In [1]:

```
#Importing libraries

import numpy as np
import pandas as pd
get_ipython().run_line_magic('matplotlib', 'inline')
import matplotlib.pyplot as plt
import seaborn as sns
color = sns.color_palette()
sns.set_style('darkgrid')
from scipy import stats
from scipy.stats import norm, skew
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import KFold, cross_val_score
from sklearn.model_selection import cross_val_predict
from sklearn.preprocessing import RobustScaler
from sklearn.pipeline import make_pipeline
```

In [2]:

```
#Importing Data

train_df = pd.read_excel("D:/MITA/PROJECTS/FLIGHT PRICE PREDICTION/Train_set.xlsx")
test_df = pd.read_excel("D:/MITA/PROJECTS/FLIGHT PRICE PREDICTION/Test_set.xlsx")
```

In [3]:

```
#Combining Train and Test Data

big_df = train_df.append(test_df)
```

```
C:\Users\kheni\AppData\Local\Temp\ipykernel_7820\744722857.py:3: FutureWar
ning: The frame.append method is deprecated and will be removed from panda
s in a future version. Use pandas.concat instead.
  big_df = train_df.append(test_df)
```

In [4]:

```
#Printing out the data

big_df
```

Out[4]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Durat
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 5
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 2
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	5h 2
4	IndiGo	01/03/2019	Banglore	New Delhi	BLR → NAG → DEL	16:50	21:35	4h 4
...	...	...	...	...	...	...	...	
2666	Air India	6/06/2019	Kolkata	Banglore	CCU → DEL → BLR	20:30	20:25 07 Jun	23h 5
2667	IndiGo	27/03/2019	Kolkata	Banglore	CCU → BLR	14:20	16:55	2h 3
2668	Jet Airways	6/03/2019	Delhi	Cochin	DEL → BOM → COK	21:50	04:25 07 Mar	6h 3
2669	Air India	6/03/2019	Delhi	Cochin	DEL → BOM → COK	04:00	19:15	15h 1
2670	Multiple carriers	15/06/2019	Delhi	Cochin	DEL → BOM → COK	04:55	19:15	14h 2

13354 rows × 11 columns

In [5]:

```
#Checking the data types of each column

big_df.dtypes
```

Out[5]:

```
Airline            object
Date_of_Journey    object
Source             object
Destination        object
Route              object
Dep_Time           object
Arrival_Time       object
Duration           object
Total_Stops        object
Additional_Info    object
Price             float64
dtype: object
```

In [6]:

```
#Spliting Date_of_Journey column into three different columns i.e., Date, Month, and Yea
#Converting the types to integer
#Dropping the old 'Date_of_Journey' column

big_df['Date'] = big_df['Date_of_Journey'].str.split('/').str[0]
big_df['Month'] = big_df['Date_of_Journey'].str.split('/').str[1]
big_df['Year'] = big_df['Date_of_Journey'].str.split('/').str[2]

big_df['Date'] = big_df['Date'].astype(int)
big_df['Month'] = big_df['Month'].astype(int)
big_df['Year'] = big_df['Year'].astype(int)

big_df=big_df.drop(['Date_of_Journey'], axis=1)
```

In [7]:

```
#Spliting the 'Arrival_Time' column

big_df['Arrival_Time'] = big_df['Arrival_Time'].str.split(' ').str[0]
big_df['Arrival_Time']
```

Out[7]:

```
0       01:10
1       13:15
2       04:25
3       23:30
4       21:35
        ...
2666    20:25
2667    16:55
2668    04:25
2669    19:15
2670    19:15
Name: Arrival_Time, Length: 13354, dtype: object
```

In [8]:

```
#Spliting Arrival_Time column into two different columns i.e., Hour, and Minutes

big_df['Arrival_Hour'] = big_df['Arrival_Time'].str.split(':').str[0]
big_df['Arrival_Minute'] = big_df['Arrival_Time'].str.split(':').str[1]

#Changing the data types of Arrival_Hours and Arrival_Minute

big_df['Arrival_Hour'] = big_df['Arrival_Hour'].astype(int)
big_df['Arrival_Minute'] = big_df['Arrival_Minute'].astype(int)
```

In [9]:

```
#Printing out 'Arrival_Hour' and 'Arrival_Minute'

big_df[['Arrival_Hour','Arrival_Minute']]
```

Out[9]:

	Arrival_Hour	Arrival_Minute
0	1	10
1	13	15
2	4	25
3	23	30
4	21	35
...	...	...
2666	20	25
2667	16	55
2668	4	25
2669	19	15
2670	19	15

13354 rows × 2 columns

In [10]:

```
#Dropping the old 'Arrival_Time' column

big_df = big_df.drop(['Arrival_Time'], axis = 1)
```

In [11]:

```
#Checking the number of NULL values in 'Total_Stops' column

big_df['Total_Stops'].isna().sum()
```

Out[11]:

1

In [12]:

```
#Filling NULL value of 'Total_Stops' column to '1 Stop'

big_df['Total_Stops'] = big_df['Total_Stops'].fillna('1 Stop')
```

In [13]:

```
#Replacing 'non-stop' to '0 stop' in 'Total_Stops' column

big_df['Total_Stops'] = big_df['Total_Stops'].replace('non-stop', '0 stop')
```

In [14]:

```
#Spliting the 'Total_Stop' column and retriving the number of stops to 'Stop' column

big_df['Stop'] = big_df['Total_Stops'].str.split(' ').str[0]
```

In [15]:

```
#Printing out the 'Stop' column

big_df['Stop']
```

Out[15]:

```
0       0
1       2
2       2
3       1
4       1
       ..
2666    1
2667    0
2668    1
2669    1
2670    1
Name: Stop, Length: 13354, dtype: object
```

In [16]:

```
#Dropping the old 'Total_Stops' column

big_df = big_df.drop(['Total_Stops'], axis = 1)
```

In [17]:

```
#Changing the datatype of 'Stop' column to integer

big_df['Stop'] = big_df['Stop'].astype(int)
```

In [18]:

```
#Printing out the 'Dep_Time' column

big_df['Dep_Time']
```

Out[18]:

```
0       22:20
1       05:50
2       09:25
3       18:05
4       16:50
        ...
2666    20:30
2667    14:20
2668    21:50
2669    04:00
2670    04:55
Name: Dep_Time, Length: 13354, dtype: object
```

In [19]:

```
#Spliting Dep_Time column into two different columns i.e., Hour, and Minute

big_df['Dep_Hour'] = big_df['Dep_Time'].str.split(':').str[0]
big_df['Dep_Minute'] = big_df['Dep_Time'].str.split(':').str[1]
```

In [20]:

```
#Printing out the 'Dep_Hour' and 'Dep_Minute' column

big_df[['Dep_Hour','Dep_Minute']]
```

Out[20]:

	Dep_Hour	Dep_Minute
0	22	20
1	05	50
2	09	25
3	18	05
4	16	50
...	...	...
2666	20	30
2667	14	20
2668	21	50
2669	04	00
2670	04	55

13354 rows × 2 columns

In [21]:

```
#Changing the datatype of 'Dep_Hour' and 'Dep_Minute' column to integer

big_df['Dep_Hour'] = big_df['Dep_Hour'].astype(int)
big_df['Dep_Minute'] = big_df['Dep_Minute'].astype(int)
```

In [22]:

```
#Dropping the old 'Dep_Time' column

big_df = big_df.drop(['Dep_Time'], axis = 1)
```

In [23]:

```
#Revising Route column by removing the arrow signs

big_df['Route_1'] = big_df['Route'].str.split('→').str[0]
big_df['Route_2'] = big_df['Route'].str.split('→').str[1]
big_df['Route_3'] = big_df['Route'].str.split('→').str[2]
big_df['Route_4'] = big_df['Route'].str.split('→').str[3]
big_df['Route_5'] = big_df['Route'].str.split('→').str[4]
```

In [24]:

```
#Checking for NULL values

big_df['Route_1'].isna().sum()
```

Out[24]:

```
1
```

In [25]:

```
#Checking for NULL values

big_df['Route_2'].isna().sum()
```

Out[25]:

```
1
```

In [26]:

```
#Checking for NULL values

big_df['Route_3'].isna().sum()
```

Out[26]:

```
4341
```

In [27]:

```
#Checking for NULL values

big_df['Route_4'].isna().sum()
```

Out[27]:

```
11397
```

In [28]:

```
#Checking for NULL values

big_df['Route_5'].isna().sum()
```

Out[28]:

```
13296
```

In [29]:

```
#Replacing NaN with None in each Route columns to get rid of NULL values

big_df['Route_1'].fillna("None", inplace = True)
big_df['Route_2'].fillna("None", inplace = True)
big_df['Route_3'].fillna("None", inplace = True)
big_df['Route_4'].fillna("None", inplace = True)
big_df['Route_5'].fillna("None", inplace = True)
```

In [30]:

```
#Checking for NULL values in 'Price' column

big_df['Price'].isna().sum()
```

Out[30]:

```
2671
```

In [31]:

```
#Imputing the mean value of price to all NULL values

big_df['Price'].fillna((big_df['Price'].mean()), inplace = True)
```

In [32]:

```
#Printing out the 'Price' column after imputing mean

big_df['Price']
```

Out[32]:

```
0        3897.000000
1        7662.000000
2       13882.000000
3        6218.000000
4       13302.000000
            ...
2666     9087.064121
2667     9087.064121
2668     9087.064121
2669     9087.064121
2670     9087.064121
Name: Price, Length: 13354, dtype: float64
```

In [33]:

```
#Used the decribe function to print out summary of statistics

big_df.describe()
```

Out[33]:

	Price	Date	Month	Year	Arrival_Hour	Arrival_Minute	
count	13354.000000	13354.000000	13354.000000	13354.0	13354.000000	13354.000000	133
mean	9087.064121	13.389846	4.710574	2019.0	13.396061	24.664146	
std	4124.447805	8.439060	1.165622	0.0	6.896145	16.559723	
min	1759.000000	1.000000	3.000000	2019.0	0.000000	0.000000	
25%	6135.250000	6.000000	3.000000	2019.0	8.000000	10.000000	
50%	9087.064121	12.000000	5.000000	2019.0	14.000000	25.000000	
75%	11087.000000	21.000000	6.000000	2019.0	19.000000	35.000000	
max	79512.000000	27.000000	6.000000	2019.0	23.000000	55.000000	

In [34]:

```
#Dropping out the old 'Route' and 'Duration' column

big_df = big_df.drop(['Route'], axis = 1)
big_df = big_df.drop(['Duration'], axis = 1)
```

In [35]:

```
#Preparing categorical variables for model using label encoder
#To convert categorical text data into model-understandable numerical data, we use the L

from sklearn.preprocessing import LabelEncoder

lb_encode = LabelEncoder()
big_df['Additional_Info'] = lb_encode.fit_transform(big_df['Additional_Info'])
big_df['Airline'] = lb_encode.fit_transform(big_df['Airline'])
big_df['Destination'] = lb_encode.fit_transform(big_df['Destination'])
big_df['Source'] = lb_encode.fit_transform(big_df['Source'])
big_df['Route_1'] = lb_encode.fit_transform(big_df['Route_1'])
big_df['Route_2'] = lb_encode.fit_transform(big_df['Route_2'])
big_df['Route_3'] = lb_encode.fit_transform(big_df['Route_3'])
big_df['Route_4'] = lb_encode.fit_transform(big_df['Route_4'])
big_df['Route_5'] = lb_encode.fit_transform(big_df['Route_5'])
```

In [36]:

```
#Finally checking the datatypes of each column

big_df.dtypes
```

Out[36]:

```
Airline              int32
Source               int32
Destination          int32
Additional_Info      int32
Price              float64
Date                 int32
Month                int32
Year                 int32
Arrival_Hour         int32
Arrival_Minute       int32
Stop                 int32
Dep_Hour             int32
Dep_Minute           int32
Route_1              int32
Route_2              int32
Route_3              int32
Route_4              int32
Route_5              int32
dtype: object
```

In [37]:

```
#Dividing the data into train and test

df_train = big_df[0:10683]
df_test = big_df[10683:]
```

In [38]:

```
#Dropping the 'Price' column from the test dataset

df_test = df_test.drop(['Price'], axis = 1)
```

In [39]:

```
#Dividing df_train to X and y respectively

X = df_train.drop(['Price'], axis = 1)
y = df_train.Price
```

In [40]:

```
#Importing model libraries for Machine Learning models

from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
```

In [41]:

```
#Assigning each to a variable

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

In [42]:

```
#Spliting the data by 70-30

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state
```

In [43]:

```
#Running for loop to fit and predict each model

for i in [dt, svr, knn, lr]:
    i.fit(X_train, y_train)
    pred = i.predict(X_test)
    test_score = r2_score(y_test, 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, pred))
    print('R2 for train data is', r2_score(y_train, i.predict(X_train)))
    print('Mean Absolute Error is', mean_absolute_error(y_test, pred))
    print('Mean Squared Error is', mean_squared_error(y_test, pred))
    print('Root Mean Squared Error is', (mean_squared_error(y_test, pred, squared = Fals
    print('-----------------------------')
```

```
DecisionTreeRegressor()
R2 score is 0.8201960872027666
R2 for train data is 0.9965922757175513
Mean Absolute Error is 731.1658346333853
Mean Squared Error is 3869196.261544462
Root Mean Squared Error is 1967.0272650739903
-----------------------------
SVR()
R2 score is -0.021465086494998342
R2 for train data is -0.025079144152643 45
Mean Absolute Error is 3649.1827690313457
Mean Squared Error is 21980883.68867493
Root Mean Squared Error is 4688.377511322539
-----------------------------
KNeighborsRegressor()
R2 score is 0.6570257833107811
R2 for train data is 0.7847906002964877
Mean Absolute Error is 1621.539282371295
Mean Squared Error is 7380454.2758564735
Root Mean Squared Error is 2716.6991507814173
-----------------------------
LinearRegression()
R2 score is 0.4978069836098046
R2 for train data is 0.49371557019830536
Mean Absolute Error is 2309.7338089909376
Mean Squared Error is 10806679.962420585
Root Mean Squared Error is 3287.3515118436276
-----------------------------
```

In [44]:

```
#Importing ensembling model libraries for Machine Learning model

from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor, GradientBoostingR
```

In [45]:

```
#Assigning each to a variable

rfr = RandomForestRegressor()
abr = AdaBoostRegressor()
gbr = GradientBoostingRegressor()
```

In [46]:

```
#Spliting the data by 70-30

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state
```

In [47]:

```
#Running for loop to fit and predict each model

for i in [rfr, abr, gbr]:
    i.fit(X_train, y_train)
    pred = i.predict(X_test)
    test_score = r2_score(y_test, 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, pred))
    print('R2 for train data is', r2_score(y_train, i.predict(X_train)))
    print('Mean Absolute Error is', mean_absolute_error(y_test, pred))
    print('Mean Squared Error is', mean_squared_error(y_test, pred))
    print('Root Mean Squared Error is', (mean_squared_error(y_test, pred, squared = Fals
    print('-----------------------------')
```

```
RandomForestRegressor()
R2 score is 0.9058344201397822
R2 for train data is 0.9795210460193386
Mean Absolute Error is 597.9978793056484
Mean Squared Error is 1900286.2936576195
Root Mean Squared Error is 1378.5087209218589
-----------------------------
AdaBoostRegressor()
R2 score is 0.47342407066518677
R2 for train data is 0.5398689292845237
Mean Absolute Error is 2557.729929211298
Mean Squared Error is 10626441.45100956
Root Mean Squared Error is 3259.8223035941023
-----------------------------
GradientBoostingRegressor()
R2 score is 0.8323481943311676
R2 for train data is 0.8478499406895647
Mean Absolute Error is 1205.400261375095
Mean Squared Error is 3383257.756096786
Root Mean Squared Error is 1839.3634105572467
-----------------------------
```

In [48]:

```
# Performing Cross Validation to remove overfitting, if any
#Importing cross_val_score library for Cross Validation

from sklearn.model_selection import cross_val_score
```

In [49]:

```
#Running for Loop for CV from 2 to 9
#This is done on Random Forest Regressor

for i in range(2,9):
    cv = cross_val_score(rfr, X, y, cv = i)
    print(rfr, cv.mean())
```

```
RandomForestRegressor() 0.8755203745091795
RandomForestRegressor() 0.8776331830603482
RandomForestRegressor() 0.8921357849302178
RandomForestRegressor() 0.8972935861723086
RandomForestRegressor() 0.89772600029634
RandomForestRegressor() 0.899954682738915
RandomForestRegressor() 0.8990727592427457
```

In [50]:

```
#Running for Loop for CV from 2 to 9
#This is done on Gradient Boosting Regressor

for i in range(2,9):
    cv = cross_val_score(gbr, X, y, cv = i)
    print(gbr, cv.mean())
```

```
GradientBoostingRegressor() 0.8237169365051635
GradientBoostingRegressor() 0.8268177381721635
GradientBoostingRegressor() 0.8278487010121367
GradientBoostingRegressor() 0.8289979222499226
GradientBoostingRegressor() 0.8275441903298814
GradientBoostingRegressor() 0.8292722192488572
GradientBoostingRegressor() 0.8296008517589071
```

In [51]:

```
#Hypertuning the model using GridSearchCV

from sklearn.model_selection import GridSearchCV
```

In [52]:

```
#GridSearchCV for Random Forest Regressor

param_grid = {'n_estimators': [10,30,50,70,100], 'max_depth': [None, 1, 2, 3], 'max_samp
              'min_samples_split': [2,4,10]}

gscv_rfr = GridSearchCV(rfr, param_grid, cv = 3)
```

In [53]:

```
# Fitting the model

res = gscv_rfr.fit(X_train, y_train)
```

In [54]:

```
# Checking out the best parameter

res.best_params_
```

Out[54]:

```
{'max_depth': None,
 'max_samples': 1000,
 'min_samples_split': 2,
 'n_estimators': 50}
```

In [55]:

```
#Printing out the result achieved by the best parameter

res.best_score_
```

Out[55]:

```
0.8191071306945039
```

In [56]:

```
#GridSearchCV for Gradient Boosting Regressor

param_grid_2 = {'alpha': [0.9,0.09,0.1], 'learning_rate':[0.1,0.01], 'max_depth': [3,4,5
               'min_samples_split': [2,3,4], 'n_estimators': [100,50,10]}
```

In [57]:

```
gscv_gbr = GridSearchCV(gbr, param_grid_2, cv=3)
```

In [58]:

```
#Fitiing the model

res2 = gscv_gbr.fit(X_train, y_train)
```

In [59]:

```
#Checking out the best parameter

res2.best_params_
```

Out[59]:

```
{'alpha': 0.9,
 'learning_rate': 0.1,
 'max_depth': 5,
 'min_samples_leaf': 1,
 'min_samples_split': 4,
 'n_estimators': 100}
```

In [60]:

```
#Printing out the result achieved by the best parameter

res2.best_score_
```

Out[60]:

```
0.8453797021053303
```

In [61]:

```
#We got our best model with a score of 84.5% (Gradient Boosting Regressor)
#Saving it using best parameter and creating a model object using joblib

model = GradientBoostingRegressor(alpha = 0.9, learning_rate = 0.1, max_depth = 5, min_s
                                  n_estimators = 100)
```

In [62]:

```
#Importing joblib library and storing our model in object named flight_price.obj

import joblib

joblib.dump(model, 'flight_price.obj')
```

Out[62]:

```
['flight_price.obj']
```

In [65]:

```
#Predicting X_test on our best model and printing out the results side by side

model = joblib.load('flight_price.obj')
model.fit(X_train, y_train)
pred = model.predict(X_test)

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

Out[65]:

	Actual	Predicted
6041	3597.0	3804.404450
5637	3383.0	4020.716200
9644	2050.0	3231.028773
3159	4423.0	4207.168513
5278	3597.0	4061.426159
...	...	...
6821	13731.0	13882.662538
10518	9416.0	10618.466659
397	3597.0	3783.702023
9847	5277.0	5238.228357
1453	11410.0	9604.898594

3205 rows × 2 columns

In [69]:

```
#Predicting our best model on the new data (test data)

flight_price = joblib.load('flight_price.obj')
flight_price.fit(X_train, y_train)
prices = flight_price.predict(df_test)
```

In [70]:

```
#Printing out the predicted values

prices
```

Out[70]:

```
array([14059.536814  ,  5470.02920794, 11532.88225686, ...,
       16923.35919605, 14158.48020774,  9226.63359671])
```

In [71]:

```
#Converting the predicted values (array) into a dataframe

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

Out[71]:

	Price
0	14059.536814
1	5470.029208
2	11532.882257
3	10684.572294
4	3736.203885
...	...
2666	9405.328788
2667	4820.390114
2668	16923.359196
2669	14158.480208
2670	9226.633597

2671 rows × 1 columns