```
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, classification_report, confusion_matrix
import warnings
import pickle
from scipy import stats

warnings.filterwarnings('ignore')
plt.style.use('fivethirtyeight')

data = pd.read_csv("/content/sample_data/Data_Train.csv")
data.head()
data.shape
data.isnull().sum()
data.dropna(inplace=True)
data.isnull().sum()
Category = ['Airline', 'Source', 'Destination', 'Additional_Info']

for i in Category:
    print(i, data[i].unique())
category_cols=data.select_dtypes (include=['object']) .columns
category_cols
#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)
data.Route =  data.Route.str.split('->')
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

#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]
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]
#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]

#Next, we divide the 'Duration' column to 'Travel_hours' and Travel_mins'  24
data.Duration=data.Duration.str.split(' ')
```

```
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']
```

```
data['Duration'] = data['Duration'].astype(str)
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
```

```
if 'City4' in data.columns and 'City5' in data.columns and 'City6' in data.columns:
    data.drop(['City4', 'City5', 'City6'], axis=1, inplace=True)

print(data.columns)
```

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

```
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()
data.info()
```

```
data['City3'].fillna ('None', inplace=True)
data['City2'].fillna ('None', inplace=True)
data['Arrival_date'].fillna (data['Date'], inplace=True)
data['Travel_Mins'].fillna(0,inplace=True)
data.info()
```

```
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.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   City1              10682 non-null  object
 7   City2              10682 non-null  object
 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  object
 18  Travel_Mins        10682 non-null  object
dtypes: int64(9), object(10)
memory usage: 1.6+ MB
```

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

```
   Airline  Source  Destination  Total_Stops  Additional_Info  Price  City1  City2  City3  Date  Month  Year  Dep
```

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

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-322-7bece7b89a9b> in <cell line: 1>()
----> 1 data.Travel_Hours=data.Travel_Hours.astype('int64')

                                   7 frames
/usr/local/lib/python3.9/dist-packages/pandas/_libs/lib.pyx in pandas._libs.lib.astype_intsafe()

ValueError: invalid literal for int() with base 10: 'n'
```

SEARCH STACK OVERFLOW

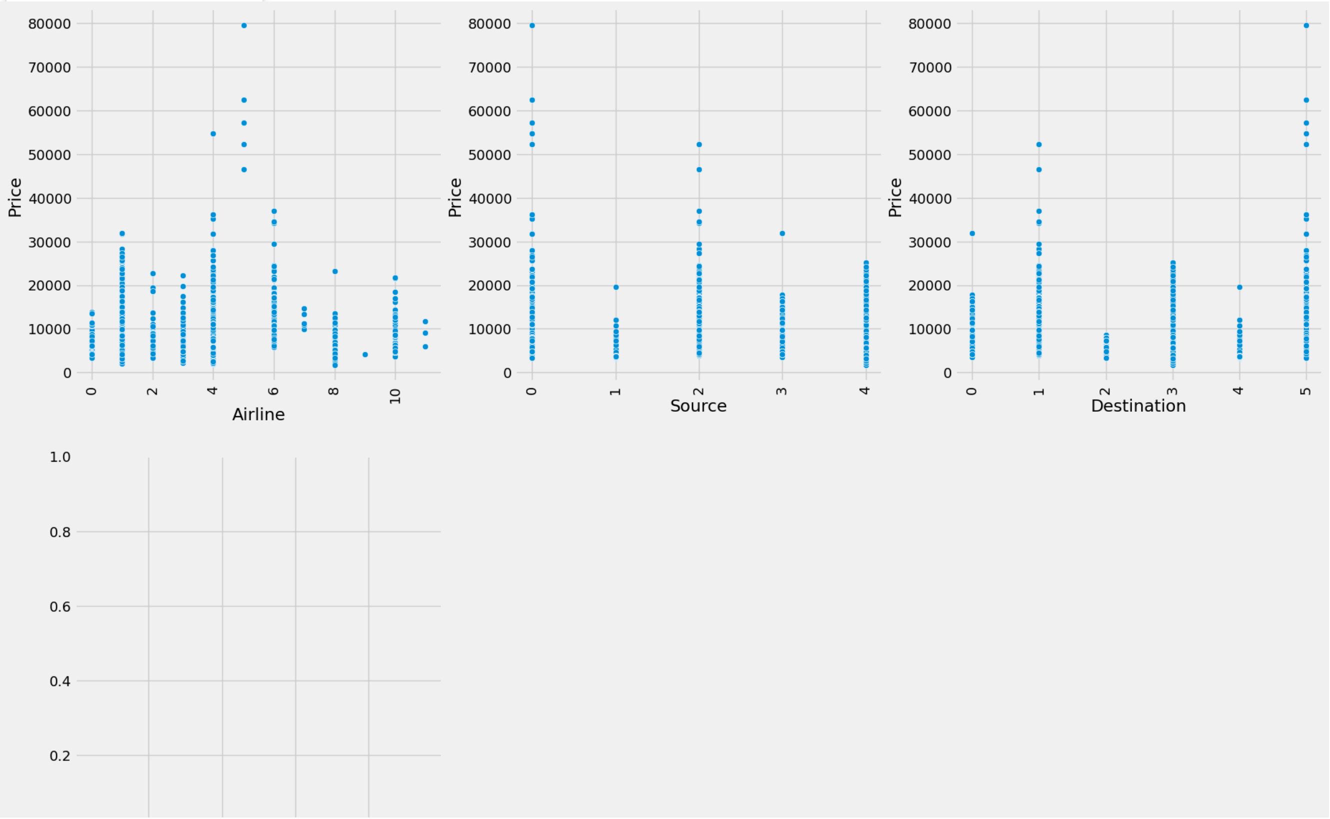
```
categorical=['Airline', 'Source', 'Destination', 'Additional Info', 'City1']
```

```
l_Stops', 'Date', 'Month', 'Year', 'Dep_Time_Hour', 'Dep_Time_Mins', 'Arrival_date', 'Arrival_Time_Hour', 'Arrival_T
```

```
import seaborn as sns
c=1
plt.figure(figsize=(20,45))
for i in categorical:
  plt.subplot(6,3,c)
  sns.scatterplot(x=data[i],y=data.Price)
  plt.xticks(rotation=90)

  c=c+1
plt.show()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.9/dist-packages/pandas/core/indexes/base.py in get_loc(self, key, method, tolerance)
   3628             try:
-> 3629                 return self._engine.get_loc(casted_key)
   3630             except KeyError as err:

                                 4 frames
pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'Additional Info'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.9/dist-packages/pandas/core/indexes/base.py in get_loc(self, key, method, tolerance)
   3629                 return self._engine.get_loc(casted_key)
   3630             except KeyError as err:
-> 3631                 raise KeyError(key) from err
   3632             except TypeError:
   3633                 # If we have a listlike key, _check_indexing_error will raise

KeyError: 'Additional Info'
```

SEARCH STACK OVERFLOW



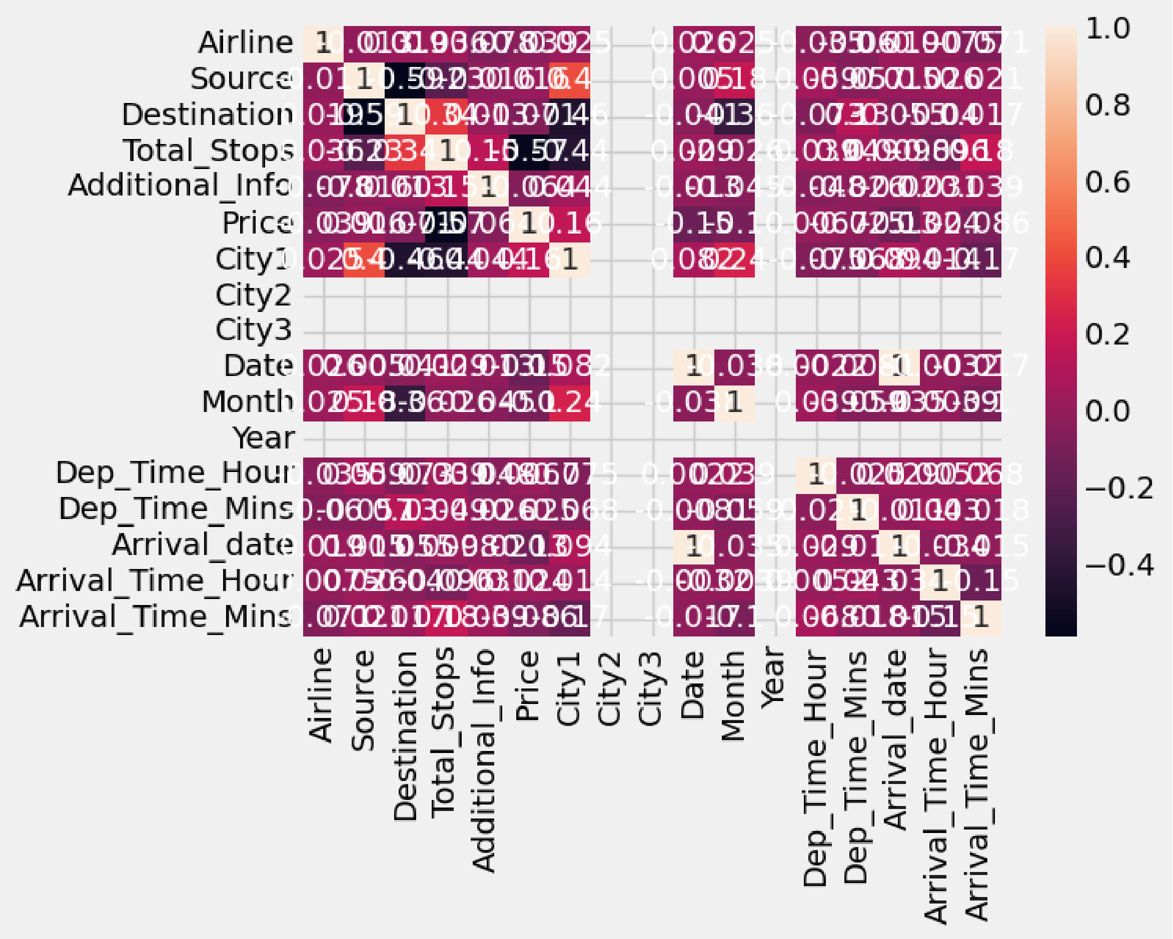
```
data[data. Price>50000]
```

```
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  De
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
```

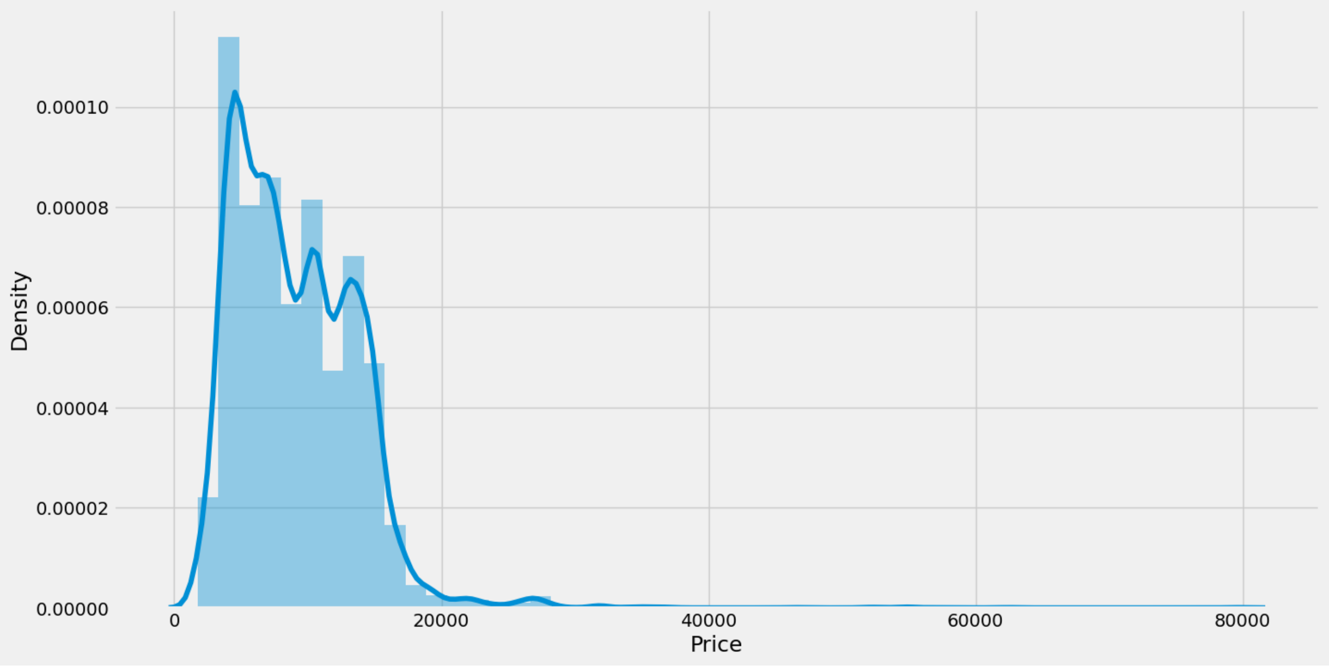
```
data['Year'].max()
sns.heatmap (data.corr(), annot=True)
```

```
<Axes: >
```



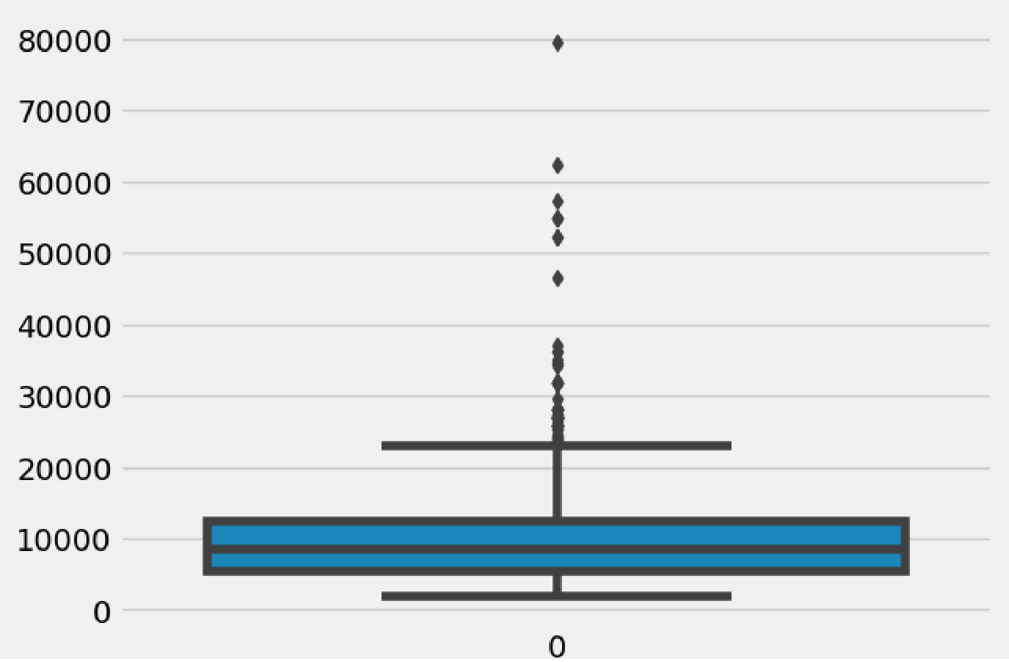
```
plt.figure(figsize=(15,8))
sns.distplot(data.Price)
```

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



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

```
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()
```



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

	Airline	Source	Destination	Total_Stops	Additional_Info	Price	City1	City2	City3	Date	Month	Year	De
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_
data.head()
```

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

```
### Scaling the Data
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
datal = ss.fit_transform(data)
```

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

	Airline	Source	Destination	Date	Month	Year	Dep_Time_Hour	Dep_Time_Mins	Arrival_date	Arri
0	-0.410934	-1.658354	2.416648	1.237192	-1.467619	0.0	1.654162	-0.234832	0.955658	
1	-1.261305	0.890262	-0.973718	-1.475375	0.250165	0.0	-1.303018	1.363790	-1.524701	
2	0.014251	0.040723	-0.295645	-0.531874	1.109057	0.0	-0.607211	0.031605	-0.461690	
3	-0.410934	0.890262	-0.973718	-0.178060	0.250165	0.0	0.958355	-1.034142	-0.225465	
4	-0.410934	-1.658354	2.416648	-1.475375	-1.467619	0.0	0.610452	1.363790	-1.524701	

```
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	A
10609	0.014251	1.739800	1.060501	0.529566	0.250165	0.0	-0.955114	-1.034142	0.483209	
1034	1.714993	0.040723	-0.295645	1.237192	-0.608727	0.0	0.436500	1.097353	1.191883	
8122	0.014251	0.040723	-0.295645	1.591005	1.109057	0.0	-1.824873	-0.501269	1.546220	
4779	0.014251	0.890262	-0.973718	-1.475375	-0.608727	0.0	-1.129066	0.298042	-1.524701	
3207	-0.410934	0.890262	-0.973718	1.237192	0.250165	0.0	0.958355	-1.034142	1.191883	

```
x_train.shape
```

```
(8544, 11)
```

```
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 Sqaured Error is", (mean_squared_error(y_pred,y_test, squared=False)))
```

```
AdaBoostRegressor()
R2 score is 0.1920075019031262
R2 for train data 0.2397620967213413
Mean Absolute Error is 0.7474281430734462
Mean Squared Error is 0.8023576762712405
Root Mean Sqaured Error is 0.8957442024770468
```

```
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 0.7357369816529409
R2 Score for train data 0.7900498333828809
Mean Absolute Error is 0.35463454315938664
Mean Squared Error is 0.26242008660326566
Root Mean Squared Error is 0.512269544871902
SVR()
R2 Score is 0.6399736388140904
R2 Score for train data 0.5969176412610055
Mean Absolute Error is 0.40820604052912457
Mean Squared Error is 0.3575155898574727
Root Mean Squared Error is 0.5979260739066935
```

```
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() 0.7913041688665847
RandomForestRegressor() 0.7922244703050666
RandomForestRegressor() 0.8010036945252192
```

```
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 [==============================] - 2s 2ms/step - loss: 1.0014 - mae: 0.7697
```

```
Epoch 2/10
428/428 [==============================] - 1s 2ms/step - loss: 0.8445 - mae: 0.6867
Epoch 3/10
428/428 [==============================] - 1s 2ms/step - loss: 0.7852 - mae: 0.6515
Epoch 4/10
428/428 [==============================] - 1s 2ms/step - loss: 0.7519 - mae: 0.6348
Epoch 5/10
428/428 [==============================] - 1s 2ms/step - loss: 0.7240 - mae: 0.6244
Epoch 6/10
428/428 [==============================] - 1s 2ms/step - loss: 0.7012 - mae: 0.6170
Epoch 7/10
428/428 [==============================] - 1s 2ms/step - loss: 0.6856 - mae: 0.6114
Epoch 8/10
428/428 [==============================] - 1s 2ms/step - loss: 0.6749 - mae: 0.6093
Epoch 9/10
428/428 [==============================] - 1s 3ms/step - loss: 0.6686 - mae: 0.6066
Epoch 10/10
428/428 [==============================] - 1s 3ms/step - loss: 0.6601 - mae: 0.6033
<keras.callbacks.History at 0x7fc48177bdf0>
```

```
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())
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)
```

```
RandomForestRegressor() 0.7909852737870107
RandomForestRegressor() 0.7935431966039664
RandomForestRegressor() 0.8004797889513
Fitting 3 folds for each of 10 candidates, totalling 30 fits
```

▸ **RandomizedSearchCV**

▸ **estimator: RandomForestRegressor**

▸ RandomForestRegressor

```
gb = GradientBoostingRegressor()

# Define parameter grid to search over
param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.1, 0.01, 0.001]
}

# Perform randomized search over parameter grid
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 0.9297123116878924
     test accuracy 0.7740524823579749
```

```
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) 0.72661809392105
     RandomForestRegressor(max_features='sqrt', n_estimators=10) 0.7287548229046766
     RandomForestRegressor(max_features='sqrt', n_estimators=10) 0.728029951483208
```

```
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 0.636247261013868
     test accuracy 0.24929533156419104
```

```
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)
```

```
     KNeighborsRegressor()
     SVR()
```

```
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 0.636247261013868
     test accuracy 0.24929533156419104
```

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

```
KNeighborsRegressor(n_jobs=-1, n_neighbors=2) 0.6306338018391912
KNeighborsRegressor(n_jobs=-1, n_neighbors=2) 0.6447308601134175
KNeighborsRegressor(n_jobs=-1, n_neighbors=2) 0.664555765507016
```

	Actual	Predicted
4830	-0.349272	-0.455760
3771	-0.251459	-0.171648
1523	-0.677410	0.638830
3393	1.562086	0.826648
4169	-0.232157	-0.719485
...	...	...
9869	-0.968245	-0.614949
10061	-0.354477	-0.354477
6011	0.348404	0.348404

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

	Price
0	-0.541116
1	-0.057338
2	0.496086
3	0.964626
4	-0.683179
...	...
2132	-0.614212
2133	-0.549343
2134	-0.374603
2135	0.738894
2136	1.056930

2137 rows × 1 columns

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

```
from flask import Flask, render_template, request
import numpy as np
import pickle
import pandas as pd
model=pickle.load(open ("model1.pkl",'rb'))

app = Flask (__name__)

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

  return render_template('/content/Flask/home.html')
@app.route("/content/Flask/predict.html")
def pred():
  return render_template('/content/Flask/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[0])
  return render_template('/content/Flask/submit.html', prediction_text=pred[0])
if __name__ == "__main__":
  app.run(debug=False)
```

```
 * Serving Flask app '__main__'
 * Debug mode: off
INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment. Use a productio
 * Running on http://127.0.0.1:5000
INFO:werkzeug:Press CTRL+C to quit
```