air_fare

February 27, 2022

1 Library

[1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
```

2 Data Sets

[2]:
```
train_data=pd.read_excel('Data_Train.xlsx')
test_data=pd.read_excel('Test_set.xlsx')
```

[3]:
```
train_data.shape,test_data.shape
```

[3]:
```
((10683, 11), (2671, 10))
```

[4]:
```
train_data.head()
```

[4]:

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

[5]:
```
train_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
```

[6]:
```
test_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2671 entries, 0 to 2670
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Airline          2671 non-null   object
 1   Date_of_Journey  2671 non-null   object
 2   Source           2671 non-null   object
 3   Destination      2671 non-null   object
 4   Route            2671 non-null   object
 5   Dep_Time         2671 non-null   object
 6   Arrival_Time     2671 non-null   object
 7   Duration         2671 non-null   object
 8   Total_Stops      2671 non-null   object
 9   Additional_Info  2671 non-null   object
dtypes: object(10)
memory usage: 208.8+ KB
```

[7]:
```
# Checking missing value in dataset
train_data.isnull().values.any(),test_data.isnull().values.any()
```

[7]:
```
(True, False)
```

[8]:
```
train_data.isnull().sum()
```

[8]:
```
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
```

[9]:
```
# Train data null value dopped
train_data.dropna(inplace=True)
```

3 Data Cleaning

[10]:
```
# Checking if there are any Duplicate values
train_data[train_data.duplicated()]
```

[10]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price
683	Jet Airways	1/06/2019	Delhi	Cochin	DEL → NAG → BOM → COK	14:35	04:25 02 Jun	13h 50m	2 stops	No info	13376
1061	Air India	21/05/2019	Delhi	Cochin	DEL → GOI → BOM → COK	22:00	19:15 22 May	21h 15m	2 stops	No info	10231
1348	Air India	18/05/2019	Delhi	Cochin	DEL → HYD → BOM → COK	17:15	19:15 19 May	26h	2 stops	No info	13376
1418	Jet Airways	6/06/2019	Delhi	Cochin	DEL → JAI → BOM → COK	05:30	04:25 07 Jun	22h 55m	2 stops	No info	13376
1674	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	18:25	21:20	2h 55m	non-stop	No info	3873
…	…	…	…	…	…	…	…	…	…	…	…
10594	Jet Airways	27/06/2019	Delhi	Cochin	DEL → AMD → BOM → COK	23:05	12:35 28 Jun	13h 30m	2 stops	No info	13376
10616	Jet Airways	1/06/2019	Delhi	Cochin	DEL → JAI → BOM → COK	09:40	12:35 02 Jun	26h 55m	2 stops	No info	13376
10634	Jet Airways	6/06/2019	Delhi	Cochin	DEL → JAI → BOM → COK	09:40	12:35 07 Jun	26h 55m	2 stops	No info	13376
10672	Jet Airways	27/06/2019	Delhi	Cochin	DEL → AMD → BOM → COK	23:05	19:00 28 Jun	19h 55m	2 stops	No info	13376
10673	Jet Airways	27/05/2019	Delhi	Cochin	DEL → AMD → BOM → COK	13:25	04:25 28 May	15h	2 stops	No info	13376

```
1348                         No info  12392
1418   In-flight meal not included  10368
1674                         No info   7303
…                               …    …
10594                        No info  12819
10616                        No info  13014
10634  In-flight meal not included  11733
10672  In-flight meal not included  11150
10673                        No info  16704

[220 rows x 11 columns]
```

[11]:
```
# Drop duplicates value
train_data.drop_duplicates(keep='first',inplace=True)
```

[12]:
```
train_data["Additional_Info"].value_counts()
```

[12]:
```
No info                         8182
In-flight meal not included     1926
No check-in baggage included     318
1 Long layover                    19
Change airports                    7
Business class                     4
No Info                            3
Red-eye flight                     1
1 Short layover                    1
2 Long layover                     1
Name: Additional_Info, dtype: int64
```

[13]:
```
# Convert No Info in No info because both are same
train_data["Additional_Info"] = train_data["Additional_Info"].replace({'No␣
↪Info': 'No info'})
```

4 Feature Engineering (Dividing data into features and labels)

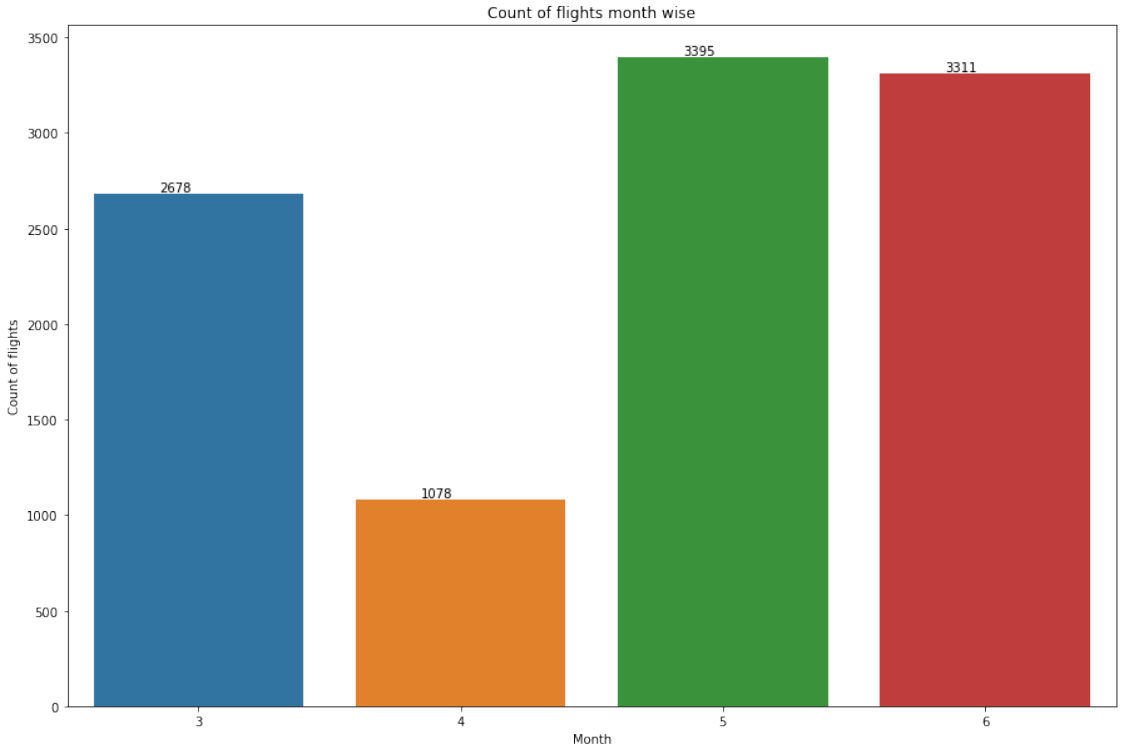
[14]:
```
train_data['Duration']=  train_data['Duration'].str.replace("h", '*60').str.
↪replace(' ','+').str.replace('m','*1').apply(eval)
test_data['Duration']=  test_data['Duration'].str.replace("h", '*60').str.
↪replace(' ','+').str.replace('m','*1').apply(eval)
```

[15]:
```
# Date_of_Journey
train_data["Journey_day"] = train_data['Date_of_Journey'].str.split('/').str[0].
↪astype(int)
train_data["Journey_month"] = train_data['Date_of_Journey'].str.split('/').
↪str[1].astype(int)
train_data.drop(["Date_of_Journey"], axis = 1, inplace = True)
```

```
# Dep_Time
train_data["Dep_hour"] = pd.to_datetime(train_data["Dep_Time"]).dt.hour
train_data["Dep_min"] = pd.to_datetime(train_data["Dep_Time"]).dt.minute
train_data.drop(["Dep_Time"], axis = 1, inplace = True)

# Arrival_Time
train_data["Arrival_hour"] = pd.to_datetime(train_data.Arrival_Time).dt.hour
train_data["Arrival_min"] = pd.to_datetime(train_data.Arrival_Time).dt.minute
train_data.drop(["Arrival_Time"], axis = 1, inplace = True)
```

[16]:
```
plt.figure(figsize = (15, 10))
plt.title('Count of flights month wise')
ax=sns.countplot(x = 'Journey_month', data = train_data)
plt.xlabel('Month')
plt.ylabel('Count of flights')
for p in ax.patches:
    ax.annotate(int(p.get_height()), (p.get_x()+0.25, p.get_height()+1),␣
 ↪va='bottom', color= 'black')
```



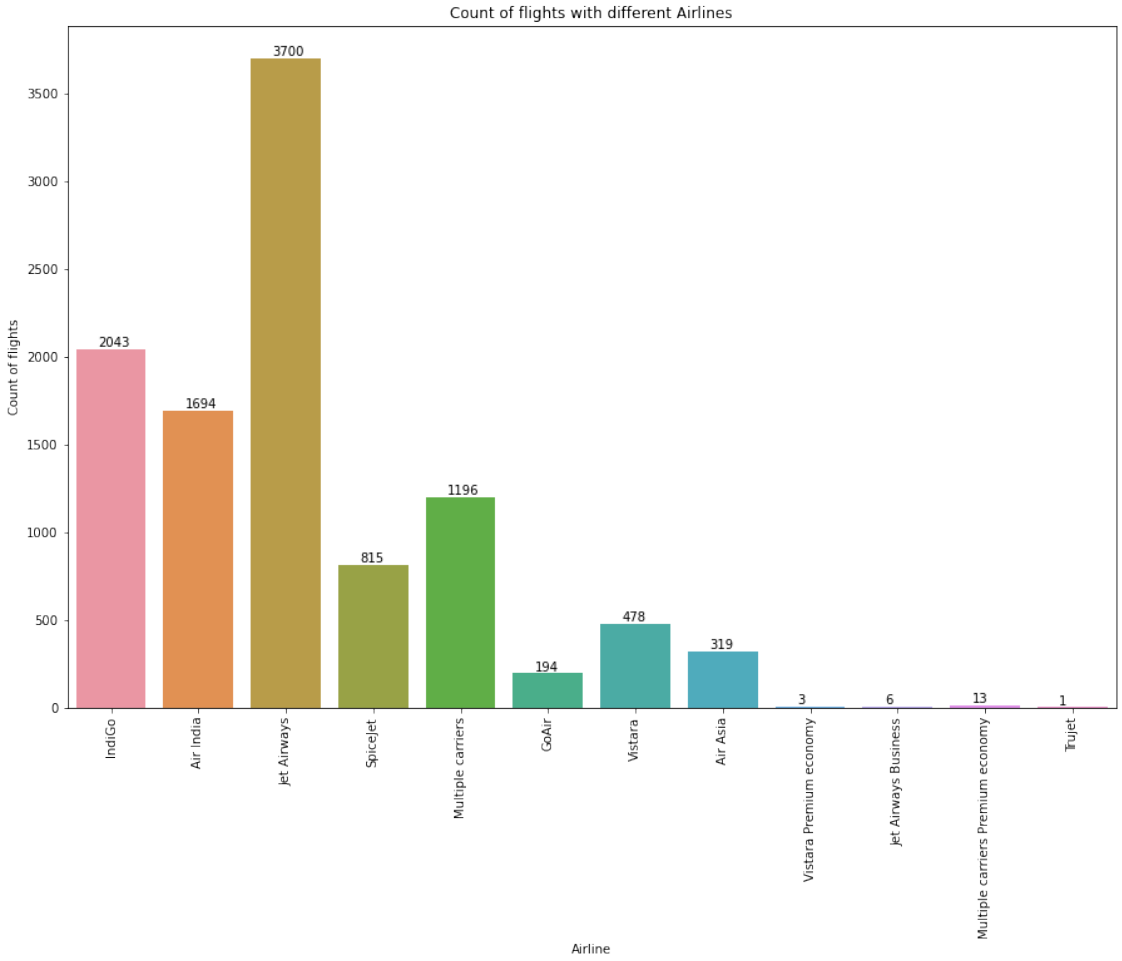
[17]:
```
# Total_Stops
train_data['Total_Stops'].replace(['1 stop', 'non-stop', '2 stops', '3 stops',␣
 ↪'4 stops'], [1, 0, 2, 3, 4], inplace=True)
```

```
test_data['Total_Stops'].replace(['1 stop', 'non-stop', '2 stops', '3 stops',␣
↪'4 stops'], [1, 0, 2, 3, 4], inplace=True)
```

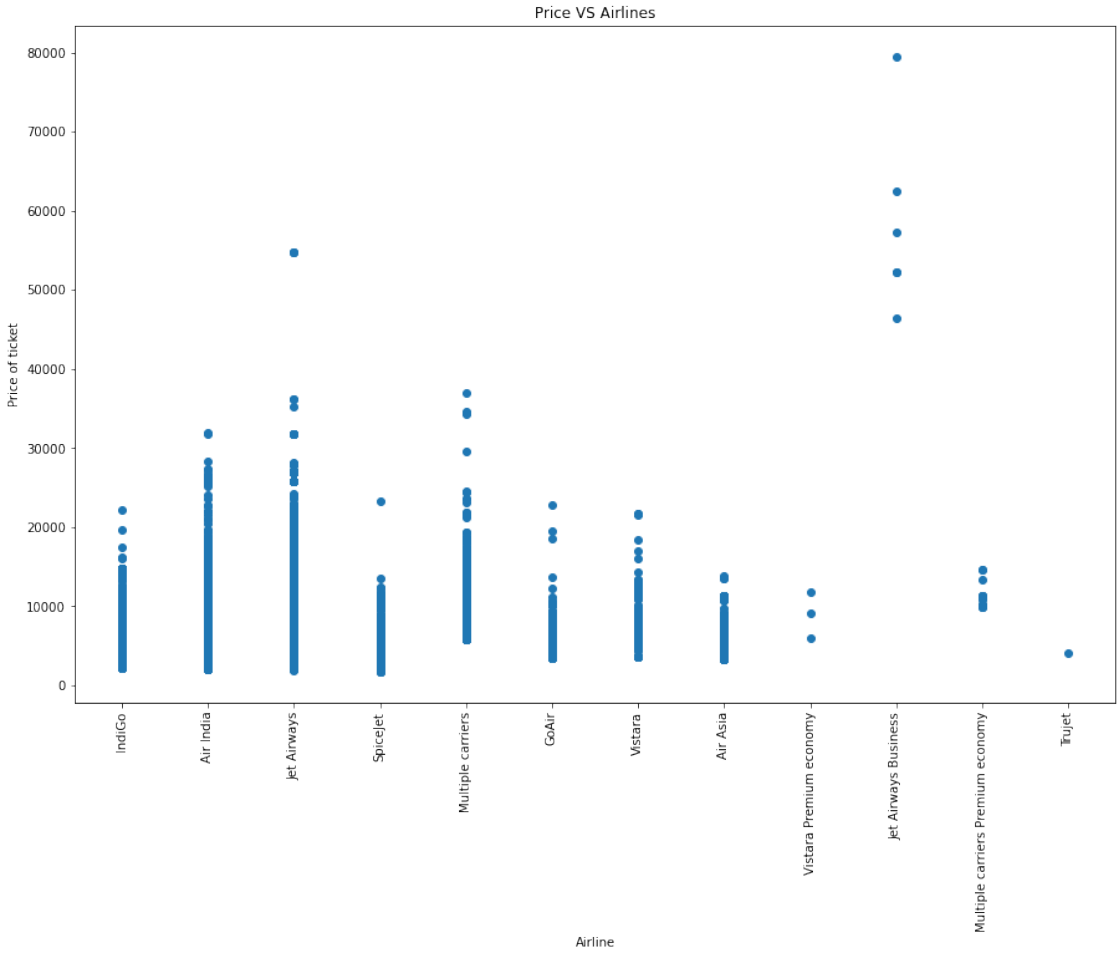
[18]:
```
train_data["Airline"].value_counts()
```

[18]:
```
Jet Airways                          3700
IndiGo                               2043
Air India                            1694
Multiple carriers                    1196
SpiceJet                              815
Vistara                               478
Air Asia                              319
GoAir                                 194
Multiple carriers Premium economy      13
Jet Airways Business                    6
Vistara Premium economy                 3
Trujet                                  1
Name: Airline, dtype: int64
```

[19]:
```
plt.figure(figsize = (15, 10))
plt.title('Count of flights with different Airlines')
ax=sns.countplot(x = 'Airline', data =train_data)
plt.xlabel('Airline')
plt.ylabel('Count of flights')
plt.xticks(rotation = 90)
for p in ax.patches:
    ax.annotate(int(p.get_height()), (p.get_x()+0.25, p.get_height()+1),␣
↪va='bottom', color= 'black')
```



[20]:
```
plt.figure(figsize = (15, 10))
plt.title('Price VS Airlines')
plt.scatter(train_data['Airline'], train_data['Price'])
plt.xticks(rotation = 90)
plt.xlabel('Airline')
plt.ylabel('Price of ticket')
plt.xticks(rotation = 90);
```



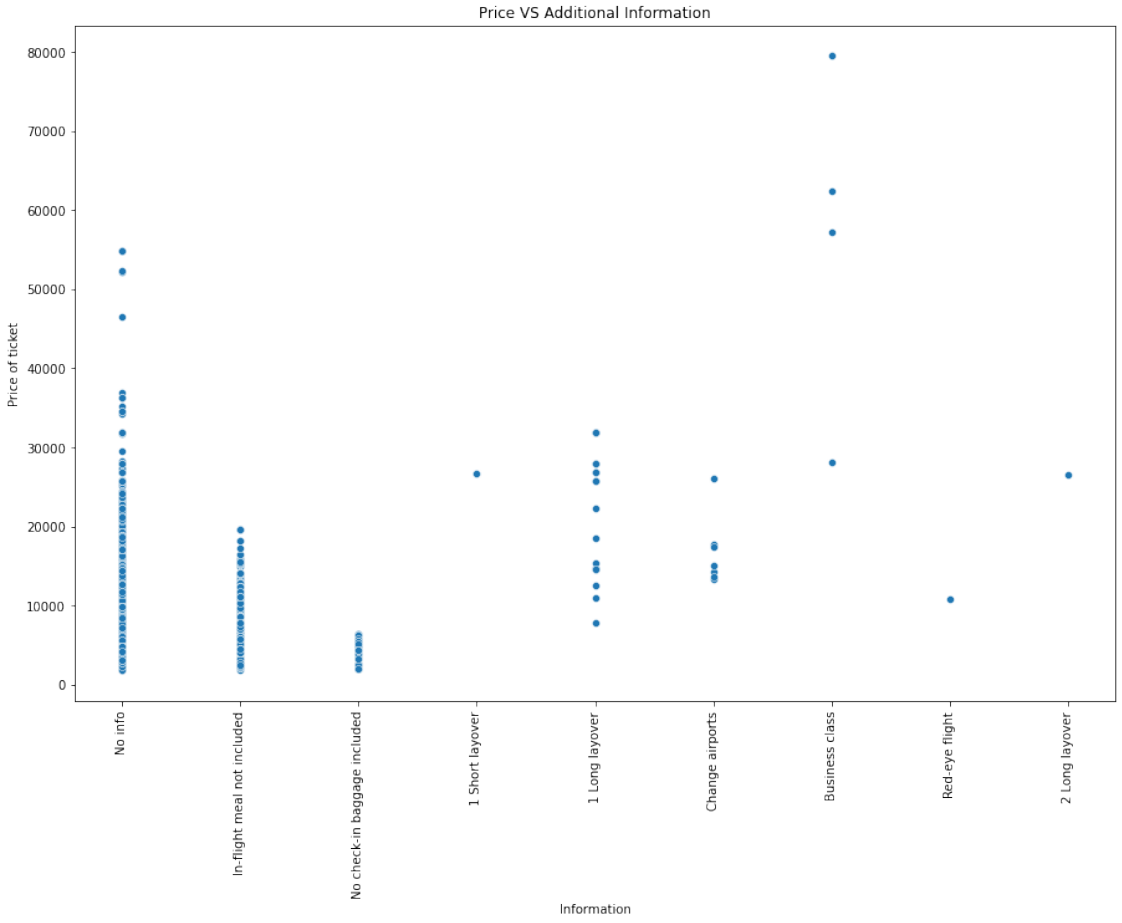
[21]:
```
# Airline
train_data["Airline"].replace({'Multiple carriers Premium economy':'Other',
                                                    'Jet Airways Business':
 ↪'Other',
                                                    'Vistara Premium␣
 ↪economy':'Other',
                                                    'Trujet':'Other'
                                                },
                                          inplace=True)

test_data["Airline"].replace({'Multiple carriers Premium economy':'Other',
                                                    'Jet Airways Business':
 ↪'Other',
                                                    'Vistara Premium␣
 ↪economy':'Other',
                                                    'Trujet':'Other'
                                                },
```

```
                                                inplace=True)
```

[22]:
```
plt.figure(figsize = (15, 10))
plt.title('Price VS Additional Information')
sns.scatterplot(train_data['Additional_Info'],␣
 ↪train_data['Price'],data=train_data)
plt.xticks(rotation = 90)
plt.xlabel('Information')
plt.ylabel('Price of ticket')
```

[22]:
```
Text(0, 0.5, 'Price of ticket')
```



[23]:
```
train_data["Additional_Info"].value_counts()
```

[23]:
```
No info                         8185
In-flight meal not included     1926
No check-in baggage included     318
1 Long layover                    19
```

```
Change airports                    7
Business class                     4
Red-eye flight                     1
1 Short layover                    1
2 Long layover                     1
Name: Additional_Info, dtype: int64
```

[24]:
```
# Additional_Info
train_data["Additional_Info"].replace({'Change airports':'Other',
                                       'Business class':
 ↪'Other',
                                       '1 Short layover':
 ↪'Other',
                                       'Red-eye flight':
 ↪'Other',
                                       '2 Long layover':
 ↪'Other',
                                      }, inplace=True)
test_data["Additional_Info"].replace({'Change airports':'Other',
                                       'Business class':
 ↪'Other',
                                       '1 Short layover':
 ↪'Other',
                                       'Red-eye flight':
 ↪'Other',
                                       '2 Long layover':
 ↪'Other',
                                      }, inplace=True)
```

[25]:
```
train_data.head()
```

[25]:

	Airline	Source	Destination	Route	Duration	Total_Stops	Additional_Info	Price	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	Arrival_min
0	IndiGo	Banglore	New Delhi	BLR → DEL	170	0	No info	3897	24	3	22	20	1	10
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	445	2	No info	7662	1	5	5			
2	Jet Airways	Delhi	Cochin	DEL → LKO → BOM → COK	1140	2	No info	13882	9	6	9			
3	IndiGo	Kolkata	Banglore	CCU → NAG → BLR	325	1	No info	6218	12	5	18			
4	IndiGo	Banglore	New Delhi	BLR → NAG → DEL	285	1	No info	13302	1	3	16			

1	50	13	15
2	25	4	25
3	5	23	30
4	50	21	35

5 Convert categorical data into numerical

[26]:
```
data = train_data.drop(["Price"], axis=1)
```

[27]:
```
train_categorical_data = data.select_dtypes(exclude=['int64', 'float','int32'])
train_numerical_data = data.select_dtypes(include=['int64', 'float','int32'])

test_categorical_data = test_data.select_dtypes(exclude=['int64',␣
↪'float','int32','int32'])
test_numerical_data  = test_data.select_dtypes(include=['int64',␣
↪'float','int32'])
```

[28]:
```
train_categorical_data.head()
```

[28]:

	Airline	Source	Destination	Route	Additional_Info
0	IndiGo	Banglore	New Delhi	BLR → DEL	No info
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	No info
2	Jet Airways	Delhi	Cochin	DEL → LKO → BOM → COK	No info
3	IndiGo	Kolkata	Banglore	CCU → NAG → BLR	No info
4	IndiGo	Banglore	New Delhi	BLR → NAG → DEL	No info

[29]:
```
#Label encode and hot encode categorical columns
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
train_categorical_data = train_categorical_data.apply(LabelEncoder().
↪fit_transform)
test_categorical_data = test_categorical_data.apply(LabelEncoder().
↪fit_transform)
```

[30]:
```
train_categorical_data.head()
```

[30]:

	Airline	Source	Destination	Route	Additional_Info
0	3	0	5	18	3
1	1	3	0	84	3
2	4	2	1	118	3
3	3	3	0	91	3
4	3	0	5	29	3

6 Concatenate both catagorical and numerical data

[31]:
```
X = pd.concat([train_categorical_data, train_numerical_data], axis=1)
y=train_data['Price']
test_set = pd.concat([test_categorical_data, test_numerical_data], axis=1)
```

[32]:
```
X.head()
```

[32]:

	Airline	Source	Destination	Route	Additional_Info	Duration	\
0	3	0	5	18	3	170	
1	1	3	0	84	3	445	
2	4	2	1	118	3	1140	
3	3	3	0	91	3	325	
4	3	0	5	29	3	285	

	Total_Stops	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	\
0	0	24	3	22	20	1	
1	2	1	5	5	50	13	
2	2	9	6	9	25	4	
3	1	12	5	18	5	23	
4	1	1	3	16	50	21	

	Arrival_min
0	10
1	15
2	25
3	30
4	35

[33]:
```
y.head()
```

[33]:
```
0     3897
1     7662
2    13882
3     6218
4    13302
Name: Price, dtype: int64
```

7 Building Machine Learning Models

[34]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import r2_score

from math import sqrt
```

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

from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.model_selection import KFold

def mean_absolute_percentage_error(y_true, y_pred):
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
```

[35]:
```
# training testing and splitting the dataset
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3,␣
 ↪random_state = 42)
```

[36]:
```
print("The size of training input is", X_train.shape)
print("The size of training output is", y_train.shape)
print(50 *'*')
print("The size of testing input is", X_test.shape)
print("The size of testing output is", y_test.shape)
```

```
The size of training input is (7323, 13)
The size of training output is (7323,)
**************************************************
The size of testing input is (3139, 13)
The size of testing output is (3139,)
```

8 Ridge

[37]:
```
params ={'alpha' :[0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000, 100000]}
ridge_regressor =GridSearchCV(Ridge(), params ,cv =5,scoring =␣
 ↪'neg_mean_absolute_error', n_jobs =-1)
ridge_regressor.fit(X_train ,y_train)
```

[37]:
```
GridSearchCV(cv=5, estimator=Ridge(), n_jobs=-1,
             param_grid={'alpha': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000,
                                   10000, 100000]},
             scoring='neg_mean_absolute_error')
```

[38]:
```
y_train_pred =ridge_regressor.predict(X_train) ##Predict train result
y_test_pred =ridge_regressor.predict(X_test) ##Predict test result
```

[39]:
```
print("Train Results for Ridge Regressor Model:")
print(50 * '-')
print("Root mean squared error: ", sqrt(mse(y_train.values, y_train_pred)))
print("Mean absolute % error: ", round(mean_absolute_percentage_error(y_train.
 ↪values, y_train_pred)))
print("R-squared: ", r2_score(y_train.values, y_train_pred))
```

```
Train Results for Ridge Regressor Model:
--------------------------------------------------
Root mean squared error:  3476.888067395797
Mean absolute % error:  32
R-squared:  0.4416286780338675
```

[40]:
```
print("Test Results for Ridge Regressor Model:")
print(50 * '-')
print("Root mean squared error: ", sqrt(mse(y_test, y_test_pred)))
print("Mean absolute % error: ", round(mean_absolute_percentage_error(y_test,␣
 ↪y_test_pred)))
print("R-squared: ", r2_score(y_test, y_test_pred))
```

```
Test Results for Ridge Regressor Model:
--------------------------------------------------
Root mean squared error:  3381.824795655435
Mean absolute % error:  32
R-squared:  0.4493252066014467
```

9 Lasso

[41]:
```
params ={'alpha' :[0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000, 100000]}
lasso_regressor =GridSearchCV(Lasso(), params ,cv =15,scoring =␣
 ↪'neg_mean_absolute_error', n_jobs =-1)
lasso_regressor.fit(X_train ,y_train)
```

[41]:
```
GridSearchCV(cv=15, estimator=Lasso(), n_jobs=-1,
             param_grid={'alpha': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000,
                                   10000, 100000]},
             scoring='neg_mean_absolute_error')
```

[42]:
```
y_train_pred =lasso_regressor.predict(X_train) ##Predict train result
y_test_pred =lasso_regressor.predict(X_test) ##Predict test result
```

[43]:
```
print("Train Results for Lasso Regressor Model:")
print(50 * '-')
print("Root mean squared error: ", sqrt(mse(y_train.values, y_train_pred)))
print("Mean absolute % error: ", round(mean_absolute_percentage_error(y_train.
 ↪values, y_train_pred)))
print("R-squared: ", r2_score(y_train.values, y_train_pred))
```

```
Train Results for Lasso Regressor Model:
--------------------------------------------------
Root mean squared error:  3488.481565031116
Mean absolute % error:  32
R-squared:  0.43789875103811027
```

[44]:
```
print("Test Results for Lasso Regressor Model:")
print(50 * '-')
print("Root mean squared error: ", sqrt(mse(y_test, y_test_pred)))
print("Mean absolute % error: ", round(mean_absolute_percentage_error(y_test,␣
 ↪y_test_pred)))
print("R-squared: ", r2_score(y_test, y_test_pred))
```

```
Test Results for Lasso Regressor Model:
--------------------------------------------------
Root mean squared error:  3389.213848435348
Mean absolute % error:  32
R-squared:  0.44691620521166686
```

10 K Neighbors Regressor

[45]:
```
k_range = list(range(1, 30))
params = dict(n_neighbors = k_range)
knn_regressor = GridSearchCV(KNeighborsRegressor(), params, cv =10, scoring =␣
 ↪'neg_mean_squared_error')
knn_regressor.fit(X_train, y_train)
```

[45]:
```
GridSearchCV(cv=10, estimator=KNeighborsRegressor(),
             param_grid={'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
                                         13, 14, 15, 16, 17, 18, 19, 20, 21, 22,
                                         23, 24, 25, 26, 27, 28, 29]},
             scoring='neg_mean_squared_error')
```

[46]:
```
y_train_pred =knn_regressor.predict(X_train) ##Predict train result
y_test_pred =knn_regressor.predict(X_test) ##Predict test result
```

[47]:
```
print("Train Results for KNN Regressor Model:")
print(50 * '-')
print("Root mean squared error: ", sqrt(mse(y_train.values, y_train_pred)))
print("Mean absolute % error: ", round(mean_absolute_percentage_error(y_train.
 ↪values, y_train_pred)))
print("R-squared: ", r2_score(y_train.values, y_train_pred))
```

```
Train Results for KNN Regressor Model:
--------------------------------------------------
Root mean squared error:  2325.6410475460157
Mean absolute % error:  14
R-squared:  0.7501800863194169
```

[48]:
```
print("Test Results for KNN Regressor Model:")
print(50 * '-')
print("Root mean squared error: ", sqrt(mse(y_test, y_test_pred)))
print("Mean absolute % errorr: ", round(mean_absolute_percentage_error(y_test,␣
 ↪y_test_pred)))
print("R-squared: ", r2_score(y_test, y_test_pred))
```

```
Test Results for KNN Regressor Model:
--------------------------------------------------
Root mean squared error:  3020.1949507075774
Mean absolute % errorr:  20
R-squared:  0.5607993808307774
```

11 Decision Tree Regressor

[49]:
```
depth  =list(range(3,30))
param_grid =dict(max_depth =depth)
tree =GridSearchCV(DecisionTreeRegressor(),param_grid,cv =10)
tree.fit(X_train,y_train)
```

[49]:
```
GridSearchCV(cv=10, estimator=DecisionTreeRegressor(),
             param_grid={'max_depth': [3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
                                       15, 16, 17, 18, 19, 20, 21, 22, 23, 24,
                                       25, 26, 27, 28, 29]})
```

[50]:
```
y_train_pred =tree.predict(X_train) ##Predict train result
y_test_pred =tree.predict(X_test) ##Predict test result
```

[61]:
```
print("Train Results for Decision Tree Regressor Model:")
print(50 * '-')
print("Root mean squared error: ", sqrt(mse(y_train.values, y_train_pred)))
print("Mean absolute % error: ", round(mean_absolute_percentage_error(y_train.
 ↪values, y_train_pred)))
print("R-squared: ", r2_score(y_train.values, y_train_pred))
```

```
Train Results for Decision Tree Regressor Model:
--------------------------------------------------
Root mean squared error:  707.1638519523252
Mean absolute % error:  6
R-squared:  0.9769016056692901
```

[52]:
```
print("Test Results for Decision Tree Regressor Model:")
print(50 * '-')
print("Root mean squared error: ", sqrt(mse(y_test, y_test_pred)))
print("Mean absolute % error: ", round(mean_absolute_percentage_error(y_test,␣
 ↪y_test_pred)))
print("R-squared: ", r2_score(y_test, y_test_pred))
```

```
Test Results for Decision Tree Regressor Model:
--------------------------------------------------
Root mean squared error:  2158.774763422783
Mean absolute % error:  10
R-squared:  0.7756078528585771
```

12 Random Forest Regressor

[53]:
```
tuned_params = {'n_estimators': [100, 200, 300, 400, 500], 'min_samples_split':␣
↪[2, 5, 10], 'min_samples_leaf': [1, 2, 4]}
random_regressor = RandomizedSearchCV(RandomForestRegressor(), tuned_params,␣
↪n_iter = 20, scoring = 'neg_mean_absolute_error', cv = 5, n_jobs = -1)
random_regressor.fit(X_train, y_train)
```

[53]:
```
RandomizedSearchCV(cv=5, estimator=RandomForestRegressor(), n_iter=20,
                   n_jobs=-1,
                   param_distributions={'min_samples_leaf': [1, 2, 4],
                                        'min_samples_split': [2, 5, 10],
                                        'n_estimators': [100, 200, 300, 400,
                                                         500]},
                   scoring='neg_mean_absolute_error')
```

[54]:
```
y_train_pred = random_regressor.predict(X_train)
y_test_pred = random_regressor.predict(X_test)
```

[55]:
```
print("Train Results for Random Forest Regressor Model:")
print(50 * '-')
print("Root mean squared error: ", sqrt(mse(y_train.values, y_train_pred)))
print("Mean absolute % error: ", round(mean_absolute_percentage_error(y_train.
↪values, y_train_pred)))
print("R-squared: ", r2_score(y_train.values, y_train_pred))
```

```
Train Results for Random Forest Regressor Model:
--------------------------------------------------
Root mean squared error:  685.7854755021119
Mean absolute % error:  3
R-squared:  0.9782770775740746
```

[56]:
```
print("Test Results for Random Forest Regressor Model:")
print(50 * '-')
print("Root mean squared error: ", sqrt(mse(y_test, y_test_pred)))
print("Mean absolute % error: ", round(mean_absolute_percentage_error(y_test,␣
↪y_test_pred)))
print("R-squared: ", r2_score(y_test, y_test_pred))
```

```
Test Results for Random Forest Regressor Model:
--------------------------------------------------
Root mean squared error:  1467.7479927917857
```

```
Mean absolute % error:  8
R-squared:  0.8962720058779787
```

13 XGB Regressor

[57]:
```
tuned_params = {'max_depth': [1, 2, 3, 4, 5], 'learning_rate': [0.01, 0.05, 0.
↪1], 'n_estimators': [100, 200, 300, 400, 500], 'reg_lambda': [0.001, 0.1, 1.
↪0, 10.0, 100.0]}
model = RandomizedSearchCV(XGBRegressor(), tuned_params, n_iter=20, scoring =␣
↪'neg_mean_absolute_error', cv=5, n_jobs=-1)
model.fit(X_train, y_train)
```

[57]:
```
RandomizedSearchCV(cv=5,
                   estimator=XGBRegressor(base_score=None, booster=None,
                                          colsample_bylevel=None,
                                          colsample_bynode=None,
                                          colsample_bytree=None, gamma=None,
                                          gpu_id=None, importance_type='gain',
                                          interaction_constraints=None,
                                          learning_rate=None,
                                          max_delta_step=None, max_depth=None,
                                          min_child_weight=None, missing=nan,
                                          monotone_constraints=None,
                                          n_estimators=100, n…
                                          random_state=None, reg_alpha=None,
                                          reg_lambda=None,
                                          scale_pos_weight=None, subsample=None,
                                          tree_method=None,
                                          validate_parameters=None,
                                          verbosity=None),
                   n_iter=20, n_jobs=-1,
                   param_distributions={'learning_rate': [0.01, 0.05, 0.1],
                                        'max_depth': [1, 2, 3, 4, 5],
                                        'n_estimators': [100, 200, 300, 400,
                                                         500],
                                        'reg_lambda': [0.001, 0.1, 1.0, 10.0,
                                                       100.0]},
                   scoring='neg_mean_absolute_error')
```

[58]:
```
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)
```

[59]:
```
print("Train Results for XGBoost Regressor Model:")
print(50 * '-')
print("Root mean squared error: ", sqrt(mse(y_train.values, y_train_pred)))
print("Mean absolute % error: ", round(mean_absolute_percentage_error(y_train.
↪values, y_train_pred)))
```

```
print("R-squared: ", r2_score(y_train.values, y_train_pred))
```

```
Train Results for XGBoost Regressor Model:
-------------------------------------------------
Root mean squared error:  707.1638519523252
Mean absolute % error:  6
R-squared:  0.9769016056692901
```

[60]:
```
print("Test Results for XGBoost Regressor Model:")
print(50 * '-')
print("Root mean squared error: ", sqrt(mse(y_test, y_test_pred)))
print("Mean absolute % error: ", round(mean_absolute_percentage_error(y_test,␣
 ↪y_test_pred)))
print("R-squared: ", r2_score(y_test, y_test_pred))
```

```
Test Results for XGBoost Regressor Model:
-------------------------------------------------
Root mean squared error:  1416.2603587348945
Mean absolute % error:  9
R-squared:  0.9034217819285152
```