndiGo	Banglore	New Delhi	1	No info	13302	BLR ? NAG ? DEL	NaN

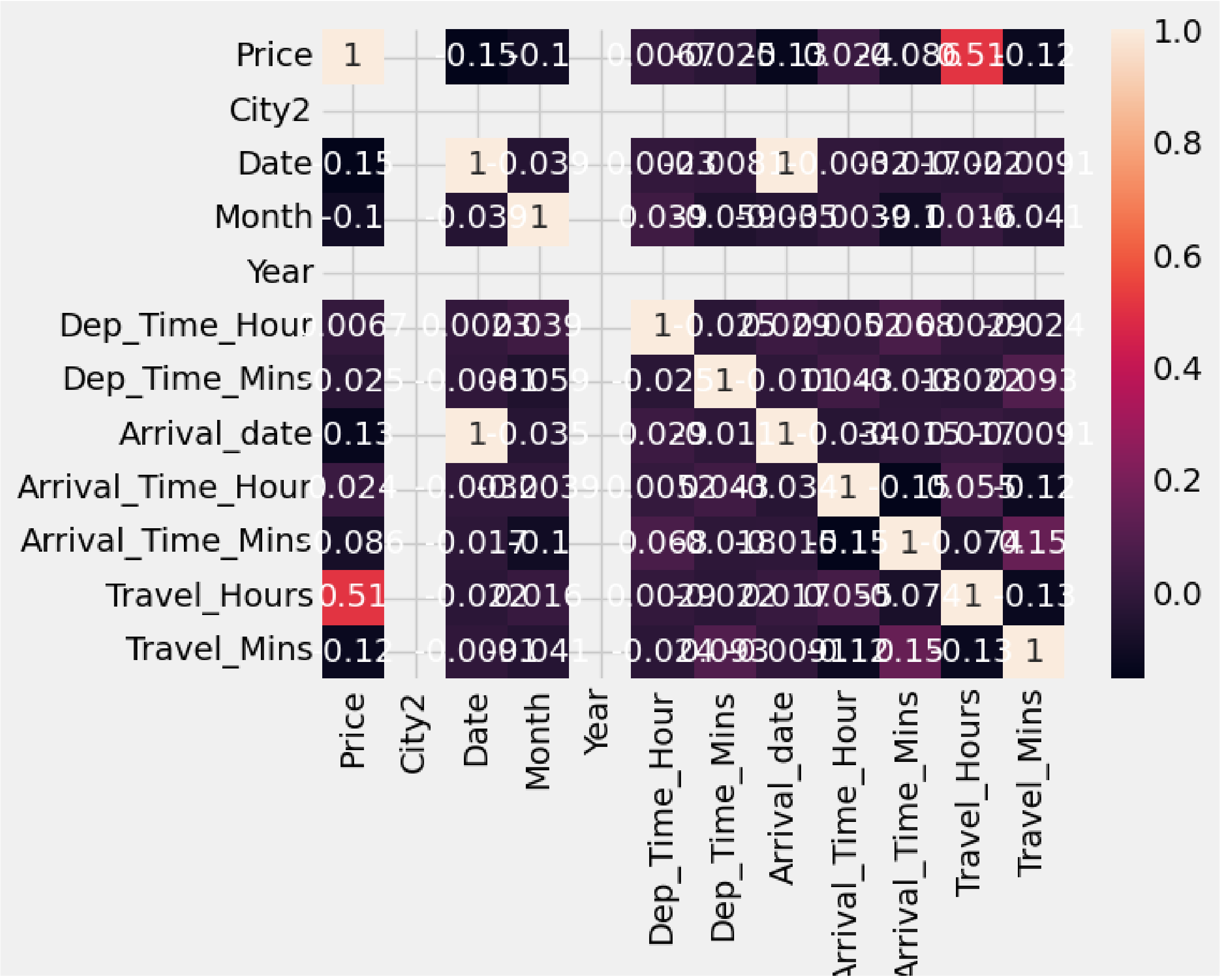
```
data['Year'].max()
```

```
2019
```

Checking the correlation using HeatMap

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

```
<Axes: >
```



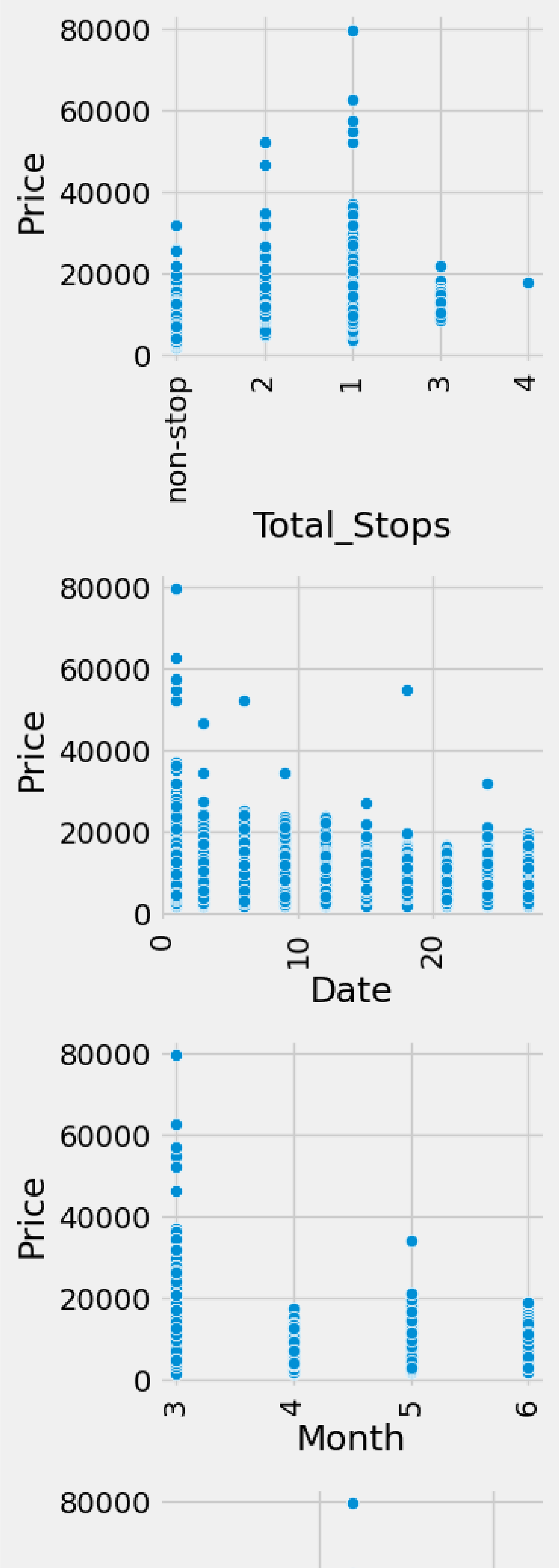
```
data.info()
```

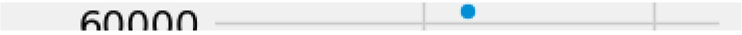
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10682 entries, 0 to 10682
Data columns (total 19 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Airline            10682 non-null  object 
 1   Source             10682 non-null  object 
 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: 1.9+ MB
```

```
data
```

	Airline	Source	Destination	Total_Stops	Additional_Info	Price	City1	Cit

```
# Checking relation price with numerical values
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()
```

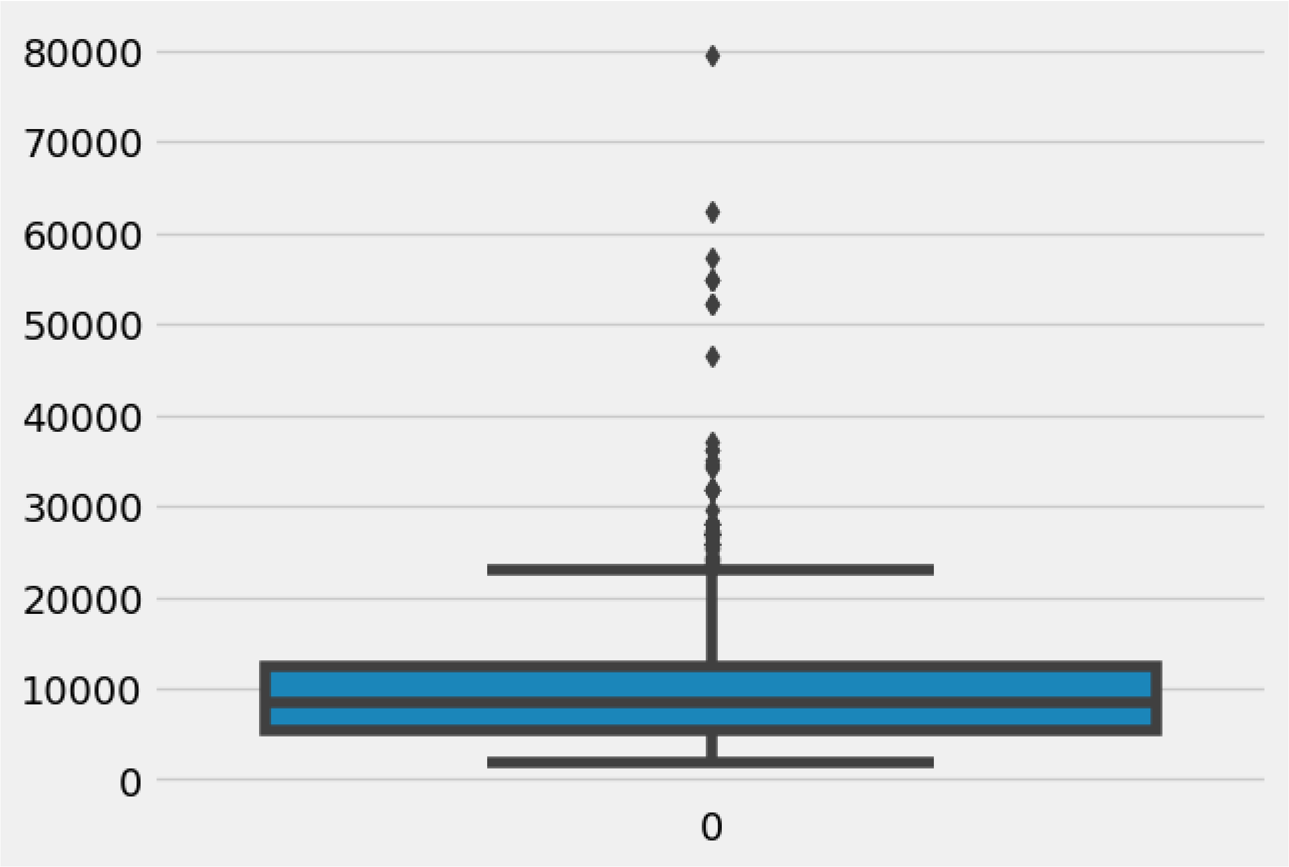

Price
80000
60000
40000
20000
0
non-stop
2
1
3
4
Total_Stops
Price
80000
60000
40000
20000
0
0
10
20
Date
Price
80000
60000
40000
20000
0
3
4
5
6
Month
80000



Outlier detection for Price column

```
# Detecting the outliers
import seaborn as sns
sns.boxplot(data['Price'])
```

```
<Axes: >
```

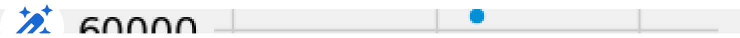


Lable Encodig

```
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	C
0	3	0	5	4	7	3897	18	0	
1	1	3	0	1	7	7662	84	0	
2	4	2	1	1	7	13882	118	0	
3	3	3	0	0	7	6218	91	0	
4	3	0	5	0	7	13302	29	0	

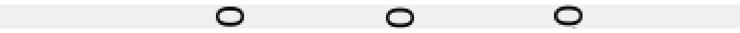


Output Columns

```
data.head()
```

	Airline	Source	Destination	Total_Stops	Additional_Info	Price	City1	City2	C
0	3	0	5	4	7	3897	18	0	
1	1	3	0	1	7	7662	84	0	
2	4	2	1	1	7	13882	118	0	
3	3	3	0	0	7	6218	91	0	
4	3	0	5	0	7	13302	29	0	

```
data=data[['Airline','Source','Destination','Date','Month','Year','Dep_Time_Hour','Dep_Tim
```



```
data.head()
```

	Airline	Source	Destination	Date	Month	Year	Dep_Time_Hour	Dep_Time_Mins	Arr
0	3	0	5	24	3	2019	22	20	
1	1	3	0	1	5	2019	5	50	
2	4	2	1	9	6	2019	9	25	
3	3	3	0	12	5	2019	18	5	
4	3	0	5	1	3	2019	16	50	

Milestone 3

Exploratory Data Analysis

Descriptive Statistical

```
data.describe()
```

```
             Airline       Source  Destination         Date        Month     Year  D
count   10682.000000 10682.000000 10682.000000 10682.000000 10682.000000  10682.0
mean        3.966205     1.952069     1.435967    13.509081     4.708762   2019.0
std         2.352090     1.177110     1.474773     8.479363     1.164294      0.0
min         0.000000     0.000000     0.000000     1.000000     3.000000   2019.0
25%         3.000000     2.000000     0.000000     6.000000     3.000000   2019.0
50%         4.000000     2.000000     1.000000    12.000000     5.000000   2019.0
75%         4.000000     3.000000     2.000000    21.000000     6.000000   2019.0
max        11.000000     4.000000     5.000000    27.000000     6.000000   2019.0
```

Scaling the Data

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

```
data1=ss.fit_transform(data)
```

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

```
     Airline    Source  Destination      Date     Month  Year  Dep_Time_Hour  Dep_Tim
0  -0.410805 -1.658435     2.416778  1.237288 -1.467707   0.0       1.654268      -0.
1  -1.261152  0.890299    -0.973732 -1.475307  0.250153   0.0      -1.303000       1.
2   0.014369  0.040721    -0.295630 -0.531796  1.109082   0.0      -0.607172       0.
3  -0.410805  0.890299    -0.973732 -0.177979  0.250153   0.0       0.958440      -1.
4  -0.410805 -1.658435     2.416778 -1.475307 -1.467707   0.0       0.610527       1.
```

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

Splitting 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_state=42)
```

```
x_train.head()
```

	Airline	Source	Destination	Date	Month	Year	Dep_Time_Hour	Dep_
10004	0.864716	0.040721	-0.29563	1.591104	0.250153	0.0	-0.781129	
3684	0.014369	0.040721	-0.29563	-0.531796	0.250153	0.0	-0.259258	
1034	1.715063	0.040721	-0.29563	1.237288	-0.608777	0.0	0.436570	
3909	0.864716	0.040721	-0.29563	0.883471	-1.467707	0.0	-0.085301	
3088	-1.261152	0.040721	-0.29563	1.237288	1.109082	0.0	0.784483	

```
x_train.shape
```

```
(8545, 9)
```

```
y_train.shape
```

```
(8545,)
```

Milestone 4

Model building

Using Ensemble Techniques

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

```
from sklearn.metrics import r2_score,mean_absolute_error,mean_squared_error
```

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

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

```
RandomForestRegressor()
R2 score is 0.833020601170679
R2 for train data 0.9105955758611345
Mean Absolute Error is 0.27361502073943944
Mean Squared Error is 0.16604267786110014
Root Mean Squarded Error is 0.4074833467285505
GradientBoostingRegressor()
R2 score is 0.7586150253290672
R2 for train data 0.7199250828810017
Mean Absolute Error is 0.3700514763508659
Mean Squared Error is 0.24003085333157612
Root Mean Squarded Error is 0.4899294370943392
AdaBoostRegressor()
R2 score is 0.3275715393439753
R2 for train data 0.348524998044629
Mean Absolute Error is 0.6374032115068605
Mean Squared Error is 0.6686562717324748
Root Mean Squarded Error is 0.817714052546778
```

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_error
```

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



for i in [knn,svr,dt]:
  i.fit(x_train,y_train)
  y_pred=i.predict(x_test)
  test_score=r2_score(y_test,y_pred)
  train_score=r2_score(y_train,i.predict(x_train))
  if abs(train_score-test_score)<=0.1:
    print(i)
    print('R2 Score is',r2_score(y_test,y_pred))
    print('R2 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_pred,squared=False)))
```

```
KNeighborsRegressor()
R2 Score is 0.7067031916509059
R2 Score for train data 0.7708199280026578
Mean Absolute Error is 0.36928172590251457
Mean Squared Error is 0.2916514720248588
Root Mean Squared Error is 0.5400476571793075
SVR()
R2 Score is 0.5890645626885227
R2 Score for train data 0.5570926824460594
Mean Absolute Error is 0.44723971869314205
Mean Squared Error is 0.40863017185110667
Root Mean Squared Error is 0.6392418727298038
```

Checking cross validation for RandomForestRegressor

```
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() 0.7687854467304049
RandomForestRegressor() 0.7696443473741302
RandomForestRegressor() 0.7817983669249348
```

```
rfr
```

```
▾                    RandomForestRegressor
RandomForestRegressor(max_features='sqrt', n_estimators=10)
```

Hypertuning the model

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

```
param_grid={'n_estimators':[10,30,50,70,100],'max_depth':[None,1,2,3],'max_features':['aut
rfr=RandomForestRegressor()
rf_res=RandomizedSearchCV(estimator=rfr,param_distributions=param_grid,cv=3,verbose=2,n_jo
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_job
gb_res.fit(x_train,y_train)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
▸         RandomizedSearchCV
▸ estimator: GradientBoostingRegressor
      ▸ GradientBoostingRegressor
```

Accuracy

```
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 0.8830371391616789
test accuracy 0.7445652199866283
```

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 0.8252359370660914
test accuracy 0.6531487599455481
```

Checking cross validation for RandomForestRegressor

```
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() 0.7655573723652831
RandomForestRegressor() 0.7724983811643485
RandomForestRegressor() 0.7836926794833963
```

```
rfr
```

```
▾                    RandomForestRegressor
RandomForestRegressor(max_features='sqrt', n_estimators=10)
```

Evaluating performance of the model and saving the 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 0.8850928588419827
test accuracy 0.7702120755570521
```

```
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	-0.110966
5032	-1.133955	-1.190563
2483	0.826714	1.171675
...	...	...
9796	-0.364002	0.824871
9870	-0.968253	-0.614942

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

8802	0.430470	0.303712

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

```
price_list
```

	Price
0	0.902105
1	-0.764443
2	-0.032446
3	-1.168961
4	0.900212
...	...
2132	0.710256
2133	-0.938430
2134	-0.368481
2135	0.234306
2136	0.670456

2137 rows × 1 columns

Milestone 6

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

✓ 0s completed at 11:50 AM ● ×