```
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')
```

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

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Du
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR ? DEL	22:20	01:10 22 Mar	
1	Air India	1/05/2019	Kolkata	Banglore	CCU ? IXR ? BBI ? BLR	05:50	13:15	
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL ? LKO ? BOM ? COK	09:25	04:25 10 Jun	
3	IndiGo	12/05/2019	Kolkata	Banglore	CCU ? NAG ? BLR	18:05	23:30	
4	IndiGo	01/03/2019	Banglore	New Delhi	BLR ? NAG ? DEL	16:50	21:35	

```
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
```

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

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

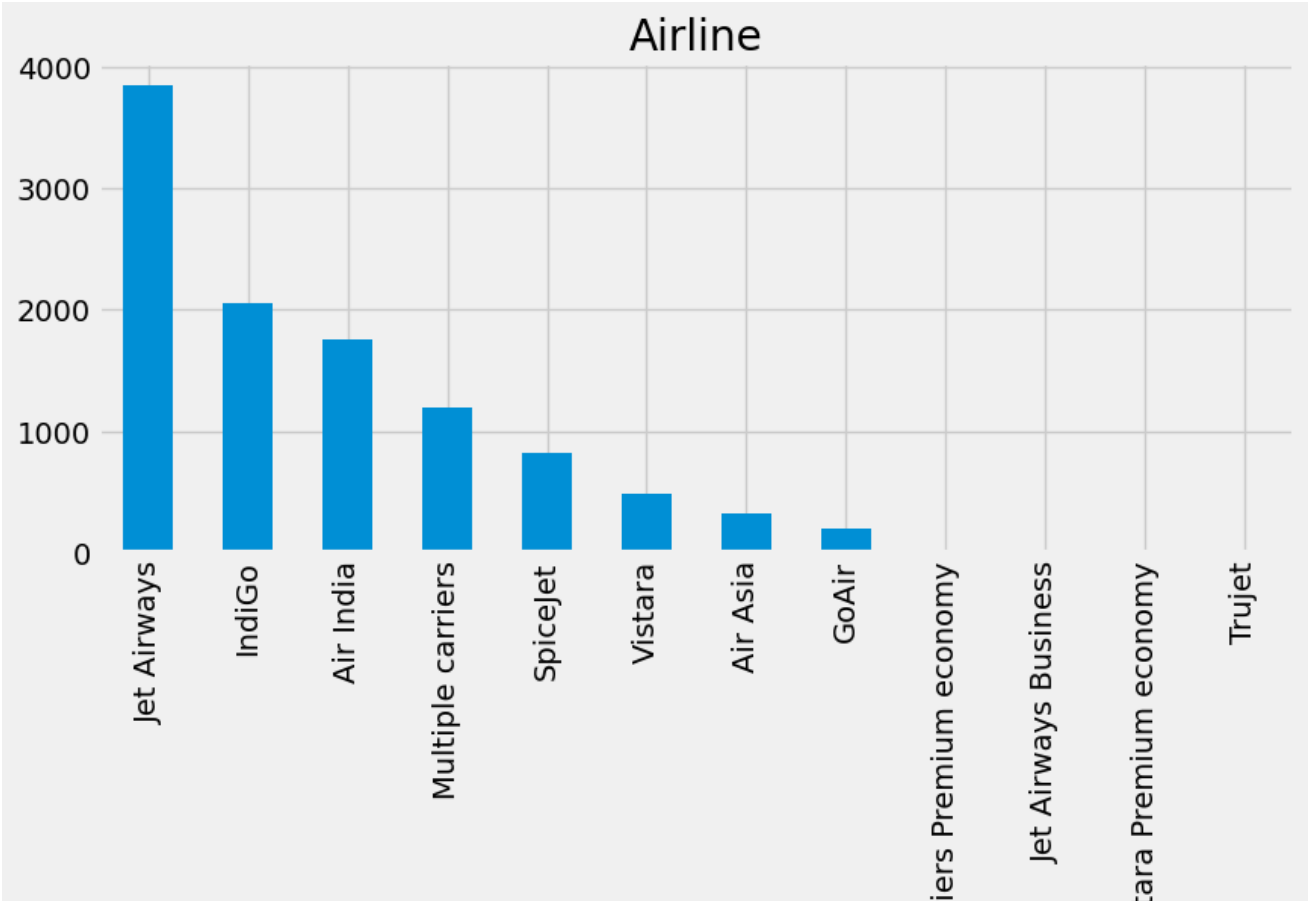
```
for i in category:
    print(i,data[i].unique())
```

```
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)
```




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

```
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     10683
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
```

```
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	De
6474	Air India	Mumbai	Hyderabad	2	No info	17327	BOM ? GOI ? PNQ ? HYD	NaN	None	6	3	2019	

```
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_Mins','Arrival_Time_Mins','Travel_Hours','Trave
```

Double-click (or enter) to edit

```
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()
```

```
    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()
```

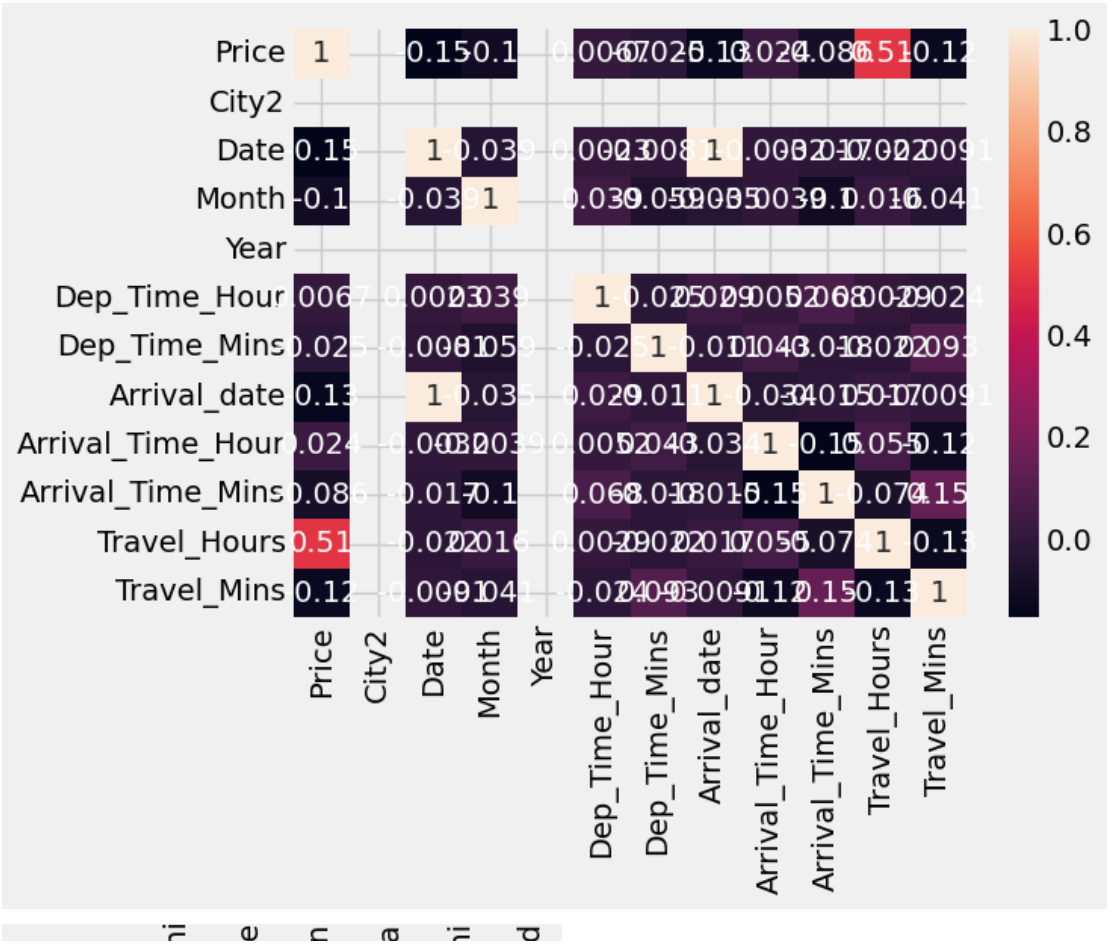
	Airline	Source	Destination	Total_Stops	Additional_Info	Price	City1	City2	City3	Date	Month	Year	Dep_
0	IndiGo	Banglore	New Delhi	non-stop	No info	3897	BLR ? DEL	NaN	None	24	3	2019	
1	Air India	Kolkata	Banglore	2	No info	7662	CCU ? IXR ? BBI ? BLR	NaN	None	1	5	2019	
2	Jet Airways	Delhi	Cochin	2	No info	13882	DEL ? LKO ? BOM ? COK	NaN	None	9	6	2019	
3	IndiGo	Kolkata	Banglore	1	No info	6218	CCU ? NAG ? BLR	NaN	None	12	5	2019	
4	IndiGo	Banglore	New Delhi	1	No info	13302	BLR ? NAG	NaN	None	1	3	2019	

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

```
2019
```

```
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
```

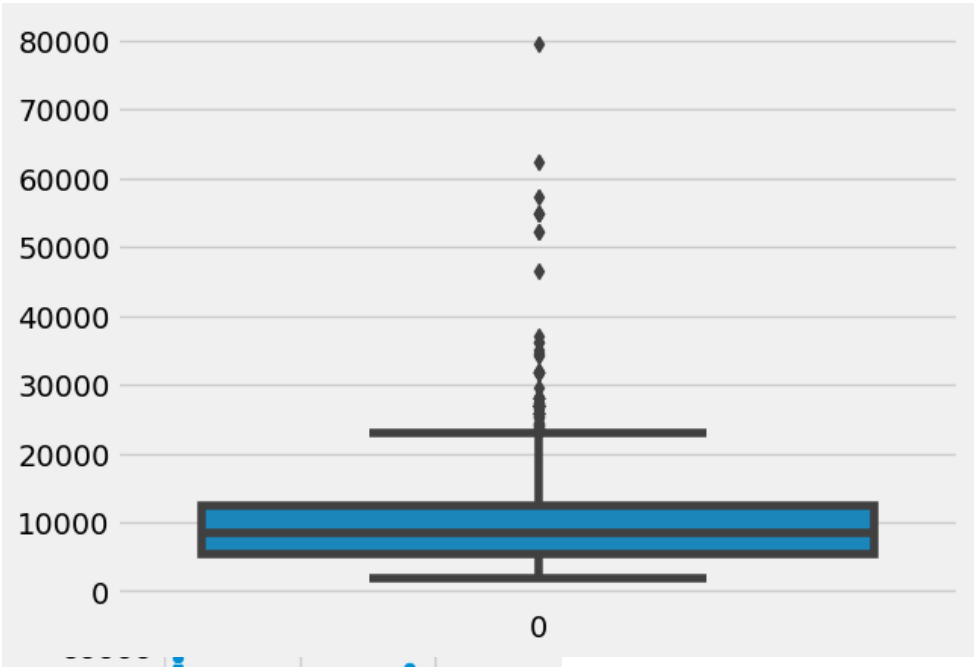
	Airline	Source	Destination	Total_Stops	Additional_Info	Price	City1	City2	City3	Date	Month	Year
0	IndiGo	Banglore	New Delhi	non-stop	No info	3897	BLR	NaN	NaN	24	3	2019

```
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()
```

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

```
<Axes: >
```




```
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_T:
0	3	0	5	4	7	3897	18	0	0	24	3	2019	
1	1	3	0	1	7	7662	84	0	0	1	5	2019	
2	4	2	1	1	7	13882	118	0	0	9	6	2019	
3	3	3	0	0	7	6218	91	0	0	12	5	2019	
4	3	0	5	0	7	13302	29	0	0	1	3	2019	

```
data = data[['Airline','Source','Destination','Date','Month','Year','Dep_Time_Hour','Dep_Time_Mins','Arrival_date','Arri
```

```
data.head()
```

	Airline	Source	Destination	Date	Month	Year	Dep_Time_Hour	Dep_Time_Mins	Arrival_date	Arrival_Time_Hour
0	3	0	5	24	3	2019	22	20	22	1
1	1	3	0	1	5	2019	5	50	1	13
2	4	2	1	9	6	2019	9	25	10	4
3	3	3	0	12	5	2019	18	5	12	23
4	3	0	5	1	3	2019	16	50	1	21

```
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
0	3	0	5	24	3	2019	22	20	22	1
1	1	3	0	1	5	2019	5	50	1	13
2	4	2	1	9	6	2019	9	25	10	4
3	3	3	0	12	5	2019	18	5	12	23
4	3	0	5	1	3	2019	16	50	1	21

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

```
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_Time_Mins	Arrival_date	Arriv
10004	0.864716	0.040721	-0.29563	1.591104	0.250153	0.0	-0.781129	0.297937	1.546321	
3684	0.014369	0.040721	-0.29563	-0.531796	0.250153	0.0	-0.259258	0.297937	-0.461621	
1034	1.715063	0.040721	-0.29563	1.237288	-0.608777	0.0	0.436570	1.097240	1.191978	
3909	0.864716	0.040721	-0.29563	0.883471	-1.467707	0.0	-0.085301	1.363674	0.955750	
3088	-1.261152	0.040721	-0.29563	1.237288	1.109082	0.0	0.784483	-0.501367	1.310092	

```
x_train.shape
```

```
(8545, 11)
```

Double-click (or enter) to edit

```
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor, AdaBoostRegressor
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 Squared Error is",(mean_squared_error(y_pred,y_test,squared=False)))


    RandomForestRegressor()
    R2 score is 0.850432571284373
    R2 for train data 0.9506969959263003
    Mean Absolute Error is 0.25475317086441596
    Mean Squared Error is 0.14872838541074593
    Root Mean Squared Error is 0.38565319318105734
    GradientBoostingRegressor()
    R2 score is 0.7652955778741217
    R2 for train data 0.7338510043179753
    Mean Absolute Error is 0.36494013808679193
    Mean Squared Error is 0.23338777734765506
    Root Mean Squared Error is 0.48310224316148137
    AdaBoostRegressor()
    R2 score is 0.2533535201196103
    R2 for train data 0.25941367300272333
    Mean Absolute Error is 0.7258662199205039
    Mean Squared Error is 0.7424579427407443
    Root Mean Squared Error is 0.8616599925380917
```

```
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"
    _________________________________________________________________
     Layer (type)                Output Shape              Param #
    =================================================================
     dense (Dense)               (None, 7)                 84

     dense_1 (Dense)             (None, 7)                 56

     dense_2 (Dense)             (None, 1)                 8

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

```
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: 1.0145 - mae: 0.7881
    Epoch 2/10
    428/428 [==============================] - 1s 2ms/step - loss: 0.8853 - mae: 0.7353
    Epoch 3/10
    428/428 [==============================] - 1s 2ms/step - loss: 0.8238 - mae: 0.7080
    Epoch 4/10
    428/428 [==============================] - 1s 2ms/step - loss: 0.7768 - mae: 0.6834
    Epoch 5/10
    428/428 [==============================] - 1s 2ms/step - loss: 0.7311 - mae: 0.6565
    Epoch 6/10
```

```
428/428 [==============================] - 1s 2ms/step - loss: 0.6987 - mae: 0.6386
Epoch 7/10
428/428 [==============================] - 1s 2ms/step - loss: 0.6760 - mae: 0.6236
Epoch 8/10
428/428 [==============================] - 1s 2ms/step - loss: 0.6555 - mae: 0.6091
Epoch 9/10
428/428 [==============================] - 1s 2ms/step - loss: 0.6388 - mae: 0.5984
Epoch 10/10
428/428 [==============================] - 1s 2ms/step - loss: 0.6243 - mae: 0.5880
<keras.callbacks.History at 0x7f72b828f790>
```

```
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() 0.7892661446182386
RandomForestRegressor() 0.7926319036646371
RandomForestRegressor() 0.8009278254254942
```

```
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 0.9223637224760787
test accuracy 0.7637242809880902
```

```
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) 0.7261930738745277
RandomForestRegressor(max_features='sqrt', n_estimators=10) 0.7290564575182709
RandomForestRegressor(max_features='sqrt', n_estimators=10) 0.727707269447425
```

```
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 0.6364865173543215
test accuracy 0.2501677267836613
```

```
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 0.7337698733752689
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	0.614706

2137 rows × 2 columns

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

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

```
price_list
```

	Price
0	1.354620
1	-0.342400
2	0.036547
3	-1.179241
4	1.243284
...	...
2132	0.842135
2133	-0.825865
2134	-0.508992
2135	0.314771
2136	0.590268

2137 rows × 1 columns

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