



# Flight Booking Price Prediction



# Agenda

**01** Importing the Libraries

**03** Data Visualization

**05** Feature Selection

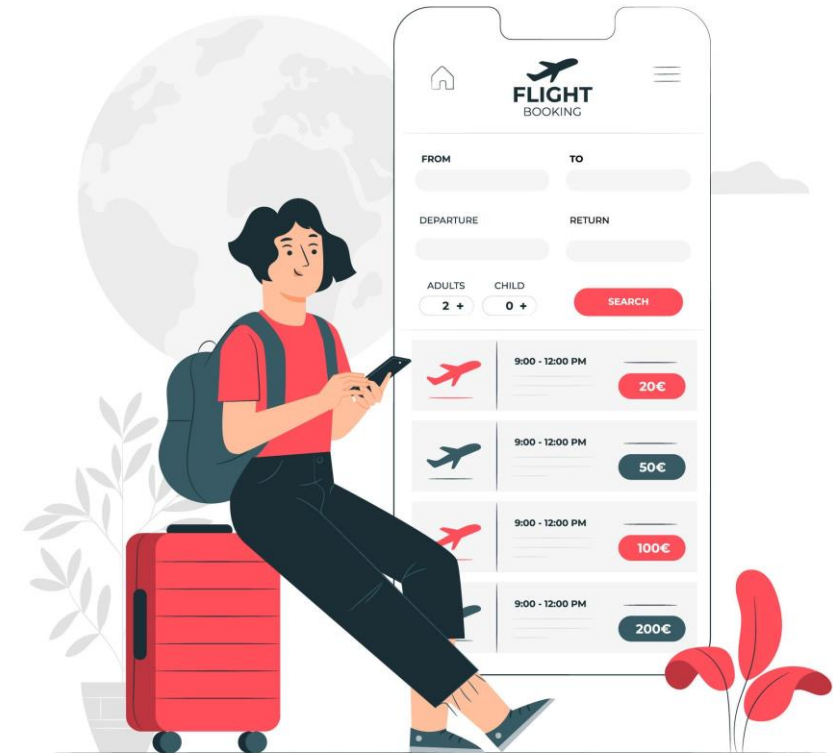
**02** Loading the Data

**04** One Hot Encoding

**06** Implementing ML Algorithms

# Problem Statement

The objective is to analyze the flight booking dataset obtained from a platform which is used to book flight tickets. A thorough study of the data will aid in the discovery of valuable insights that will be of enormous value to passengers. Apply EDA, statistical methods and Machine learning algorithms in order to get meaningful information from it.



Flight booking price prediction dataset contains around 3 lacs records with 11 attributes .



# Dataset Information

| Attributes       | Description   |
|------------------|---|
| Airline          | Name of the airline company                                     |
| Flight           | Plane's flight code   |
| Source City      | City from which the flight takes off                            |
| Departure Time   | Time of Departure   |
| Stops            | Number of stops between the source and destination cities       |
| Arrival Time     | Time of Arrival   |
| Destination City | City where the flight will land                                 |
| Class            | Contains information on seat class                              |
| Duration         | Overall amount of time taken to travel between cities in hours. |
| Days left        | Subtracting the trip date by the booking date.                  |
| Price            | Ticket price  |

# Importing the Libraries

---

We start off this project by importing all the necessary libraries that will be required for the process.

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
```

## Loading the data and removing unnecessary column from the dataframe

```
import pandas as pd
df=pd.read_csv("Flight_Booking.csv")
df=df.drop(columns=["Unnamed: 0"])
df.head()
```

|   | airline  | flight  | source_city | departure_time | stops | arrival_time  | destination_city | class   | duration | days_left | price |
|---|----------|---------|-------------|----------------|-------|---------------|------------------|---------|----------|-----------|-------|
| 0 | SpiceJet | SG-8709 | Delhi       | Evening        | zero  | Night         | Mumbai           | Economy | 2.17     | 1         | 5953  |
| 1 | SpiceJet | SG-8157 | Delhi       | Early_Morning  | zero  | Morning       | Mumbai           | Economy | 2.33     | 1         | 5953  |
| 2 | AirAsia  | I5-764  | Delhi       | Early_Morning  | zero  | Early_Morning | Mumbai           | Economy | 2.17     | 1         | 5956  |
| 3 | Vistara  | UK-995  | Delhi       | Morning        | zero  | Afternoon     | Mumbai           | Economy | 2.25     | 1         | 5955  |
| 4 | Vistara  | UK-963  | Delhi       | Morning        | zero  | Morning       | Mumbai           | Economy | 2.33     | 1         | 5955  |

# Loading the Data

```
df.shape  
df.info()  
df.describe()
```

```
(300153, 11)
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 300153 entries, 0 to 300152  
Data columns (total 11 columns):  
#   Column                Non-Null Count  Dtype    
---  -  
0   airline               300153 non-null object   
1   flight                300153 non-null object   
2   source_city           300153 non-null object   
3   departure_time        300153 non-null object   
4   stops                300153 non-null object   
5   arrival_time          300153 non-null object   
6   destination_city      300153 non-null object   
7   class                 300153 non-null object   
8   duration              300153 non-null float64   
9   days_left             300153 non-null int64    
10  price                 300153 non-null int64    
dtypes: float64(1), int64(2), object(8)  
memory usage: 25.2+ MB
```

Checking the shape of a dataframe and datatypes of all columns along with calculating the statistical data.

|       | duration      | days_left     | price         |
|-------|---------------|---------------|---------------|
| count | 300153.000000 | 300153.000000 | 300153.000000 |
| mean  | 12.221021     | 26.004751     | 20889.660523  |
| std   | 7.191997      | 13.561004     | 22697.767366  |
| min   | 0.830000      | 1.000000      | 1105.000000   |
| 25%   | 6.830000      | 15.000000     | 4783.000000   |
| 50%   | 11.250000     | 26.000000     | 7425.000000   |
| 75%   | 16.170000     | 38.000000     | 42521.000000  |
| max   | 49.830000     | 49.000000     | 123071.000000 |



Checking out the missing values in a dataframe

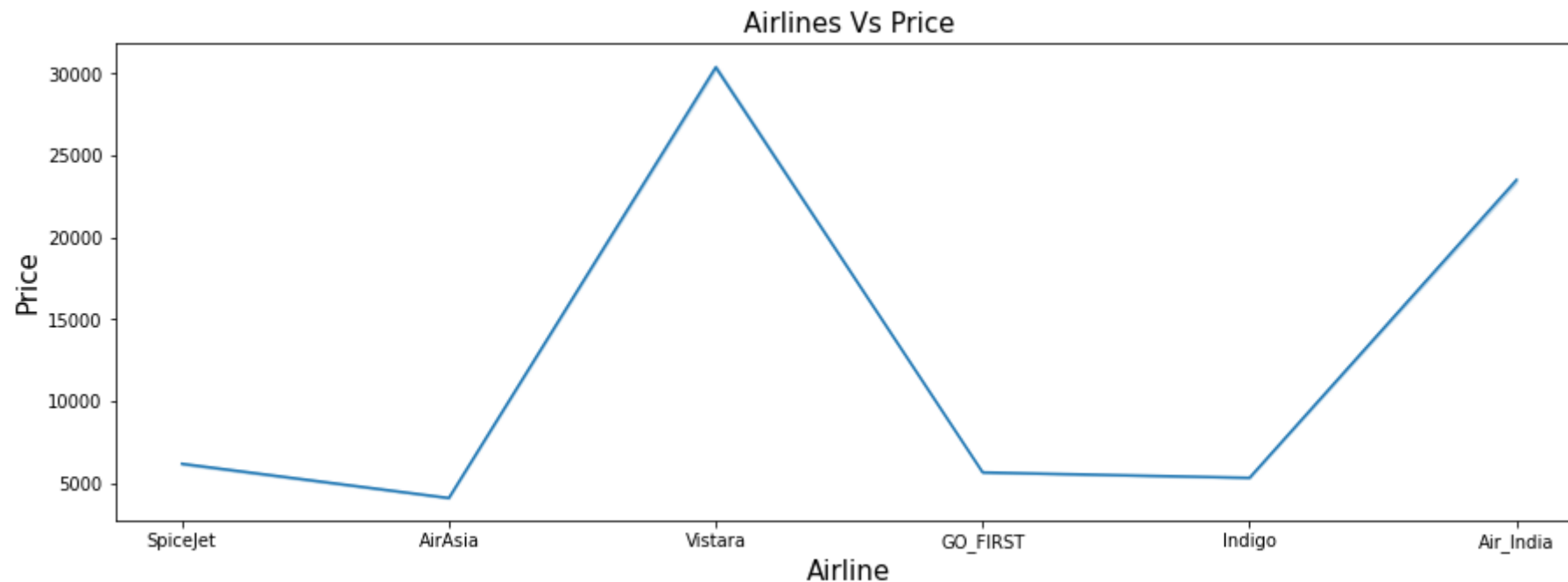
```
df.isnull().sum()
```

```
airline      0
flight       0
source_city  0
departure_time  0
stops        0
arrival_time  0
destination_city  0
class        0
duration     0
days_left   0
price        0
dtype: int64
```

# Data Visualization

```
plt.figure(figsize=(15,5))
sns.lineplot(x=df['airline'],y=df['price'])
plt.title('Airlines Vs Price',fontsize=15)
plt.xlabel('Airline',fontsize=15)
plt.ylabel('Price',fontsize=15)
plt.show()
```

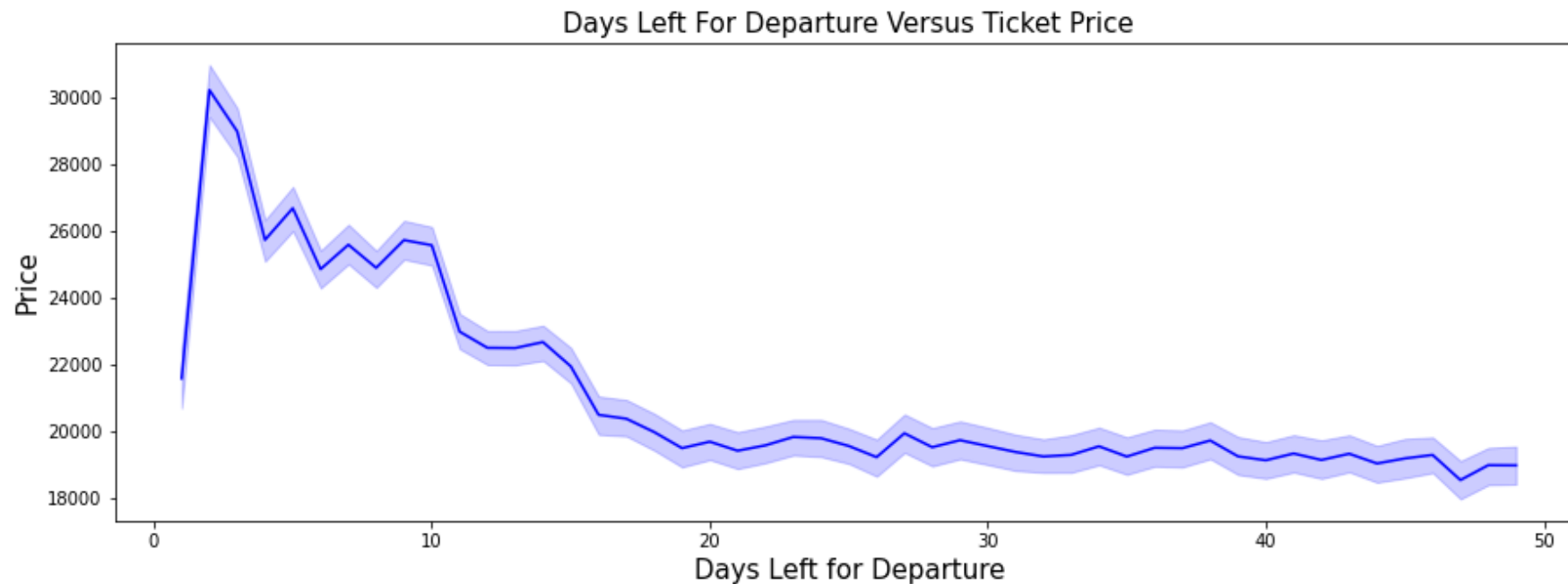
There is a variation in price with different airlines



# Data Visualization

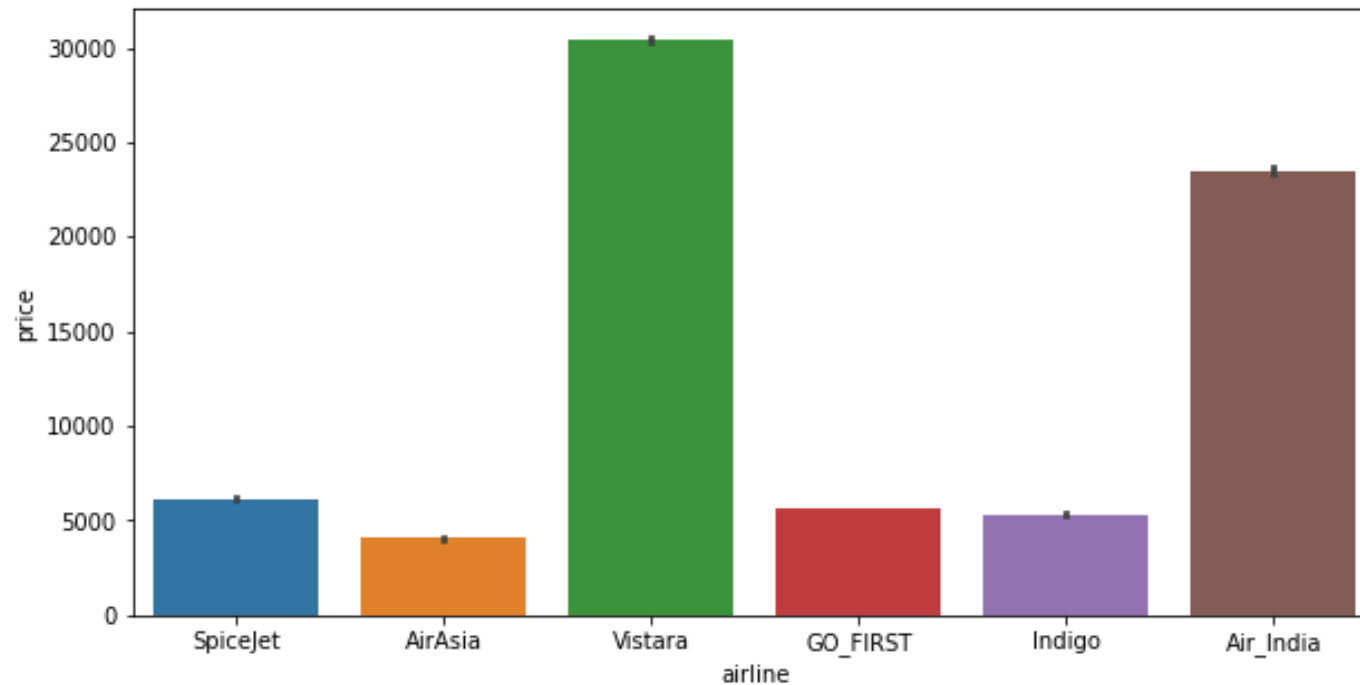
```
plt.figure(figsize=(15,5))
sns.lineplot(data=df,x='days_left',y='price',color='blue')
plt.title('Days Left For Departure Versus Ticket Price',fontsize=15)
plt.xlabel('Days Left for Departure',fontsize=15)
plt.ylabel('Price',fontsize=15)
plt.show()
```

The price of the ticket increases as the days left for departure decreases



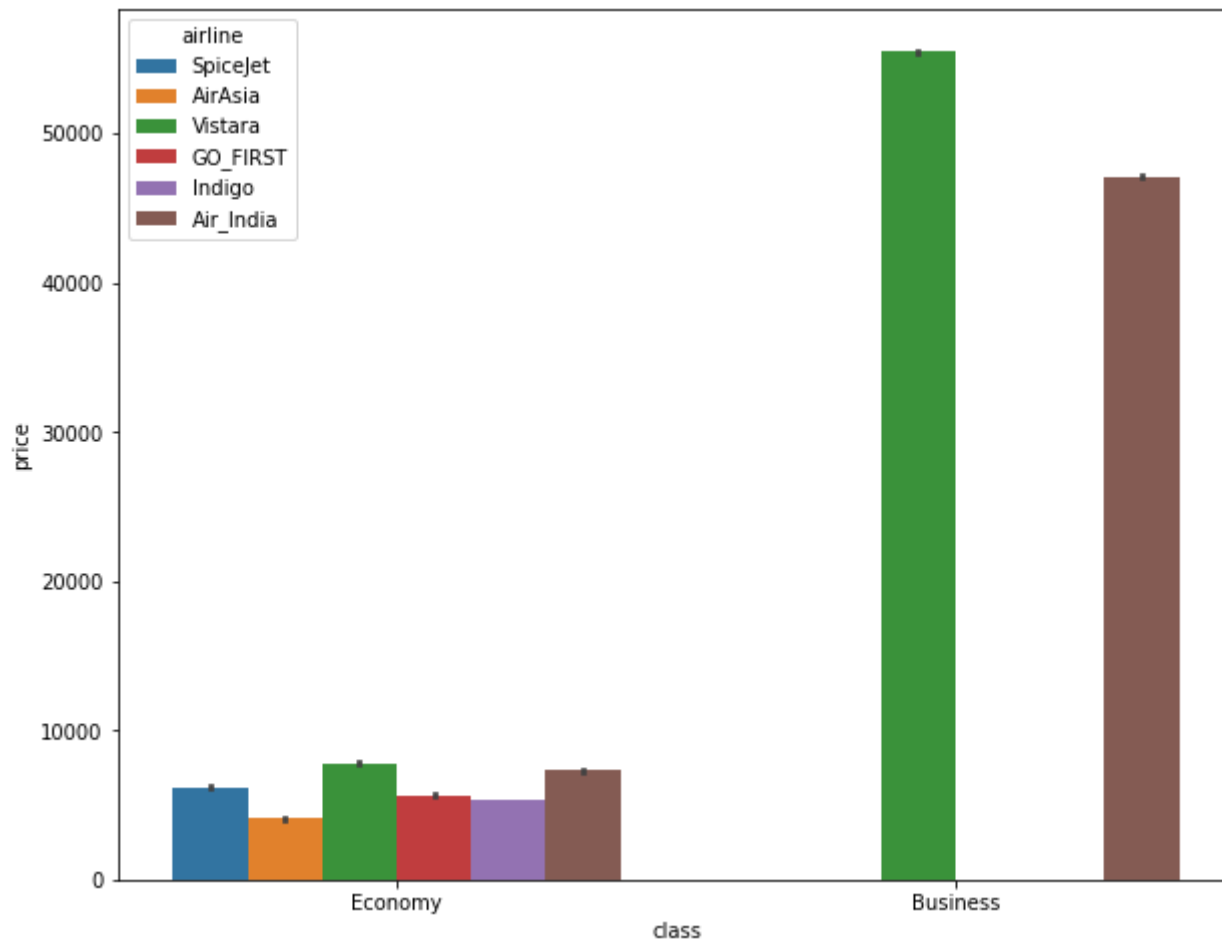
```
plt.figure(figsize=(10,5));  
sns.barplot(x='airline',y='price',data=df)
```

Price range of all the flights



# Data Visualization

```
plt.figure(figsize=(10,8));  
sns.barplot(x='class',y='price',data=df,hue='airline')
```

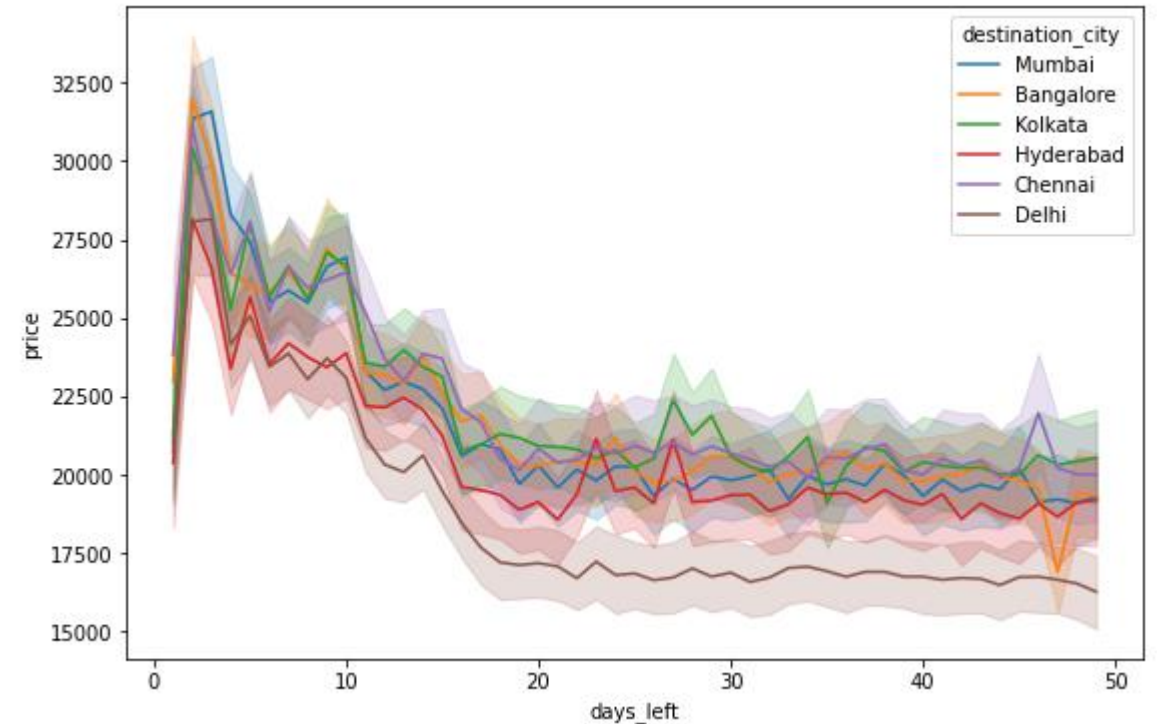
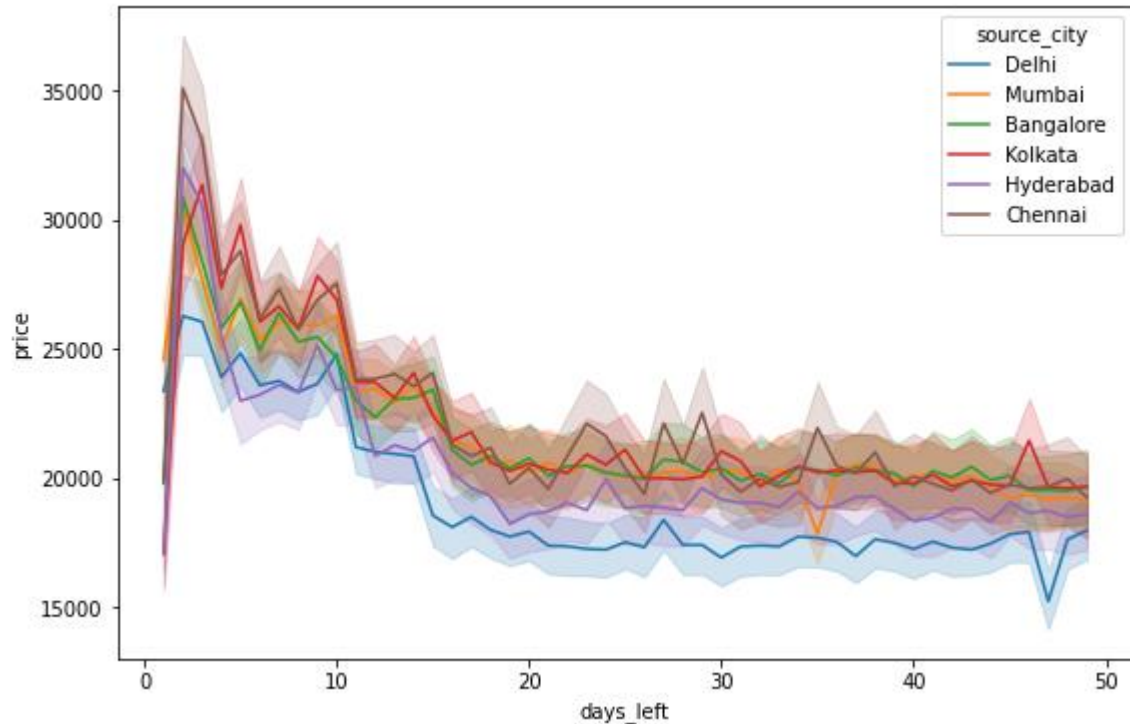


Range of price of all the flights of Economy and Business class

# Data Visualization

```
fig,ax=plt.subplots(1,2,figsize=(20,6))
sns.lineplot(x='days_left',y='price',data=df,hue='source_city',ax=ax[0])
sns.lineplot(x='days_left',y='price',data=df,hue='destination_city',ax=ax[1])
plt.show()
```

Range of price of flights with source and destination city according to the days left



# Data Visualization

## Visualization of categorical features with countplot

```
plt.figure(figsize=(15,23))

plt.subplot(4, 2, 1)
sns.countplot(x=df["airline"], data=df)
plt.title("Frequency of Airline")

plt.subplot(4, 2, 2)
sns.countplot(x=df["source_city"], data=df)
plt.title("Frequency of Source City")

plt.subplot(4, 2, 3)
sns.countplot(x=df["departure_time"], data=df)
plt.title("Frequency of Departure Time")

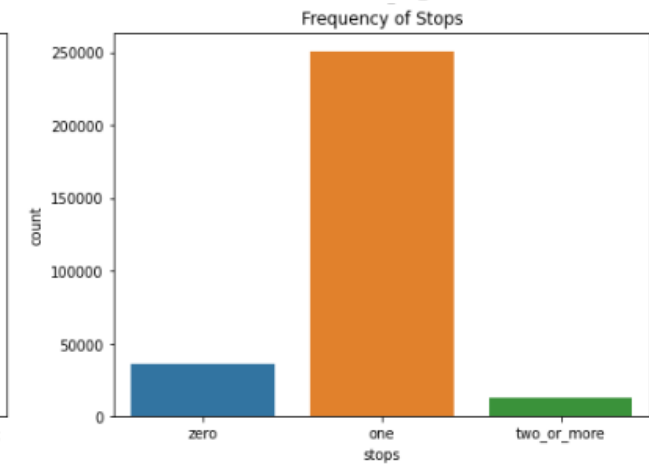
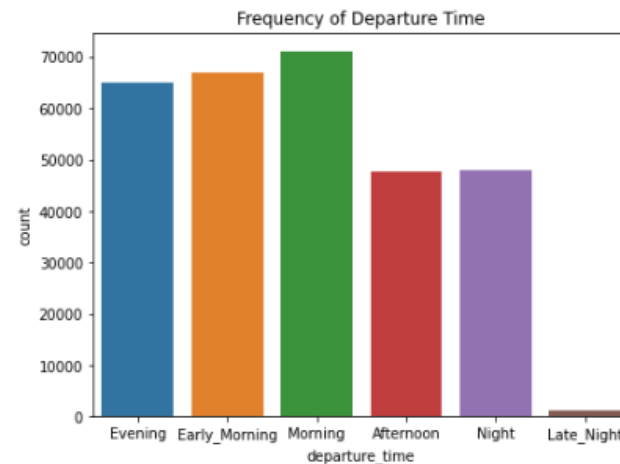
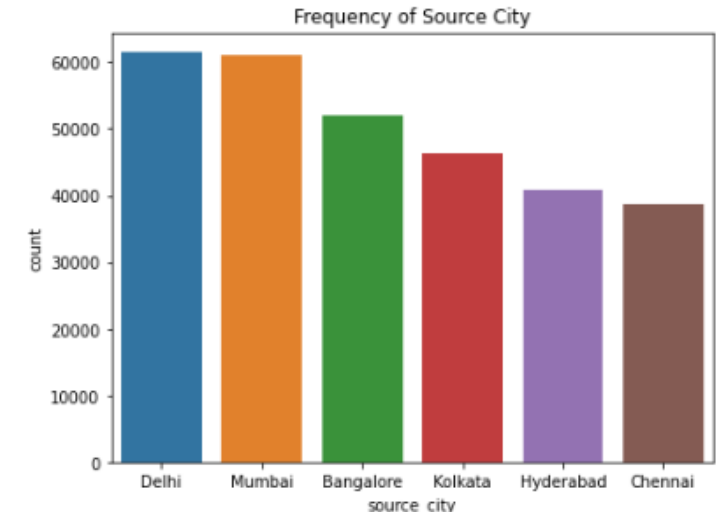
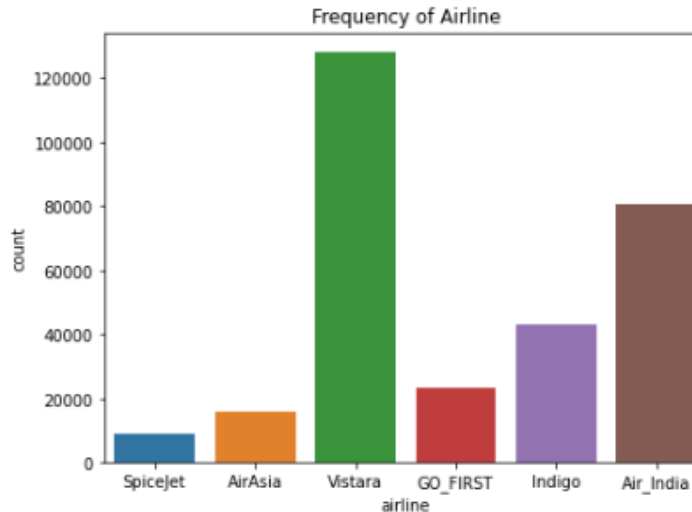
plt.subplot(4, 2, 4)
sns.countplot(x=df["stops"], data=df)
plt.title("Frequency of Stops")

plt.subplot(4, 2, 5)
sns.countplot(x=df["arrival_time"], data=df)
plt.title("Frequency of Arrival Time")

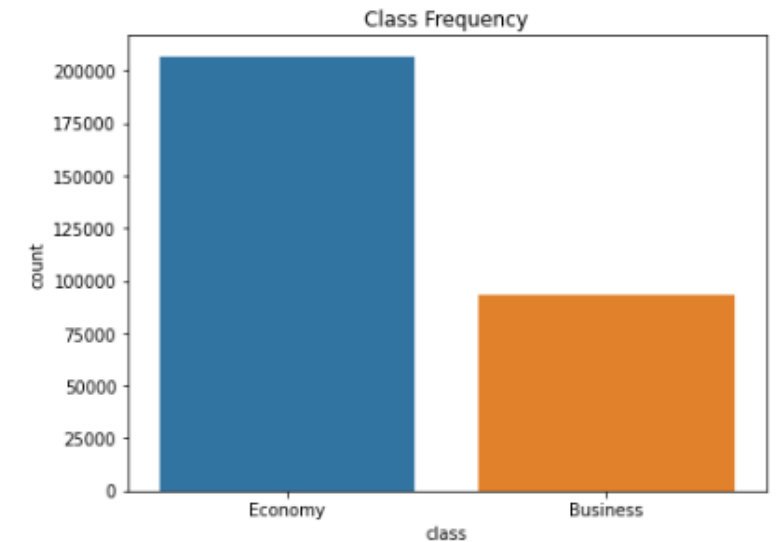
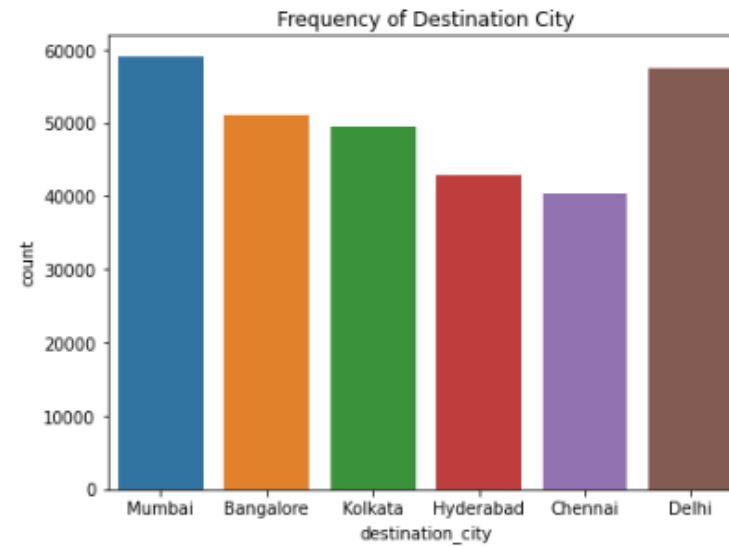
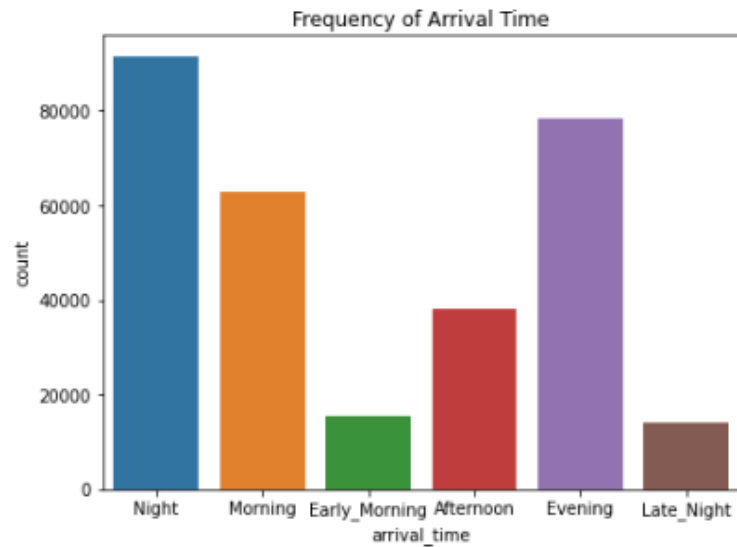
plt.subplot(4, 2, 6)
sns.countplot(x=df["destination_city"], data=df)
plt.title("Frequency of Destination City")

plt.subplot(4, 2, 7)
sns.countplot(x=df["class"], data=df)
plt.title("Class Frequency")

plt.show()
```



## Visualization of categorical features with countplot





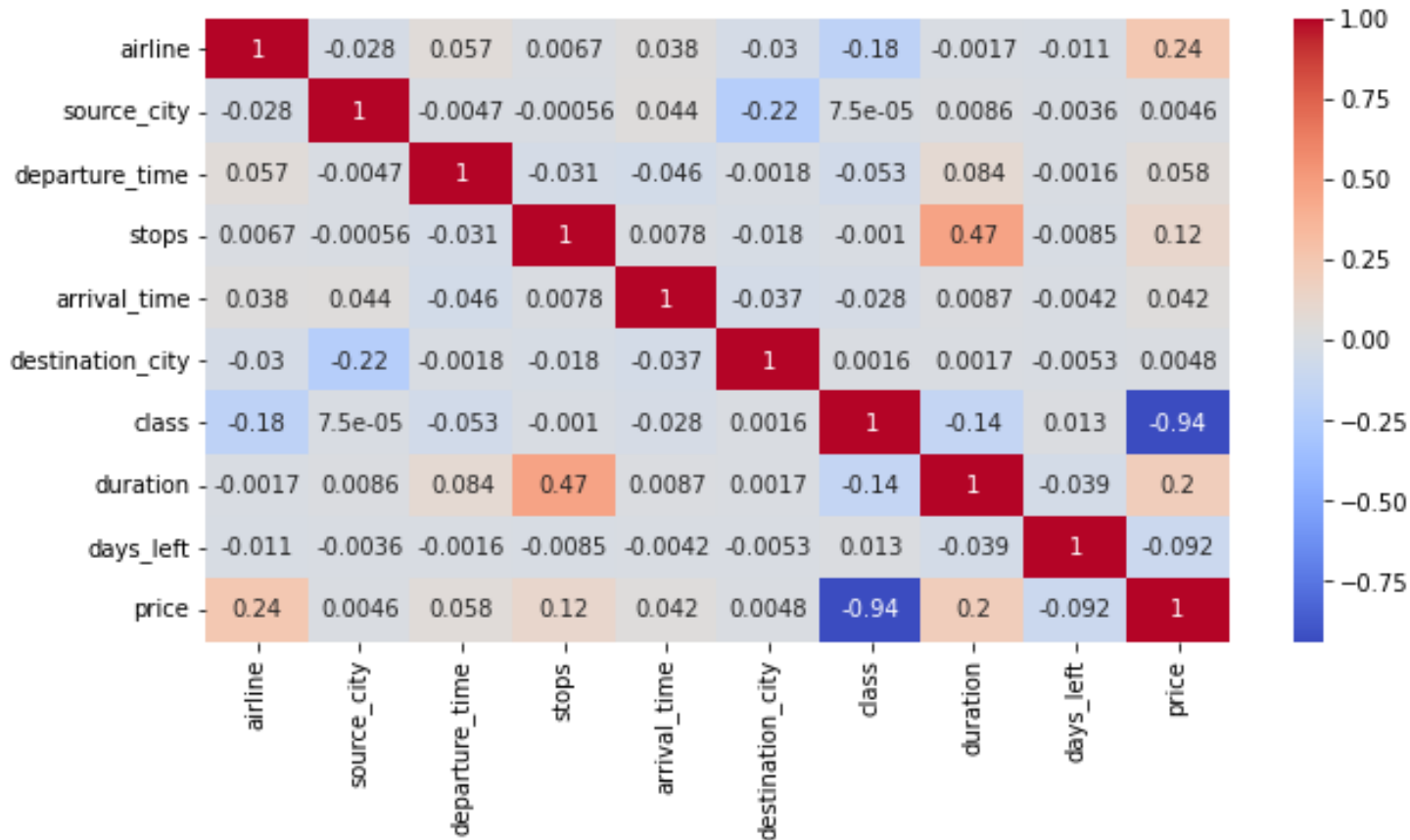
## Performing One Hot Encoding for categorical features of a dataframe

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
df["airline"]=le.fit_transform(df["airline"])
df["source_city"]=le.fit_transform(df["source_city"])
df["departure_time"]=le.fit_transform(df["departure_time"])
df["stops"]=le.fit_transform(df["stops"])
df["arrival_time"]=le.fit_transform(df["arrival_time"])
df["destination_city"]=le.fit_transform(df["destination_city"])
df["class"]=le.fit_transform(df["class"])
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 300153 entries, 0 to 300152
Data columns (total 11 columns):
#   Column                Non-Null Count  Dtype
---  -
0   airline                300153 non-null  int64
1   flight                 300153 non-null  object
2   source_city            300153 non-null  int64
3   departure_time         300153 non-null  int64
4   stops                  300153 non-null  int64
5   arrival_time           300153 non-null  int64
6   destination_city       300153 non-null  int64
7   class                  300153 non-null  int64
8   duration                300153 non-null  float64
9   days_left              300153 non-null  int64
10  price                  300153 non-null  int64
dtypes: float64(1), int64(9), object(1)
memory usage: 25.2+ MB
```

# Feature Selection

```
plt.figure(figsize=(10,5))
sns.heatmap(df.corr(),annot=True,cmap="coolwarm")
plt.show()
```



Plotting the correlation graph to see the correlation between features and dependent variable.

# Feature Selection

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
col_list = []
for col in df.columns:
    if ((df[col].dtype != 'object') & (col != 'price')):
        col_list.append(col)

X = df[col_list]
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns
vif_data["VIF"] = [variance_inflation_factor(X.values, i)
                   for i in range(len(X.columns))]
print(vif_data)
```

| feature          | VIF      |
|------------------|----------|
| airline          | 3.461766 |
| source_city      | 2.933064 |
| departure_time   | 2.746367 |
| stops            | 7.464236 |
| arrival_time     | 3.684695 |
| destination_city | 2.893218 |
| class            | 2.917521 |
| duration         | 5.037943 |
| days_left        | 4.035735 |

Selecting the features using VIF. VIF should be less than 5.  
So drop the stops feature.

# Feature Selection

```
df=df.drop(columns=["stops"])

from statsmodels.stats.outliers_influence import variance_inflation_factor
col_list = []
for col in df.columns:
    if ((df[col].dtype != 'object') & (col != 'price')):
        col_list.append(col)

X = df[col_list]
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns
vif_data["VIF"] = [variance_inflation_factor(X.values, i)
                   for i in range(len(X.columns))]
print(vif_data)
```

| feature          | VIF      |
|------------------|----------|
| airline          | 3.370020 |
| source_city      | 2.895803 |
| departure_time   | 2.746255 |
| arrival_time     | 3.632792 |
| destination_city | 2.857808 |
| class            | 2.776721 |
| duration         | 3.429344 |
| days_left        | 3.950132 |

Dropping the stops column.  
All features are having VIF  
less than 5.

# Linear Regression

Applying standardization and implementing Linear Regression Model to predict the price of a flight.

```
X = df.drop(columns=["price"])
y = df['price']
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)
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x_train=sc.fit_transform(x_train)
x_test=sc.transform(x_test)
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
y_pred=lr.predict(x_test)
difference=pd.DataFrame(np.c_[y_test,y_pred],columns=["Actual_Value","Predicted_Value"])
difference
```

|       | Actual_Value | Predicted_Value |
|-------|--------------|-----------------|
| 0     | 7366.0       | 4673.755319     |
| 1     | 64831.0      | 51713.744720    |
| 2     | 6195.0       | 6610.897658     |
| 3     | 60160.0      | 55489.844234    |
| 4     | 6578.0       | 5120.342596     |
| ...   | ...          | ...             |
| 60026 | 5026.0       | 4960.777767     |
| 60027 | 3001.0       | 4693.865426     |
| 60028 | 6734.0       | 4974.962678     |
| 60029 | 5082.0       | 2729.650066     |
| 60030 | 66465.0      | 59638.748598    |

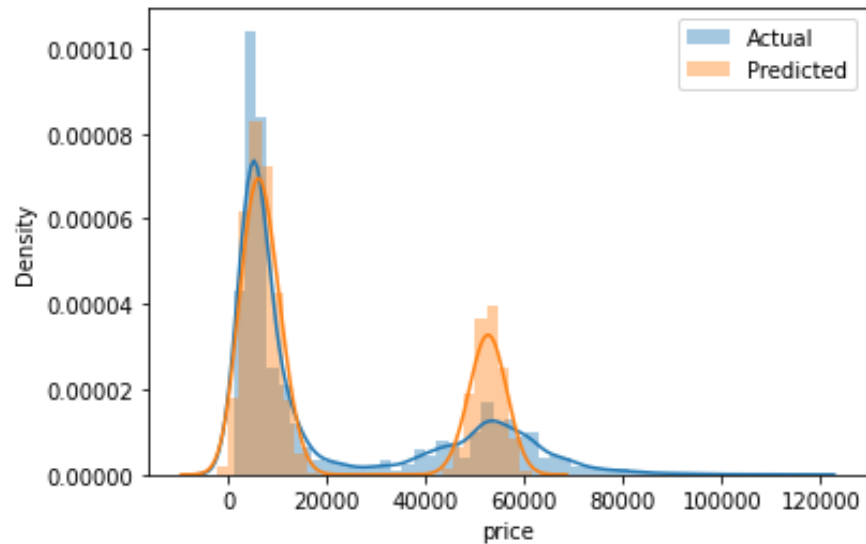
Calculating r2 score, MAE, MAPE, MSE, RMSE. Root Mean square error (RMSE) of the Linear regression model is 7259.93 and Mean absolute percentage error (MAPE) is 34 percent. Lower the RMSE and MAPE better the model.

```
from sklearn.metrics import r2_score
r2_score(y_test, y_pred)
from sklearn import metrics
mean_abs_error = metrics.mean_absolute_error(y_test, y_pred)
mean_abs_error
from sklearn.metrics import mean_absolute_percentage_error
mean_absolute_percentage_error(y_test, y_pred)
mean_sq_error = metrics.mean_squared_error(y_test, y_pred)
mean_sq_error
root_mean_sq_error = np.sqrt(metrics.mean_squared_error(y_test, y_pred))
root_mean_sq_error
```

```
0.897752737512321
4468.426673542113
0.3476580461068184
52706651.33334208
7259.934664536733
```

# Linear Regression

```
sns.distplot(y_test, label="Actual")  
sns.distplot(y_pred, label="Predicted")  
plt.legend()
```



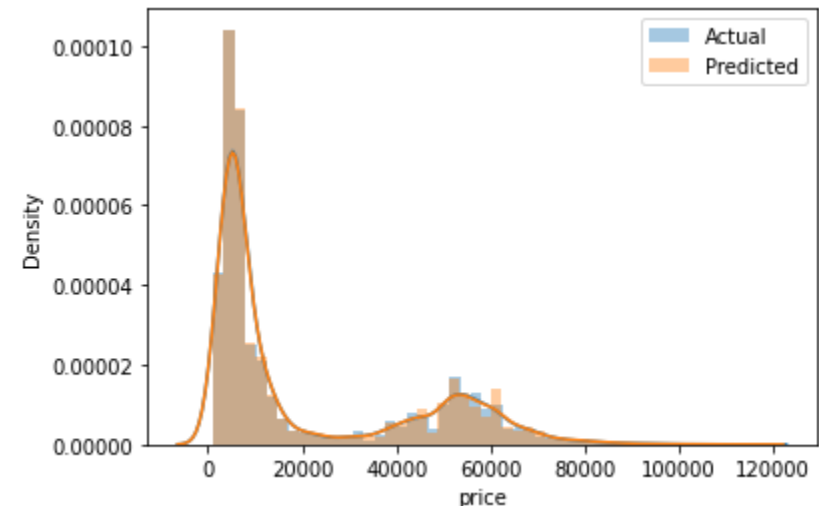
Plotting the graph of actual and predicted price of flight

# Decision Tree Regressor

```
from sklearn.tree import DecisionTreeRegressor
dt=DecisionTreeRegressor()
dt.fit(x_train,y_train)
y_pred=dt.predict(x_test)
r2_score(y_test,y_pred)
mean_abs_error= metrics.mean_absolute_error(y_test,y_pred)
mean_abs_error
from sklearn.metrics import mean_absolute_percentage_error
mean_absolute_percentage_error(y_test, y_pred)
mean_sq_error=metrics.mean_squared_error(y_test,y_pred)
mean_sq_error
root_mean_sq_error = np.sqrt(metrics.mean_squared_error(y_test,y_pred))
root_mean_sq_error
```

```
0.9745774442285287
1219.455742310917
0.07732296917115203
13104876.849009493
3620.0658625237047
```

Mean absolute percentage error is 7.7 percent and RMSE is 3620 which is less than the linear regression model





# Random Forest Regressor

```
from sklearn.ensemble import RandomForestRegressor
rfr=RandomForestRegressor()
rfr.fit(x_train,y_train)
y_pred=rfr.predict(x_test)
r2_score(y_test,y_pred)
mean_abs_error= metrics.mean_absolute_error(y_test,y_pred)
mean_abs_error
from sklearn.metrics import mean_absolute_percentage_error
mean_absolute_percentage_error(y_test, y_pred)
mean_sq_error=metrics.mean_squared_error(y_test,y_pred)
mean_sq_error
root_mean_sq_error = np.sqrt(metrics.mean_squared_error(y_test,y_pred))
root_mean_sq_error
```

```
sns.distplot(y_test,label="Actual")
sns.distplot(y_pred,label="Predicted")
plt.legend()
```

```
0.9845246238799552
1122.6731295238862
0.07319114674216119
7977282.066694117
2824.4082684155487
```

Mean absolute percentage error is 7.3 percent and RMSE is 2824 which is less than the linear regression and decision tree model

