**Activity 1: Collect the dataset**

**Activity 1.1: Importing the libraries**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

#from sklearn.ensemble import RandomForestClassifier,GradientBoosting

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')

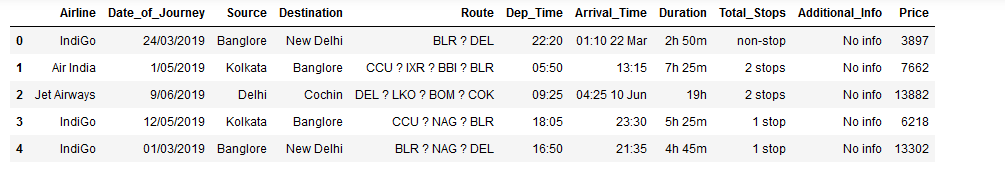
plt.style.use('fivethirtyeight')

**Activity 1.2: Reading the Dataset**

import pandas as pd

df=pd.read\_csv("E:\\NMDS\\FlightBooking.csv")

df.head()



**Activity 2: Data Preparation**

In this activity we clean the dataset as per our needs.

We no start exploring the columns available in our dataset. The first thing e do is to create a list of categorical columns, and check the unique values present in these columns

for i in category:

print(i,data[i].unique())

We now split the Date column to extract the 'Date','Month' and 'Year' values, and store them in

new columns in our data frame

df.date\_of\_Journey=df.Date\_of\_Journey.str.split('/')

df.Date\_of\_Journey

Further, we split the Route column to create multiple columns with cities that the flight travels through. We check the maximum number of stops that a flight has, to confirm what should be the maximum number of cities in the longest route

#Split the Route column

df.Total\_Stops.unique()

array(['non-stop', '2 stops', '1 stop', '3 stops', nan, '4 stops'],

dtype=object)

Since the maximum number of stops is 4, there should be maximum 6 cities in any particular route. We split the data in route column, and store all the city names in separate columns

#We split the data in route column

df.Route=df.Route.str.split('->')

df.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

df.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

#import prision

#from prision import Category

#for i in category:

#print(i,data[i].unique())

#We now split the Date column to extract the 'Date','Month' and 'Year' values, and store them in

#new columns in our data frame

df.date\_of\_Journey=df.Date\_of\_Journey.str.split('/')

df.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

#Traiting the data\_column

df['Date']=df.Date\_of\_Journey.str[0]

df['Month']=df.Date\_of\_Journey.str[1]

df['Year']=df.Date\_of\_Journey.str[2]

#Split the Route column

df.Total\_Stops.unique()

array(['non-stop', '2 stops', '1 stop', '3 stops', nan, '4 stops'],

dtype=object)

#We split the data in route column

df.Route=df.Route.astype(str).str.split('->')

df.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

df['City1']=df.Route.str[0]

df['City2']=df.Route.str[1]

df['City3']=df.Route.str[2]

df['City4']=df.Route.str[3]

df['City5']=df.Route.str[4]

df['City6']=df.Route.str[5]

#In similar manner, we split the Dep\_time column, and create separate for departue hours and minutes

df.Dep\_Time=df.Dep\_Time.astype(str).str.split(':')

df['Dep\_Time\_Hour']=df.Dep\_Time.str[0]

df['Dep\_Time\_Mins']=df.Dep\_Time.str[1]

#We also split the Arrival\_Time Column

df.Arrival\_Time=df.Arrival\_Time.astype(str).str.split('')

df['Arrival\_date']=df.Arrival\_Time.str[1]

df['Time\_of\_Arrival']=df.Arrival\_Time.str[0]

df['Time\_of\_Arrival']=df.Time\_of\_Arrival.astype(str).str.split(':')

df['Arrival\_Time\_Hour']=df.Time\_of\_Arrival.str[0]

df['Arrival\_Time\_Mins']=df.Time\_of\_Arrival.str[1]

#we also treat the 'Total\_stops', column, and replace non-stop flights with 0 values and

#extract the integer part of the "Total\_Stops"

df.Total\_Stops.replace('npn\_stop',0,inplace=True)

df.Total\_Stops=df.Total\_Stops.str.split('')

df.Total\_Stops=df.Total\_Stops.str[0]

df.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)

#df.Additional\_Info.replace('No Info','No Info',inplace=True)

df.shape

(10683, 26)

df.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 10683 entries, 0 to 10682

Data columns (total 26 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 10683 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

11 Date 10683 non-null object

12 Month 10683 non-null object

13 Year 10683 non-null object

14 City1 10683 non-null object

15 City2 0 non-null float64

16 City3 0 non-null float64

17 City4 0 non-null float64

18 City5 0 non-null float64

19 City6 0 non-null float64

20 Dep\_Time\_Hour 10683 non-null object

21 Dep\_Time\_Mins 10683 non-null object

22 Arrival\_date 10683 non-null object

23 Time\_of\_Arrival 10683 non-null object

24 Arrival\_Time\_Hour 10683 non-null object

25 Arrival\_Time\_Mins 0 non-null float64

dtypes: float64(6), int64(1), object(19)

memory usage: 2.1+ MB

#we also drop some columns like 'city6' an 'city5', since majority of the data in these columns was NaN(null)

#df.drop(['City4','City5','City6'],axis=1,inplace=True)

#df.drop(['Date\_of\_Journey','Route','Dep\_Time','Duration'],axis=1,inplace=True)

#df.drop(['Time\_of\_Arrival'],axis=1,inplace=True)

df.isnull().sum()

Airline 0

Date\_of\_Journey 0

Source 0

Destination 0

Route 0

Dep\_Time 0

Arrival\_Time 0

Duration 0

Total\_Stops 1

Additional\_Info 0

Price 0

Date 0

Month 0

Year 0

City1 0

City2 10683

City3 10683

City4 10683

City5 10683

City6 10683

Dep\_Time\_Hour 0

Dep\_Time\_Mins 0

Arrival\_date 0

Time\_of\_Arrival 0

Arrival\_Time\_Hour 0

Arrival\_Time\_Mins 10683

dtype: int64

#Activity 2.1:Replacing Missing Values

#filling City3 as None, the missing value are less

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

0 None,inplace=True

1 None,inplace=True

2 None,inplace=True

3 None,inplace=True

4 None,inplace=True

...

10678 None,inplace=True

10679 None,inplace=True

10680 None,inplace=True

10681 None,inplace=True

10682 None,inplace=True

Name: City3, Length: 10683, dtype: object

#filling Arrival\_Date as Departure\_Date

df['Arrival\_date'].fillna((df['Date']),inplace=True)

#filling Travel\_Mins as Zero(0)

#df['Travel\_Mins'].fillna(0,inplace=True)

df['Arrival\_Time\_Mins'].fillna(0,inplace=True)

df.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 10683 entries, 0 to 10682

Data columns (total 26 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 10683 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

11 Date 10683 non-null object

12 Month 10683 non-null object

13 Year 10683 non-null object

14 City1 10683 non-null object

15 City2 0 non-null float64

16 City3 0 non-null float64

17 City4 0 non-null float64

18 City5 0 non-null float64

19 City6 0 non-null float64

20 Dep\_Time\_Hour 10683 non-null object

21 Dep\_Time\_Mins 10683 non-null object

22 Arrival\_date 10683 non-null object

23 Time\_of\_Arrival 10683 non-null object

24 Arrival\_Time\_Hour 10683 non-null object

25 Arrival\_Time\_Mins 10683 non-null float64

dtypes: float64(6), int64(1), object(19)

memory usage: 2.1+ MB

df.skew()

Price 1.812552

Date 1.298651

City2 NaN

City3 NaN

City4 NaN

City5 NaN

City6 NaN

Dep\_Time\_Hour 0.113073

Dep\_Time\_Mins 0.167029

Arrival\_date 0.151783

Arrival\_Time\_Mins 0.000000

dtype: float64

#changing the numerical columns from object to int

#df.Total\_Stops=df.Total\_Stops.astype('int64')

df.Date=df.Date.astype('int64')

df.Month=df.Month.astype(str)

df.Year=df.Year.astype(str)

df.Dep\_Time\_Hour=df.Dep\_Time\_Hour.astype('int64')

df.Dep\_Time\_Hour=df.Dep\_Time\_Hour.astype('int64')

df.Dep\_Time\_Mins=df.Dep\_Time\_Mins.astype('int64')

[df['Arrival\_Time\_Hour']=='5m']

[0 False

1 False

2 False

3 False

4 False

...

10678 False

10679 False

10680 False

10681 False

10682 False

Name: Arrival\_Time\_Hour, Length: 10683, dtype: bool]

df.drop(index=6474,inplace=True,axis=1)

#df.Travel\_Hours=df.Travel\_Hours.astype('int64')

#df.Arrival\_Time\_Hour=df.Arrival\_Time\_Hour.astype('int64')

#Creating a list of Different types of Columns

Categorical=['Airline','Source','Destination','Additional\_Info','City1']

Numerical=['Total\_Stops','Date','Month','Year','Dep\_Time\_Hour','Dep\_Time\_Mins','Arrival\_date',

'Arrival\_Time\_Hour','Arrival\_Time\_Mins','Travel\_Time\_Mins','Travel\_Hours','Travel\_Mins']

#Activity 2.2: Label Encoding

from sklearn.preprocessing import LabelEncoder

le=LabelEncoder()

df.Airline=le.fit\_transform(df.Airline)

df.Source=le.fit\_transform(df.Source)

df.Destination=le.fit\_transform(df.Destination)

df.Total\_Stops=le.fit\_transform(df.Total\_Stops)

df.City1=le.fit\_transform(df.City1)

df.City2=le.fit\_transform(df.City2)

df.City3=le.fit\_transform(df.City3)

df.Additional\_info=le.fit\_transform(df.Additional\_Info)

df.head()

|  | **Airline** | **Date\_of\_Journey** | **Source** | **Destination** | **Route** | **Dep\_Time** | **Arrival\_Time** | **Duration** | **Total\_Stops** | **Additional\_Info** | **...** | **City3** | **City4** | **City5** | **City6** | **Dep\_Time\_Hour** | **Dep\_Time\_Mins** | **Arrival\_date** | **Time\_of\_Arrival** | **Arrival\_Time\_Hour** | **Arrival\_Time\_Mins** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 3 | 24/03/2019 | 0 | 5 | [BLR ? DEL] | [22, 20] | [, 0, 1, :, 1, 0, , 2, 2, , M, a, r, ] | 2h 50m | 0 | No info | ... | 0 | NaN | NaN | NaN | 22 | 20 | 0 | [] |  | 0.0 |
| **1** | 1 | 1/05/2019 | 3 | 0 | [CCU ? IXR ? BBI ? BLR] | [05, 50] | [, 1, 3, :, 1, 5, ] | 7h 25m | 0 | No info | ... | 0 | NaN | NaN | NaN | 5 | 50 | 1 | [] |  | 0.0 |
| **2** | 4 | 9/06/2019 | 2 | 1 | [DEL ? LKO ? BOM ? COK] | [09, 25] | [, 0, 4, :, 2, 5, , 1, 0, , J, u, n, ] | 19h | 0 | No info | ... | 0 | NaN | NaN | NaN | 9 | 25 | 0 | [] |  | 0.0 |
| **3** | 3 | 12/05/2019 | 3 | 0 | [CCU ? NAG ? BLR] | [18, 05] | [, 2, 3, :, 3, 0, ] | 5h 25m | 0 | No info | ... | 0 | NaN | NaN | NaN | 18 | 5 | 2 | [] |  | 0.0 |
| **4** | 3 | 01/03/2019 | 0 | 5 | [BLR ? NAG ? DEL] | [16, 50] | [, 2, 1, :, 3, 5, ] | 4h 45m | 0 | No info | ... | 0 | NaN | NaN | NaN | 16 | 50 | 2 | [] |  | 0.0 |

5 rows × 26 columns

#Activity 2.3: Output Columns

df=df[['Airline','Source','Destination','Date','Month','Year','Dep\_Time\_Hour','Dep\_Time\_Mins','Arrival\_date',

'Arrival\_Time\_Hour','Arrival\_Time\_Mins','Price']]

df.head()

|  | **Airline** | **Source** | **Destination** | **Date** | **Month** | **Year** | **Dep\_Time\_Hour** | **Dep\_Time\_Mins** | **Arrival\_date** | **Arrival\_Time\_Hour** | **Arrival\_Time\_Mins** | **Price** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 3 | 0 | 5 | 2 | 4 | / | 22 | 20 | 0 |  | 0.0 | 3897 |
| **1** | 1 | 3 | 0 | 1 | / | 0 | 5 | 50 | 1 |  | 0.0 | 7662 |
| **2** | 4 | 2 | 1 | 9 | / | 0 | 9 | 25 | 0 |  | 0.0 | 13882 |
| **3** | 3 | 3 | 0 | 1 | 2 | / | 18 | 5 | 2 |  | 0.0 | 6218 |
| **4** | 3 | 0 | 5 | 0 | 1 | / | 16 | 50 | 2 |  | 0.0 | 13302 |

#Activity 3:Exploratory Data Analyis

#Activity 3.1: Descriptive statistical

df.describe()

|  | **Airline** | **Source** | **Destination** | **Date** | **Dep\_Time\_Hour** | **Dep\_Time\_Mins** | **Arrival\_Time\_Mins** | **Price** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **count** | 10682.000000 | 10682.000000 | 10682.000000 | 10682.000000 | 10682.000000 | 10682.000000 | 10682.0 | 10682.000000 |
| **mean** | 3.966205 | 1.952069 | 1.435967 | 2.941678 | 12.490358 | 24.408819 | 0.0 | 9086.292735 |
| **std** | 2.352090 | 1.177110 | 1.474773 | 2.732095 | 5.748819 | 18.767225 | 0.0 | 4610.885695 |
| **min** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.0 | 1759.000000 |
| **25%** | 3.000000 | 2.000000 | 0.000000 | 1.000000 | 8.000000 | 5.000000 | 0.0 | 5277.000000 |
| **50%** | 4.000000 | 2.000000 | 1.000000 | 2.000000 | 11.000000 | 25.000000 | 0.0 | 8372.000000 |
| **75%** | 4.000000 | 3.000000 | 2.000000 | 3.000000 | 18.000000 | 40.000000 | 0.0 | 12373.000000 |
| **max** | 11.000000 | 4.000000 | 5.000000 | 9.000000 | 23.000000 | 55.000000 | 0.0 | 79512.000000 |

#Ploting Countplots for Categorical Data

import matplotlib.pyplot as plt

import seaborn as sns

c=1

plt.figure(figsize=(20,45))

for i in Categorical:

plt.subplot(6,3,c)

sns.countplot(df[i])

plt.xticks(rotation=90)

plt.tight\_layout(pad=3.0)

c=c+1

plt.show()

---------------------------------------------------------------------------

KeyError Traceback (most recent call last)

File ~\anaconda3\lib\site-packages\pandas\core\indexes\base.py:3802, in Index.get\_loc(self, key, method, tolerance)

3801 try:

-> 3802 return self.\_engine.get\_loc(casted\_key)

3803 except KeyError as err:

File ~\anaconda3\lib\site-packages\pandas\\_libs\index.pyx:138, in pandas.\_libs.index.IndexEngine.get\_loc()

File ~\anaconda3\lib\site-packages\pandas\\_libs\index.pyx:165, in pandas.\_libs.index.IndexEngine.get\_loc()

File pandas\\_libs\hashtable\_class\_helper.pxi:5745, in pandas.\_libs.hashtable.PyObjectHashTable.get\_item()

File pandas\\_libs\hashtable\_class\_helper.pxi:5753, 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)

Cell In[112], line 8

6 for i in Categorical:

7 plt.subplot(6,3,c)

----> 8 sns.countplot(df[i])

9 plt.xticks(rotation=90)

10 plt.tight\_layout(pad=3.0)

File ~\anaconda3\lib\site-packages\pandas\core\frame.py:3807, in DataFrame.\_\_getitem\_\_(self, key)

3805 if self.columns.nlevels > 1:

3806 return self.\_getitem\_multilevel(key)

-> 3807 indexer = self.columns.get\_loc(key)

3808 if is\_integer(indexer):

3809 indexer = [indexer]

File ~\anaconda3\lib\site-packages\pandas\core\indexes\base.py:3804, in Index.get\_loc(self, key, method, tolerance)

3802 return self.\_engine.get\_loc(casted\_key)

3803 except KeyError as err:

-> 3804 raise KeyError(key) from err

3805 except TypeError:

3806 # If we have a listlike key, \_check\_indexing\_error will raise

3807 # InvalidIndexError. Otherwise we fall through and re-raise

3808 # the TypeError.

3809 self.\_check\_indexing\_error(key)

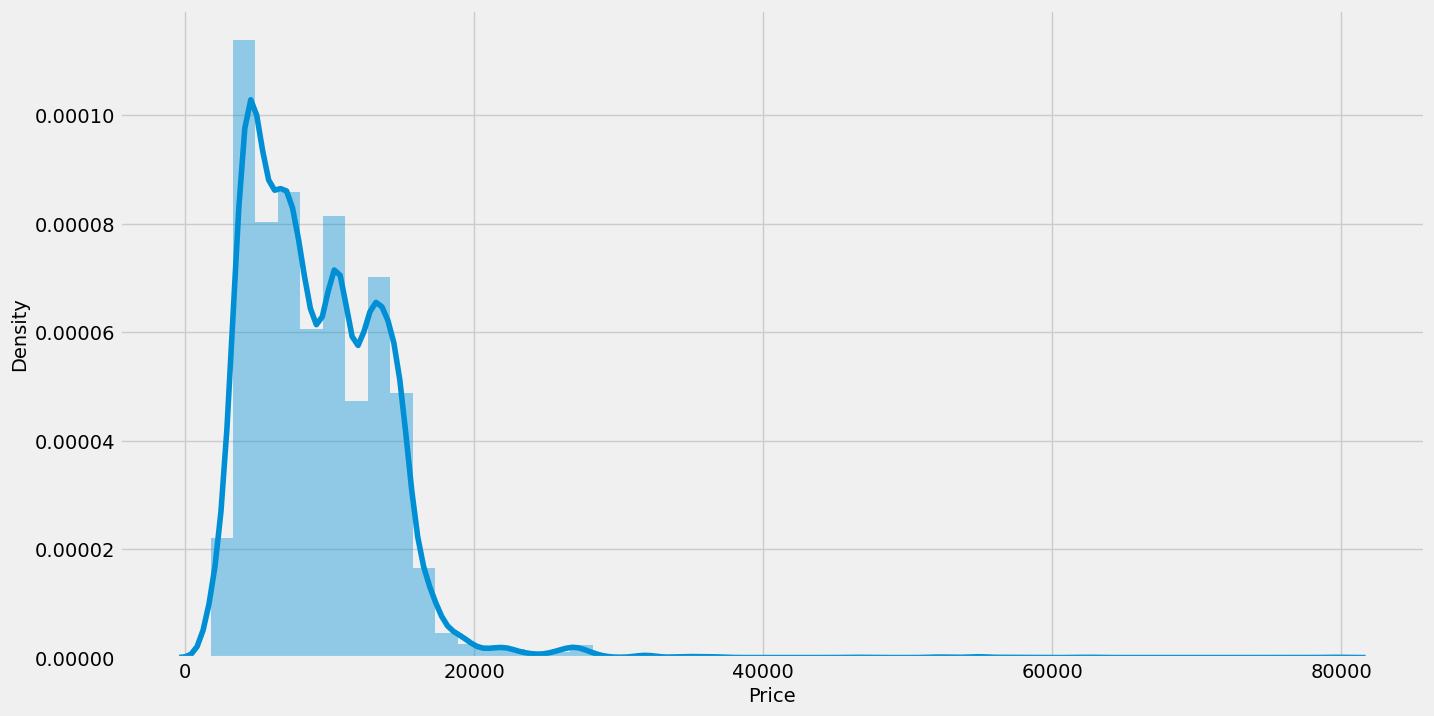
KeyError: 'Additional\_Info'

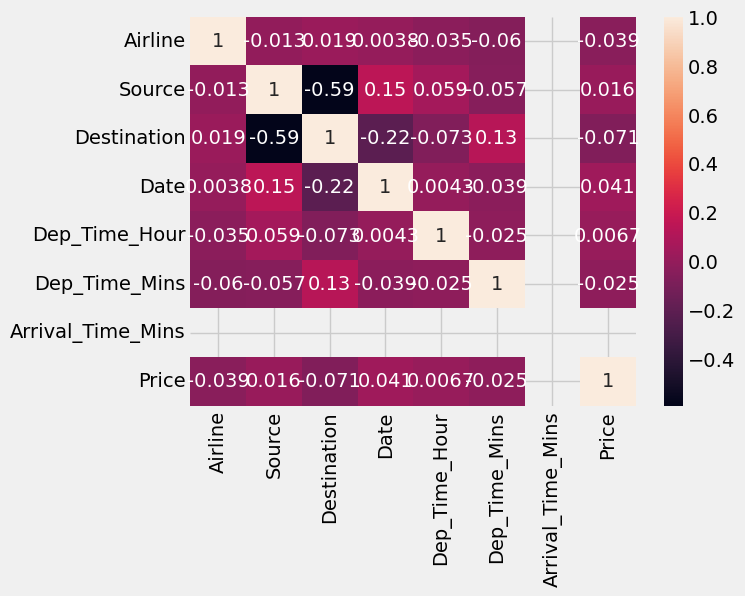
#Distribution of price column

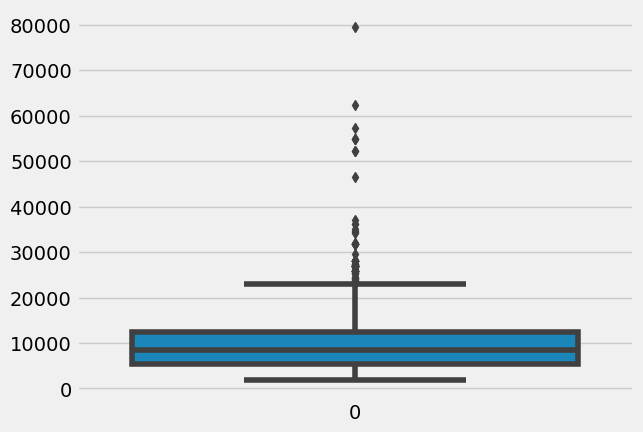
plt.figure(figsize=(15,8))

sns.distplot(df.Price)

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







#Scaling the data

df=df[['Airline','Source','Destination','Date','Month','Year','Dep\_Time\_Hour','Dep\_Time\_Mins','Arrival\_date',

'Arrival\_Time\_Hour','Arrival\_Time\_Mins','Price']]

y=df['Price']

x=df.drop(columns=['Price'],axis=1)

import pandas as pd

from sklearn.preprocessing import StandardScaler

ss=StandardScaler()

#x\_scaled=ss.fit\_transform(x)

#x\_scaled=pd.DataFrame(x\_scaled,columns=x.columns)

x.head()

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.tail()

#Model Building

#model 1:RandomForestClassifier,GradientBoostingRegressor,AdaBoostRegressor

from sklearn.ensemble import RandomForestClassifier,GradientBoostingRegressor,AdaBoostRegressor

rfr=RandomForestClassifier()

gb=GradientBoostingRegressor()

ad=AdaBoostRegressor()

from sklearn.metrics import f1\_score

from sklearn.metrics import classification\_report,confusion\_matrix

from sklearn.metrics import r2\_score,mean\_absoulte\_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 Absoult 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)))

#model 2:RandomForestClassifier,GradientBoostingRegressor,AdaBoostRegressor

from sklearn.neighbors import KNeighborsRegressor

from sklearn.svm import SVR

from.sklearn.tree import DecisionTreeRegressor

from sklearn.metrics import r2\_score,mean\_absoulte\_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 for train data",r2\_score(y\_train,i.predict(x\_train)))

print("Mean Absoult 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)))

#model 3: Checking Cross Validation for RandomForestRegressor

from sklearn.model\_selection import cross\_val\_score

from sklearn.ensemble import RandomForestClassifier,GradientBoostingRegressor,AdaBoostRegressor

rfr=RandomForestClassifier()

gb=GradientBoostingRegressor()

ad=AdaBoostRegressor()

for i in [rfr,gb,ad]:

i.fit(x\_train,y\_train)

y\_pred=i.predict(x\_test)

for i in range(2,5):

CV=cross\_val\_score(rfr,x,y,CV=i)

print(rfr,CV.mean())

<https://python-course.eu/machine-learning/boosting-algorithm-in-python.php>