

# Flight\_Price\_Pridiction

April 30, 2023

## 1 Project Name : - Flight Fare Prediction

### 1.1 1) Problem Statement

- This Dataset comprises of flight price taken from Kaggle
- A user can predict the price of the Flight Fare based on input features.
- Prediction results can be useful for traveller to get suggested price

#### 1.1.1 Import required Library

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

#### 1.1.2 Loading Flight Data

```
[2]: df=pd.read_excel("Data_Train.xlsx")
df.head(5)
```

```
[2]:      Airline Date_of_Journey  Source Destination  Route \
0      IndiGo      24/03/2019  Bangalore  New Delhi      BLR → DEL
1      Air India      1/05/2019  Kolkata    Bangalore  CCU → IXR → BBI → BLR
2      Jet Airways      9/06/2019    Delhi      Cochin  DEL → LKO → BOM → COK
3      IndiGo      12/05/2019  Kolkata    Bangalore      CCU → NAG → BLR
4      IndiGo      01/03/2019  Bangalore  New Delhi      BLR → NAG → DEL
```

```
      Dep_Time  Arrival_Time  Duration  Total_Stops  Additional_Info  Price
0      22:20    01:10 22 Mar      2h 50m    non-stop      No info    3897
1      05:50           13:15      7h 25m      2 stops      No info    7662
2      09:25    04:25 10 Jun          19h      2 stops      No info   13882
3      18:05           23:30      5h 25m      1 stop      No info     6218
4      16:50           21:35      4h 45m      1 stop      No info   13302
```

```
[3]: df.shape
```

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

Here Dataset Have :- Observation ==> 10683 and Feature ==> 11

### Summary of the DataSet

```
[4]: df.head(3)
```

```
[4]:
```

	Airline	Date_of_Journey	Source	Destination	Route \
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL → LKO → BOM → COK

	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price
0	22:20	01:10 22 Mar	2h 50m	non-stop	No info	3897
1	05:50	13:15	7h 25m	2 stops	No info	7662
2	09:25	04:25 10 Jun	19h	2 stops	No info	13882

```
[5]: # let's look relationship between numeric feature
df.describe()
```

```
[5]:
```

	Price
count	10683.000000
mean	9087.064121
std	4611.359167
min	1759.000000
25%	5277.000000
50%	8372.000000
75%	12373.000000
max	79512.000000

```
[6]: 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

```

All Feature is Categorical Except Output Feature Price

## 1.2 2) Data Exploring

```

[7]: # define numerical & categorical columns
numeric_features=[feature for feature in df.columns if df[feature].dtype != 'O']
categorical_features=[feature for feature in df.columns if df[feature].dtype == 'O']

# print columns
print(f'We have {len(numeric_features)} numerical features :{numeric_features}')
print(f'We have {len(categorical_features)} categorical features :
      {categorical_features}')

```

We have 1 numerical features : ['Price']

We have 10 categorical features : ['Airline', 'Date\_of\_Journey', 'Source', 'Destination', 'Route', 'Dep\_Time', 'Arrival\_Time', 'Duration', 'Total\_Stops', 'Additional\_Info']

### 1.2.1 Feature Information

- **Airline:** Name of the Airline from which the Ticket is Booked.
- **Date\_of\_Journey:** Date of Journey of the Traveller.
- **Source:** Source from which the Airline Would Departure.
- **Destination:** Destination to Which Airline Would Arrive.
- **Route:** Route of the Airline from Source to Destination.
- **Dep\_Time:** Time at which Flight Would Departure from the Source.
- **Arrival\_Time:** Time at which Flight Would Arrive at the Destination.
- **Duration:** Duration that Airline Takes to fly from Source to Destination.
- **Total\_Stops:** Total No of Stops that Airline takes Between Source and Destination.
- **Additional\_Info:** Any Additional Info about the Airline.
- **Price:** Fare of the Ticket to fly from Source to Destination.

```

[8]: df.value_counts()

```

```

[8]: Airline      Date_of_Journey  Source  Destination  Route
Dep_Time  Arrival_Time  Duration  Total_Stops  Additional_Info
Price
Jet Airways      27/05/2019      Delhi      Cochin      DEL → AMD → BOM
→ COK  19:10      19:00 28 May  23h 50m  2 stops      No info
15129      3
      18/05/2019      Delhi      Cochin      DEL → JAI → BOM

```

```

→ COK 05:30      04:25 19 May 22h 55m 2 stops      In-flight meal not
included 13029    3
Air India                24/06/2019      Delhi      Cochin      DEL → GOI → BOM
→ COK 22:00      19:15 25 Jun 21h 15m 2 stops      No info
9181      3
Jet Airways              9/05/2019      Delhi      Cochin      DEL → JAI → BOM
→ COK 05:30      04:25 10 May 22h 55m 2 stops      In-flight meal not
included 13029    3
                        1/06/2019      Delhi      Cochin      DEL → JAI → BOM
→ COK 05:30      04:25 02 Jun 22h 55m 2 stops      No info
13014     3
..
IndiGo                27/05/2019      Delhi      Cochin      DEL → HYD → COK
05:05      12:10      7h 5m      1 stop      No info      6600
1
                        16:10      11h 5m      1 stop      No info
6600      1
                        06:50      12:10      5h 20m      1 stop      No info
5530      1
                        16:10      9h 20m      1 stop      No info
6674      1
Vistara Premium economy 6/04/2019      Bangalore Delhi      BLR → DEL
11:30      14:20      2h 50m      non-stop      No info      5969
1
Length: 10462, dtype: int64

```

```
[9]: df.count()
```

```

[9]: Airline      10683
Date_of_Journey  10683
Source          10683
Destination     10683
Route           10682
Dep_Time        10683
Arrival_Time    10683
Duration        10683
Total_Stops     10682
Additional_Info  10683
Price           10683
dtype: int64

```

```

[10]: # proportion of count data of each categorical columns
for col in categorical_features:
    print(df[col].value_counts(normalize=True)*100)
    print('-----')

```

```

Jet Airways      36.029205
IndiGo           19.217448

```

Air India	16.399888
Multiple carriers	11.195357
SpiceJet	7.657025
Vistara	4.483759
Air Asia	2.986053
GoAir	1.815969
Multiple carriers Premium economy	0.121689
Jet Airways Business	0.056164
Vistara Premium economy	0.028082
Trujet	0.009361

Name: Airline, dtype: float64

```
-----
18/05/2019    4.717776
6/06/2019    4.708415
21/05/2019    4.652251
9/06/2019    4.633530
12/06/2019    4.614809
9/05/2019    4.530563
21/03/2019    3.959562
15/05/2019    3.791070
27/05/2019    3.575775
27/06/2019    3.323037
24/06/2019    3.285594
1/06/2019     3.201348
3/06/2019     3.117102
15/06/2019    3.070299
24/03/2019    3.023495
6/03/2019     2.883085
27/03/2019    2.798839
24/05/2019    2.677151
6/05/2019     2.639708
1/05/2019     2.592905
12/05/2019    2.424413
1/04/2019     2.405691
3/03/2019     2.040625
9/03/2019     1.872133
15/03/2019    1.516428
18/03/2019    1.460264
01/03/2019    1.422821
12/03/2019    1.329215
9/04/2019     1.170083
3/04/2019     1.029673
21/06/2019    1.020313
18/06/2019    0.982870
09/03/2019    0.954788
6/04/2019     0.936067
03/03/2019    0.907985
06/03/2019    0.889263
```

```

27/04/2019    0.879903
24/04/2019    0.861181
3/05/2019     0.842460
15/04/2019    0.833099
21/04/2019    0.767575
18/04/2019    0.627165
12/04/2019    0.589722
1/03/2019     0.439951
Name: Date_of_Journey, dtype: float64
-----
Delhi         42.469344
Kolkata       26.874473
Bangalore     20.565384
Mumbai        6.524385
Chennai       3.566414
Name: Source, dtype: float64
-----
Cochin        42.469344
Bangalore     26.874473
Delhi         11.841243
New Delhi     8.724141
Hyderabad     6.524385
Kolkata       3.566414
Name: Destination, dtype: float64
-----
DEL → BOM → COK          22.243026
BLR → DEL                14.529114
CCU → BOM → BLR          9.164950
CCU → BLR                6.777757
BOM → HYD                5.813518

...
CCU → VTZ → BLR          0.009362
CCU → IXZ → MAA → BLR    0.009362
BOM → COK → MAA → HYD    0.009362
BOM → CCU → HYD          0.009362
BOM → BBI → HYD          0.009362
Name: Route, Length: 128, dtype: float64
-----
18:55    2.181035
17:00    2.124871
07:05    1.918937
10:00    1.900215
07:10    1.890855

...
16:25    0.009361
01:35    0.009361
21:35    0.009361
04:15    0.009361

```

```

03:00      0.009361
Name: Dep_Time, Length: 222, dtype: float64
-----
19:00      3.959562
21:00      3.369840
19:15      3.117102
16:10      1.441543
12:35      1.142001
...
00:25 02 Jun 0.009361
08:55 13 Mar 0.009361
11:05 19 May 0.009361
12:30 22 May 0.009361
21:20 13 Mar 0.009361
Name: Arrival_Time, Length: 1343, dtype: float64
-----
2h 50m      5.148367
1h 30m      3.613217
2h 45m      3.154545
2h 55m      3.154545
2h 35m      3.079659
...
31h 30m      0.009361
30h 25m      0.009361
42h 5m       0.009361
4h 10m       0.009361
47h 40m      0.009361
Name: Duration, Length: 368, dtype: float64
-----
1 stop      52.658678
non-stop    32.681146
2 stops     14.229545
3 stops      0.421269
4 stops      0.009362
Name: Total_Stops, dtype: float64
-----
No info      78.114762
In-flight meal not included 18.552841
No check-in baggage included 2.995413
1 Long layover 0.177853
Change airports 0.065525
Business class 0.037443
No Info      0.028082
1 Short layover 0.009361
Red-eye flight 0.009361
2 Long layover 0.009361
Name: Additional_Info, dtype: float64
-----

```

```
[11]: for i in categorical_features:
      print(df[i].value_counts(normalize=True)*100)
```

```
Jet Airways          36.029205
IndiGo               19.217448
Air India            16.399888
Multiple carriers    11.195357
SpiceJet             7.657025
Vistara              4.483759
Air Asia             2.986053
GoAir                1.815969
Multiple carriers Premium economy 0.121689
Jet Airways Business 0.056164
Vistara Premium economy 0.028082
Trujet               0.009361
```

```
Name: Airline, dtype: float64
```

```
18/05/2019    4.717776
6/06/2019     4.708415
21/05/2019    4.652251
9/06/2019     4.633530
12/06/2019    4.614809
9/05/2019     4.530563
21/03/2019    3.959562
15/05/2019    3.791070
27/05/2019    3.575775
27/06/2019    3.323037
24/06/2019    3.285594
1/06/2019     3.201348
3/06/2019     3.117102
15/06/2019    3.070299
24/03/2019    3.023495
6/03/2019     2.883085
27/03/2019    2.798839
24/05/2019    2.677151
6/05/2019     2.639708
1/05/2019     2.592905
12/05/2019    2.424413
1/04/2019     2.405691
3/03/2019     2.040625
9/03/2019     1.872133
15/03/2019    1.516428
18/03/2019    1.460264
01/03/2019    1.422821
12/03/2019    1.329215
9/04/2019     1.170083
3/04/2019     1.029673
21/06/2019    1.020313
18/06/2019    0.982870
```



09/03/2019	0.954788
6/04/2019	0.936067
03/03/2019	0.907985
06/03/2019	0.889263
27/04/2019	0.879903
24/04/2019	0.861181
3/05/2019	0.842460
15/04/2019	0.833099
21/04/2019	0.767575
18/04/2019	0.627165
12/04/2019	0.589722
1/03/2019	0.439951

Name: Date\_of\_Journey, dtype: float64

Delhi	42.469344
Kolkata	26.874473
Banglore	20.565384
Mumbai	6.524385
Chennai	3.566414

Name: Source, dtype: float64

Cochin	42.469344
Banglore	26.874473
Delhi	11.841243
New Delhi	8.724141
Hyderabad	6.524385
Kolkata	3.566414

Name: Destination, dtype: float64

DEL → BOM → COK	22.243026
BLR → DEL	14.529114
CCU → BOM → BLR	9.164950
CCU → BLR	6.777757
BOM → HYD	5.813518
...	
CCU → VTZ → BLR	0.009362
CCU → IXZ → MAA → BLR	0.009362
BOM → COK → MAA → HYD	0.009362
BOM → CCU → HYD	0.009362
BOM → BBI → HYD	0.009362

Name: Route, Length: 128, dtype: float64

18:55	2.181035
17:00	2.124871
07:05	1.918937
10:00	1.900215
07:10	1.890855
...	
16:25	0.009361
01:35	0.009361
21:35	0.009361
04:15	0.009361

```

03:00      0.009361
Name: Dep_Time, Length: 222, dtype: float64
19:00      3.959562
21:00      3.369840
19:15      3.117102
16:10      1.441543
12:35      1.142001
...
00:25 02 Jun 0.009361
08:55 13 Mar 0.009361
11:05 19 May 0.009361
12:30 22 May 0.009361
21:20 13 Mar 0.009361
Name: Arrival_Time, Length: 1343, dtype: float64
2h 50m      5.148367
1h 30m      3.613217
2h 45m      3.154545
2h 55m      3.154545
2h 35m      3.079659
...
31h 30m      0.009361
30h 25m      0.009361
42h 5m       0.009361
4h 10m       0.009361
47h 40m      0.009361
Name: Duration, Length: 368, dtype: float64
1 stop      52.658678
non-stop    32.681146
2 stops     14.229545
3 stops      0.421269
4 stops      0.009362
Name: Total_Stops, dtype: float64
No info      78.114762
In-flight meal not included 18.552841
No check-in baggage included 2.995413
1 Long layover 0.177853
Change airports 0.065525
Business class 0.037443
No Info      0.028082
1 Short layover 0.009361
Red-eye flight 0.009361
2 Long layover 0.009361
Name: Additional_Info, dtype: float64

```

### 1.3 Univariate Analysis

- The term univariate analysis refers to the analysis of one variable prefix “uni” means “one.” The purpose of univariate analysis is to understand the distribution of values for a single

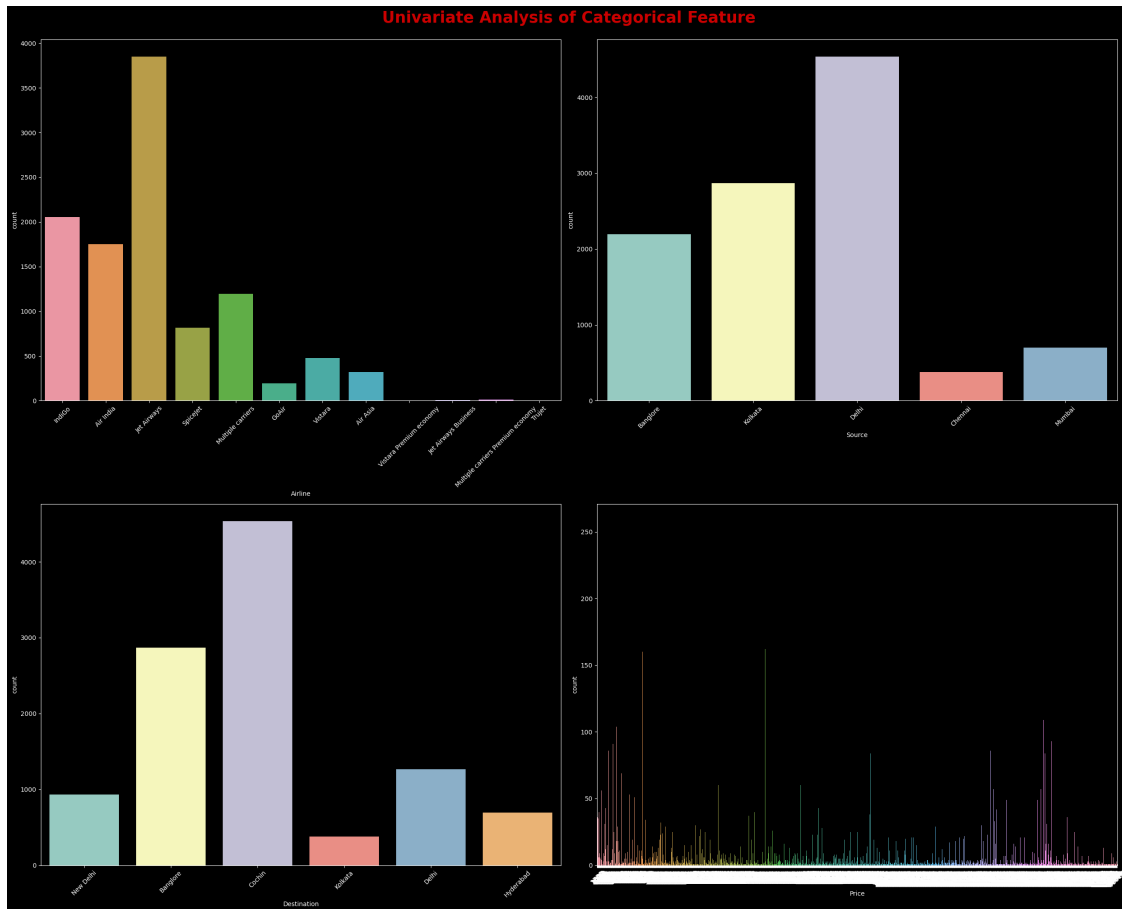
variable.

### 1.3.1 Categorical Features

```
[12]: df.columns.unique()
```

```
[12]: Index(['Airline', 'Date_of_Journey', 'Source', 'Destination', 'Route',  
          'Dep_Time', 'Arrival_Time', 'Duration', 'Total_Stops',  
          'Additional_Info', 'Price'],  
          dtype='object')
```

```
[97]: plt.figure(figsize=(25,20))  
plt.suptitle('Univariate Analysis of Categorical_  
          ↪Feature', fontsize=25, fontweight='bold', alpha=0.8, y=1., color='red',)  
cat1=['Airline', 'Source', 'Destination', 'Price',]  
plt.style.use('dark_background')  
  
plt.rcParams.update({  
    'text.color': 'white'           # set the color of the text  
})  
for i in range(0, len(cat1)):  
    plt.subplot(2,2,i+1)  
    sns.countplot(x=df[cat1[i]])  
    plt.xlabel(cat1[i])  
    plt.xticks(rotation=45)  
    plt.tight_layout()
```



## 2 Multivariate Analysis

- Multivariate analysis is the analysis of more than one variable

### 2.0.1 Check Multicollinearity for Categorical features

- A chi-squared test (also chi-square or 2 test) is a statistical hypothesis test that is valid to perform when the test statistic is chi-squared distributed under the null hypothesis, specifically Pearson's chi-squared test
- A chi-square statistic is one way to show a relationship between two categorical variables.
- Here we test correlation of Categorical columns with Target column i.e Price

```
[14]: from scipy.stats import chi2_contingency
chi2_test=[]
for feature in categorical_features:
    if chi2_contingency(pd.crosstab(df['Price'],df[feature]))[1] <0.05:
        chi2_test.append('Rejet Null Hypothesis')
```

```

else:
    chi2_test.append('Fail to Reject Null Hypothesis')
result=pd.DataFrame(data=[categorical_features,chi2_test]).T
result.columns=['Column','Hypothesis Result']
result

```

```

[14]:
      Column      Hypothesis Result
0      Airline  Rejet Null Hypothesis
1  Date_of_Journey  Rejet Null Hypothesis
2        Source  Rejet Null Hypothesis
3   Destination  Rejet Null Hypothesis
4        Route  Rejet Null Hypothesis
5     Dep_Time  Rejet Null Hypothesis
6  Arrival_Time  Rejet Null Hypothesis
7     Duration  Rejet Null Hypothesis
8   Total_Stops  Rejet Null Hypothesis
9  Additional_Info  Rejet Null Hypothesis

```

```

[15]: df.isnull()

```

```

[15]:
      Airline  Date_of_Journey  Source  Destination  Route  Dep_Time  \
0      False                False  False          False  False  False
1      False                False  False          False  False  False
2      False                False  False          False  False  False
3      False                False  False          False  False  False
4      False                False  False          False  False  False
...
10678  False                False  False          False  False  False
10679  False                False  False          False  False  False
10680  False                False  False          False  False  False
10681  False                False  False          False  False  False
10682  False                False  False          False  False  False

      Arrival_Time  Duration  Total_Stops  Additional_Info  Price
0      False      False      False      False      False  False
1      False      False      False      False      False  False
2      False      False      False      False      False  False
3      False      False      False      False      False  False
4      False      False      False      False      False  False
...
10678  False      False      False      False      False  False
10679  False      False      False      False      False  False
10680  False      False      False      False      False  False
10681  False      False      False      False      False  False
10682  False      False      False      False      False  False

```

```

[10683 rows x 11 columns]

```

```
[16]: df.isnull().sum()
```

```
[16]: 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
```

```
[17]: df.dropna(inplace=True)
```

```
[18]: df.isnull().sum()
```

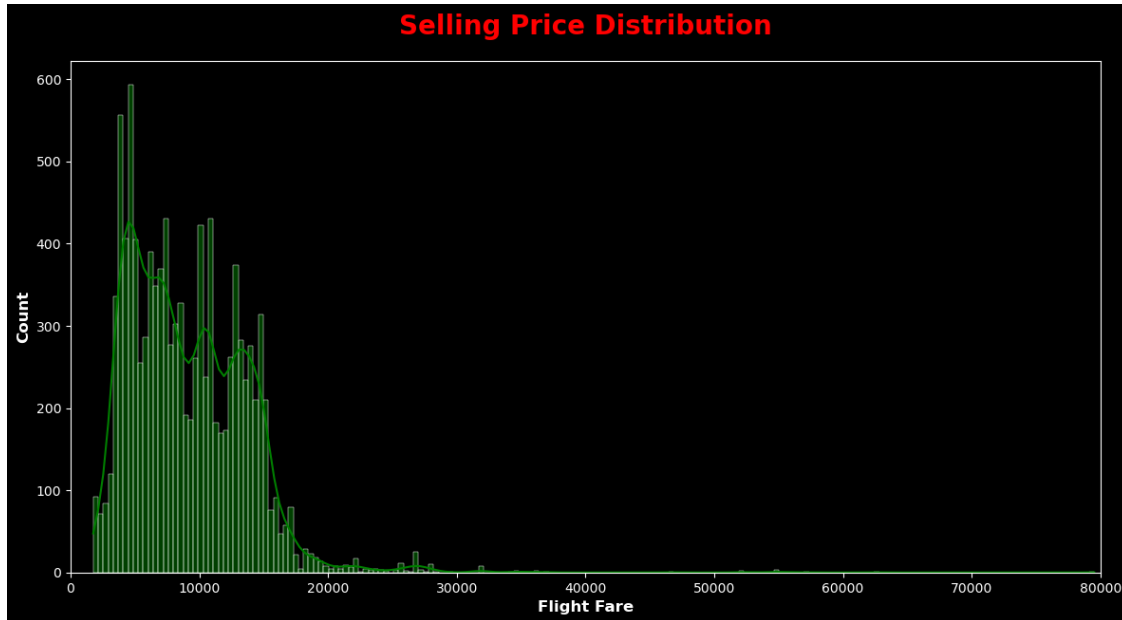
```
[18]: Airline          0
      Date_of_Journey  0
      Source          0
      Destination     0
      Route           0
      Dep_Time        0
      Arrival_Time    0
      Duration        0
      Total_Stops     0
      Additional_Info  0
      Price           0
      dtype: int64
```

## 2.1 Initial Analysis Report

Report \* Jet Airways has highest customer footfall followed by Indigo and Air India .  
\* Jet Airways has a market Share of 36.03 % followed by Indigo which has a market share of 19.22 % and Air India Which has market share of 16.40 % . \* Delhi has the highest footfall for source and Cochin has the highest footfall for Destination .

```
[19]: plt.subplots(figsize=(14,7))
      sns.histplot(df.Price, bins=200, kde=True, color = 'green')
      plt.style.use('dark_background')
      plt.rcParams.update({'text.color': 'white'})
      plt.title("Selling Price Distribution", weight="bold", fontsize=20,
        ↳pad=20, color='red')
      plt.ylabel("Count", weight="bold", fontsize=12)
      plt.xlabel("Flight Fare", weight="bold", fontsize=12)
      plt.xlim(0,80000)
```

```
plt.show()
```



- From the chart it is clear that the Target Variable is Skewed

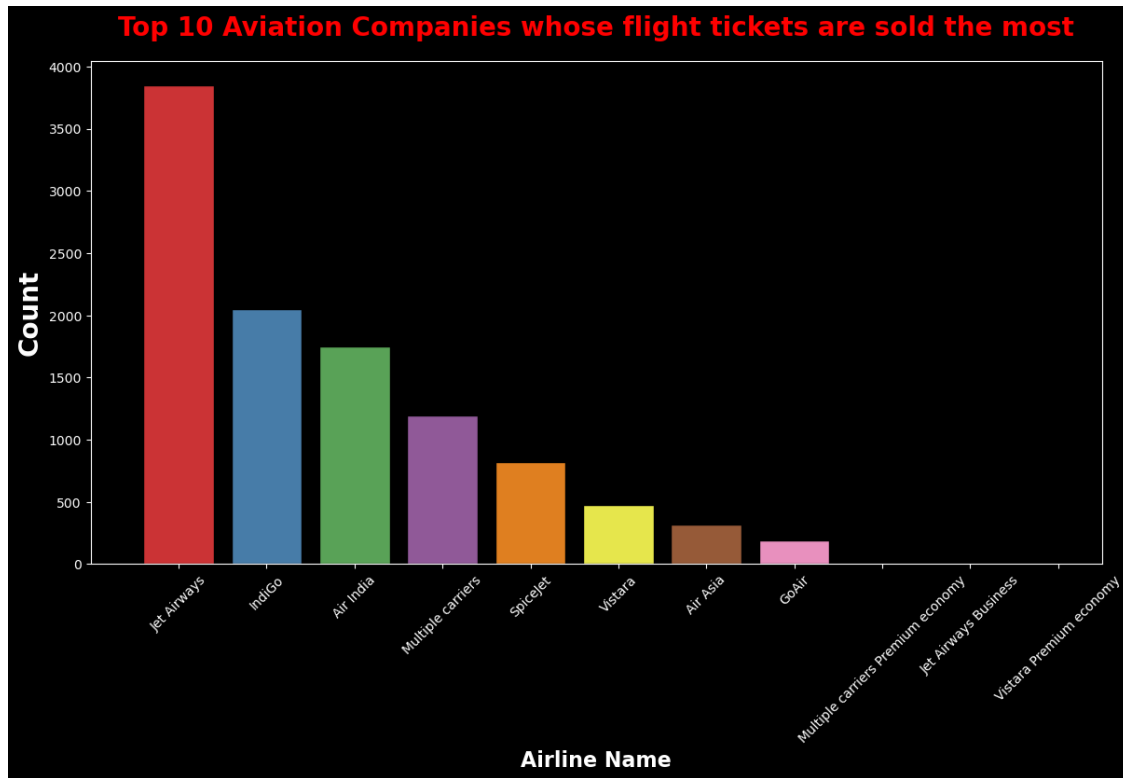
```
[20]: df.Airline.value_counts()[0:10]
```

```
[20]: Jet Airways          3849
      IndiGo              2053
      Air India           1751
      Multiple carriers    1196
      SpiceJet            818
      Vistara             479
      Air Asia            319
      GoAir               194
      Multiple carriers Premium economy  13
      Jet Airways Business    6
      Name: Airline, dtype: int64
```

- Most of the sold tickets are of the Jet Airways Airline

```
[21]: plt.subplots(figsize=(14,7))
      sns.countplot(x="Airline", data=df,ec = "black",palette="Set1",order =
      ↪df['Airline'].value_counts().index)
      plt.style.use('dark_background')
      plt.rcParams.update({'text.color':'white'})
      plt.title("Top 10 Aviation Companies whose flight tickets are sold the most",
      ↪weight="bold",fontsize=20, pad=20,color='red')
```

```
plt.ylabel("Count", weight="bold", fontsize=20)
plt.xlabel("Airline Name", weight="bold", fontsize=16)
plt.xticks(rotation= 45)
plt.xlim(-1,10.5)
plt.show()
```



- Check mean price of Jet Airways whose flight tickets are sold the most

```
[22]: jet_airways = df[df['Airline'] == 'Jet Airways']['Price'].mean()
print(f'The mean price of Jet Airways Flight Tickets is {jet_airways:.2f}₹Rupees')
```

The mean price of Jet Airways Flight Tickets is 11643.92 Rupees

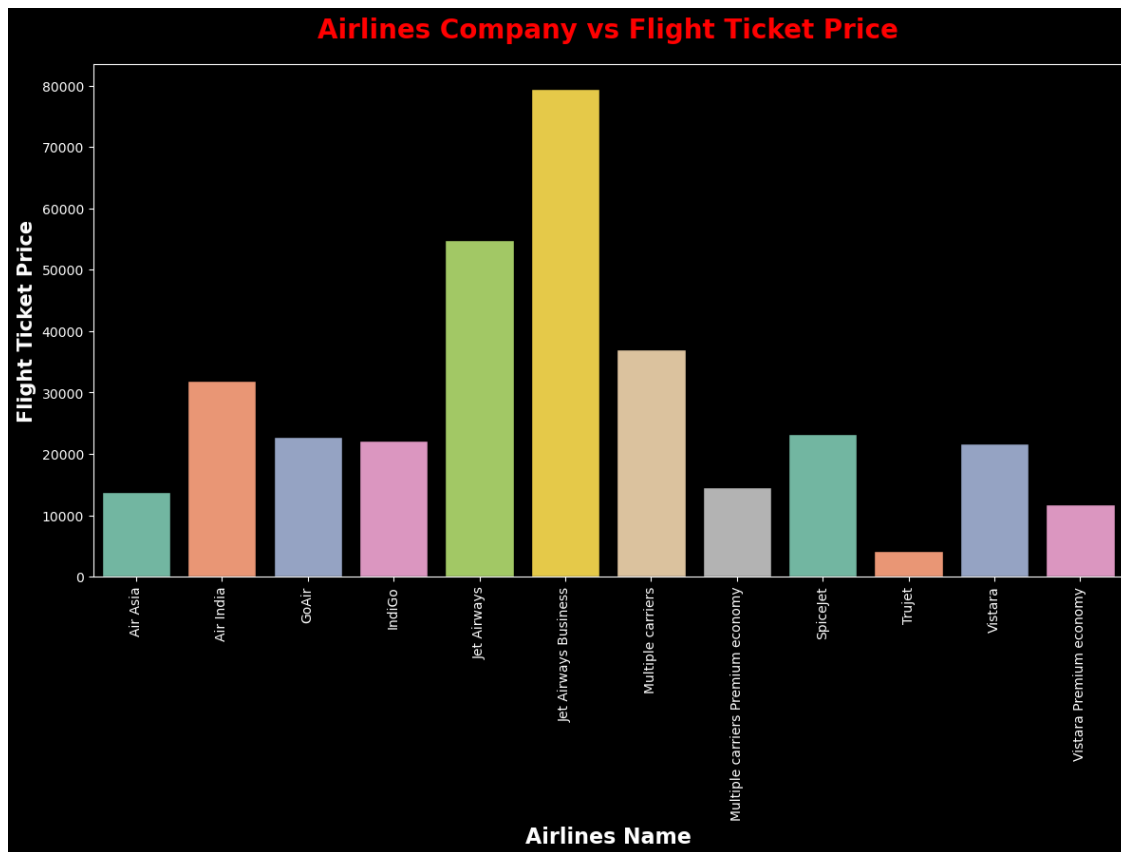
**Report:** \* As per the Chart these are top 10 aviation companies whose tickets are sold the most. \* Of the total flight tickets sold Jet Airways has the highest share followed by Indigo . \* Mean Price of Jet Airways Flight Ticket is Rs 11,643.92. \* This Feature has impact on the Target Variable.

```
[23]: aviation_company_airline = df.groupby('Airline').Price.max()
aviation_company= aviation_company_airline.to_frame().
sort_values('Price',ascending=False)[0:10]
aviation_company
```



Airline	Price
Jet Airways Business	79512
Jet Airways	54826
Multiple carriers	36983
Air India	31945
SpiceJet	23267
GoAir	22794
IndiGo	22153
Vistara	21730
Multiple carriers Premium economy	14629
Air Asia	13774

```
[24]: plt.subplots(figsize=(14,7))
sns.barplot(x=aviation_company_airline.index, y=aviation_company_airline.
           ↪values,ec = "black",palette="Set2")
plt.style.use('dark_background')
plt.rcParams.update({'text.color':'white'})
plt.title("Airlines Company vs Flight Ticket Price", weight="bold",fontsize=20,
           ↪pad=20,color='red')
plt.ylabel("Flight Ticket Price", weight="bold", fontsize=15)
plt.xlabel("Airlines Name", weight="bold", fontsize=16)
plt.xticks(rotation=90)
plt.show()
```



**Report:** \* Costliest Flight Tickets Sold is of Jet Airways Business . \* Second Most Costliest Flight Tickets Sold is of Jet Airways . \* As can be seen, the airline's name is important. The most expensive option is 'JetAirways Business.' The cost of other carriers varies as well. \* We'll use one-hot encoding to handle the Airline variable because it's Nominal Categorical Data (airline names have no order of any kind).

### 3 Extracting Date, Month & Day from Date\_Of\_Journey Features

```
[25]: df.head(2)
```

```
[25]:
```

	Airline	Date_of_Journey	Source	Destination	Route \
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR

	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price
0	22:20	01:10 22 Mar	2h 50m	non-stop	No info	3897
1	05:50	13:15	7h 25m	2 stops	No info	7662

```
[26]: df["journey_Date"] = pd.to_datetime(df['Date_of_Journey'], format= "%d/%m/%Y").
      ↪dt.day
      df["journey_Month"] = pd.to_datetime(df['Date_of_Journey'], format= "%d/%m/%Y").
      ↪dt.month
```

```
[27]: df.head(2)
```

```
[27]:
```

	Airline	Date_of_Journey	Source	Destination	Route	\
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	

	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price	\
0	22:20	01:10	22 Mar	2h 50m	non-stop	No info	3897
1	05:50	13:15	7h 25m	2 stops	No info	7662	

	journey_Date	journey_Month
0	24	3
1	1	5

```
[28]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10682 entries, 0 to 10682
Data columns (total 13 columns):
#   Column                Non-Null Count  Dtype
---  -
0   Airline                10682 non-null  object
1   Date_of_Journey        10682 non-null  object
2   Source                 10682 non-null  object
3   Destination            10682 non-null  object
4   Route                  10682 non-null  object
5   Dep_Time               10682 non-null  object
6   Arrival_Time           10682 non-null  object
7   Duration               10682 non-null  object
8   Total_Stops            10682 non-null  object
9   Additional_Info        10682 non-null  object
10  Price                  10682 non-null  int64
11  journey_Date           10682 non-null  int64
12  journey_Month          10682 non-null  int64
dtypes: int64(3), object(10)
memory usage: 1.1+ MB
```

- Above Date ,Months and Year Have data type object so converting into int
- Drop original Categorical Features

```
[29]: df.head(2)
```

```
[29]:
```

	Airline	Date_of_Journey	Source	Destination	Route	\
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	

	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price	\
0	22:20	01:10 22 Mar	2h 50m	non-stop	No info	3897	
1	05:50	13:15	7h 25m	2 stops	No info	7662	

	journey_Date	journey_Month
0	24	3
1	1	5

Drop Feature Date\_of\_journer

```
[30]: df.drop('Date_of_Journey',axis=1,inplace=True)
```

```
[31]: df.head(3)
```

```
[31]:
```

	Airline	Source	Destination	Route	Dep_Time	\
0	IndiGo	Banglore	New Delhi	BLR → DEL	22:20	
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	
2	Jet Airways	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	

	Arrival_Time	Duration	Total_Stops	Additional_Info	Price	journey_Date	\
0	01:10 22 Mar	2h 50m	non-stop	No info	3897	24	
1	13:15	7h 25m	2 stops	No info	7662	1	
2	04:25 10 Jun	19h	2 stops	No info	13882	9	

	journey_Month
0	3
1	5
2	6

### 3.0.1 Now Exatraction for Departure Time

```
[32]: # Extracting Hours
df['Dep_hour']=pd.to_datetime(df['Dep_Time']).dt.hour #pd.to_datetime

#Extracting minutes
df['Dep_min']=pd.to_datetime(df['Dep_Time']).dt.minute

#Now we will drop the dep_time, no use
df.drop(['Dep_Time'],axis=1,inplace=True)
```

### 3.0.2 Now Extraction for Arrival\_Time

```
[33]: # Extracting Hours
df['Arrival_hour']=pd.to_datetime(df['Arrival_Time']).dt.hour #pd.to_datetime

#Extracting minutes
df['Arrival_min']=pd.to_datetime(df['Arrival_Time']).dt.minute

#Now we will drop the dep_time, no use
df.drop(['Arrival_Time'],axis=1,inplace=True)
```

```
[34]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10682 entries, 0 to 10682
Data columns (total 14 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   Route                  10682 non-null  object
4   Duration                10682 non-null  object
5   Total_Stops            10682 non-null  object
6   Additional_Info        10682 non-null  object
7   Price                  10682 non-null  int64
8   journey_Date           10682 non-null  int64
9   journey_Month          10682 non-null  int64
10  Dep_hour               10682 non-null  int64
11  Dep_min                10682 non-null  int64
12  Arrival_hour           10682 non-null  int64
13  Arrival_min            10682 non-null  int64
dtypes: int64(7), object(7)
memory usage: 1.2+ MB
```

- DataType conversion again

```
[35]: df.head()
```

```
[35]:
```

	Airline	Source	Destination	Route	Duration	\
0	IndiGo	Banglore	New Delhi	BLR → DEL	2h 50m	
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	7h 25m	
2	Jet Airways	Delhi	Cochin	DEL → LKO → BOM → COK	19h	
3	IndiGo	Kolkata	Banglore	CCU → NAG → BLR	5h 25m	
4	IndiGo	Banglore	New Delhi	BLR → NAG → DEL	4h 45m	

	Total_Stops	Additional_Info	Price	journey_Date	journey_Month	Dep_hour	\
0	non-stop	No info	3897	24	3	22	
1	2 stops	No info	7662	1	5	5	

2	2 stops	No info	13882	9	6	9
3	1 stop	No info	6218	12	5	18
4	1 stop	No info	13302	1	3	16

	Dep_min	Arrival_hour	Arrival_min
0	20	1	10
1	50	13	15
2	25	4	25
3	5	23	30
4	50	21	35

```
[36]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10682 entries, 0 to 10682
Data columns (total 14 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   Route                  10682 non-null  object
4   Duration                10682 non-null  object
5   Total_Stops            10682 non-null  object
6   Additional_Info        10682 non-null  object
7   Price                  10682 non-null  int64
8   journey_Date           10682 non-null  int64
9   journey_Month          10682 non-null  int64
10  Dep_hour               10682 non-null  int64
11  Dep_min                10682 non-null  int64
12  Arrival_hour           10682 non-null  int64
13  Arrival_min            10682 non-null  int64
dtypes: int64(7), object(7)
memory usage: 1.2+ MB
```

### 3.0.3 Now For Duration Column

```
[37]: df.head(2)
```

```
[37]:
```

	Airline	Source	Destination	Route	Duration	\
0	IndiGo	Bangalore	New Delhi	BLR → DEL	2h 50m	
1	Air India	Kolkata	Bangalore	CCU → IXR → BBI → BLR	7h 25m	

	Total_Stops	Additional_Info	Price	journey_Date	journey_Month	Dep_hour	\
0	non-stop	No info	3897	24	3	22	
1	2 stops	No info	7662	1	5	5	

	Dep_min	Arrival_hour	Arrival_min
--	---------	--------------	-------------

0	20	1	10
1	50	13	15

```
[38]: # Assigning and converting Duration column into list to extract hours and
      ↪ minutes separately
duration = list(df["Duration"])
for i in range(len(duration)):
    if len(duration[i].split()) != 2: # Check if duration contains only hour or
    ↪ mins
        if "h" in duration[i]:
            duration[i] = duration[i].strip() + " 0m" # Adds 0 minute
        else:
            duration[i] = "0h " + duration[i] # Adds 0 hour

duration_hours = []
duration_mins = []
for i in range(len(duration)):
    duration_hours.append(int(duration[i].split(sep = "h")[0])) # Extract
    ↪ hours from duration
    duration_mins.append(int(duration[i].split(sep = "m")[0].split()[-1])) #
    ↪ Extracts only minutes from duration
```

- Adding duration\_hours and duration\_mins into DataFrame

```
[39]: df["Duration_hours"] = duration_hours
df["Duration_mins"] = duration_mins

#we will remove the Duration column
df.drop(['Duration'],axis=1,inplace=True)
```

```
[40]: df.head(3)
```

```
[40]:
```

	Airline	Source	Destination	Route	Total_Stops	\
0	IndiGo	Banglore	New Delhi	BLR → DEL	non-stop	
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	2 stops	
2	Jet Airways	Delhi	Cochin	DEL → LKO → BOM → COK	2 stops	

	Additional_Info	Price	journey_Date	journey_Month	Dep_hour	Dep_min	\
0	No info	3897	24	3	22	20	
1	No info	7662	1	5	5	50	
2	No info	13882	9	6	9	25	

	Arrival_hour	Arrival_min	Duration_hours	Duration_mins
0	1	10	2	50
1	13	15	7	25
2	4	25	19	0

```
[41]: df.info()
```

```

<class 'pandas.core.frame.DataFrame'>
Int64Index: 10682 entries, 0 to 10682
Data columns (total 15 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   Route                  10682 non-null  object
4   Total_Stops            10682 non-null  object
5   Additional_Info        10682 non-null  object
6   Price                  10682 non-null  int64
7   journey_Date           10682 non-null  int64
8   journey_Month          10682 non-null  int64
9   Dep_hour               10682 non-null  int64
10  Dep_min                10682 non-null  int64
11  Arrival_hour           10682 non-null  int64
12  Arrival_min            10682 non-null  int64
13  Duration_hours         10682 non-null  int64
14  Duration_mins          10682 non-null  int64
dtypes: int64(9), object(6)
memory usage: 1.3+ MB

```

## 4 Handling Categorical Data:

Airline, Source, Destination, Route, Total\_Stops, Additional\_info are the categorical variables we have in our data.

Let's handle each one by one.

Nominal data → are not in any order → OneHotEncoder is used in this case

Ordinal data → are in order → LabelEncoder is used in this case

Trying to find out unique values in column Airline and counts of the unique values as well.

### 4.1 Boxplots

Airline vs Price:

- Let's see how the Airline variable is related to the Price variable.

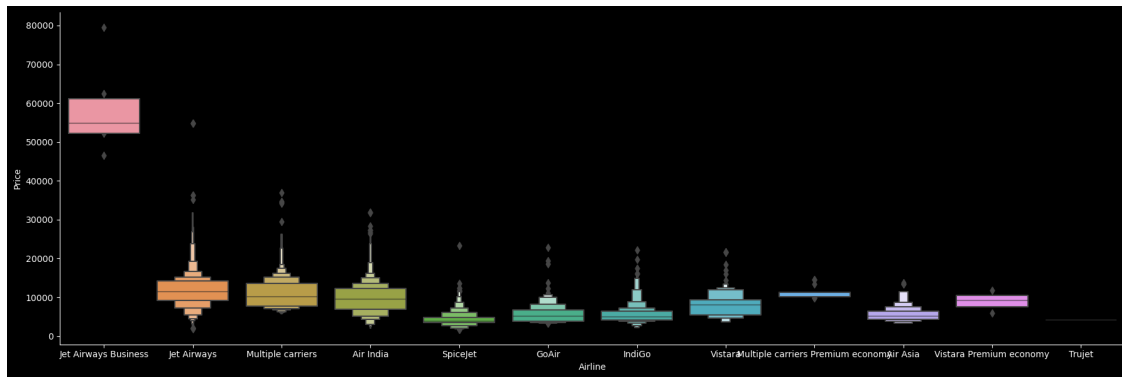
Airline vs Price

```

[42]: sns.catplot(y = "Price", x = "Airline", data = df.sort_values("Price",
    ↪ascending = False), kind="boxen", height = 6, aspect = 3)
plt.style.use('dark_background')
plt.rcParams.update({'text.color': 'white'})
plt.show()

```





- From the above we can see that Jet Airways Business has premium flight fares as compared to other Airlines
- Some Outlier value present above see Diagram

```
[43]: #OneHotEncoding -----> Nominal data
Airline = df[["Airline"]]
Airline = pd.get_dummies(df['Airline'],drop_first=False)
Airline.head()
```

```
[43]:
```

	Air Asia	Air India	GoAir	IndiGo	Jet Airways	Jet Airways Business	\
0	0	0	0	1	0	0	
1	0	1	0	0	0	0	
2	0	0	0	0	1	0	
3	0	0	0	1	0	0	
4	0	0	0	1	0	0	

	Multiple carriers	Multiple carriers Premium economy	SpiceJet	Trujet	\
0	0	0	0	0	
1	0	0	0	0	
2	0	0	0	0	
3	0	0	0	0	
4	0	0	0	0	

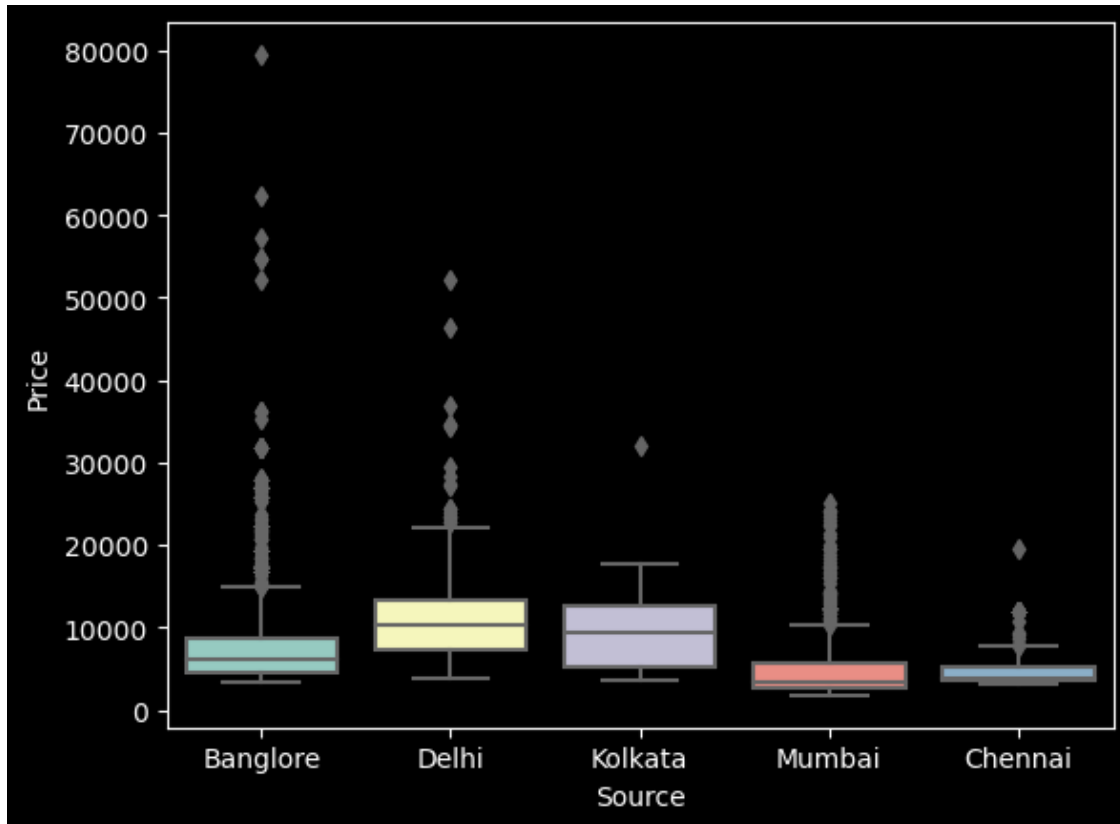
  

	Vistara	Vistara Premium economy
0	0	0
1	0	0
2	0	0
3	0	0
4	0	0

**Source vs Destination:** \* Again, the variables ‘Source’ and ‘Destination’ are Nominal Categorical Data. To deal with these two variables, we’ll employ One-Hot encoding once more.

**Source vs Price**

```
[44]: sns.boxplot(y = "Price", x = "Source", data = df.sort_values("Price", ascending_u
      ↪= False))
plt.style.use('dark_background')
plt.rcParams.update({'text.color': 'white'})
plt.show()
```



- From the Above we can see that Flights Originating From Bangalore has high flight fares as compared to other sources from where flights are originating

```
[45]: #OneHotEncoding ----> Nominal data
Source = df[["Source"]]
Source = pd.get_dummies(df['Source'],drop_first=True)
Source.head()
```

```
[45]:
```

	Chennai	Delhi	Kolkata	Mumbai
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	0

Now For Destination

```
[46]: # As Destination is Nominal Categorical data we will perform OneHotEncoding
```

```
Destination = df[["Destination"]]

Destination = pd.get_dummies(Destination, drop_first = True)

Destination.head()
```

```
[46]:
```

	Destination_Cochin	Destination_Delhi	Destination_Hyderabad	\
0	0	0	0	
1	0	0	0	
2	1	0	0	
3	0	0	0	
4	0	0	0	

	Destination_Kolkata	Destination_New Delhi
0	0	1
1	0	0
2	0	0
3	0	0
4	0	1

**Variable route:** \* The journey's path is represented by the route variable. \* I opted to remove this field because the 'Total Stops' value captures whether the flight is direct or connected.

```
[47]: # dropping column, because Additinal_info has since 80 % has no information
# Route---> is related to no of stops
df.drop(["Route", "Additional_Info"], axis = 1, inplace = True)
```

```
[48]: df.head(3)
```

```
[48]:
```

	Airline	Source	Destination	Total_Stops	Price	journey_Date	\
0	IndiGo	Banglore	New Delhi	non-stop	3897		24
1	Air India	Kolkata	Banglore	2 stops	7662		1
2	Jet Airways	Delhi	Cochin	2 stops	13882		9

	journey_Month	Dep_hour	Dep_min	Arrival_hour	Arrival_min	\
0	3	22	20	1	10	
1	5	5	50	13	15	
2	6	9	25	4	25	

	Duration_hours	Duration_mins
0	2	50
1	7	25
2	19	0

**Total\_Stops Variable:** \* Non-stop refers to a flight with no stops, i.e. a straight flight. It is self-evident that other values have the same meaning. We can see that it's Ordinal Categorical Data, thus we'll use LabelEncoder to deal with it.

```
[49]: df['Total_Stops'].unique()
```

```
[49]: array(['non-stop', '2 stops', '1 stop', '3 stops', '4 stops'],  
      dtype=object)
```

```
[50]: df['Total_Stops'].value_counts()  
# As this is case of Ordinal Categorical type we perform LabelEncoder  
#we replace the values in key values  
df.replace({'non-stop':0, '1 stop':1, '2 stops':2, '3 stops':3, '4 stops':  
↪4}, inplace=True)  
df.head()
```

```
[50]:
```

	Airline	Source	Destination	Total_Stops	Price	journey_Date	\
0	IndiGo	Banglore	New Delhi	0	3897		24
1	Air India	Kolkata	Banglore	2	7662		1
2	Jet Airways	Delhi	Cochin	2	13882		9
3	IndiGo	Kolkata	Banglore	1	6218		12
4	IndiGo	Banglore	New Delhi	1	13302		1

	journey_Month	Dep_hour	Dep_min	Arrival_hour	Arrival_min	\
0	3	22	20	1	10	
1	5	5	50	13	15	
2	6	9	25	4	25	
3	5	18	5	23	30	
4	3	16	50	21	35	

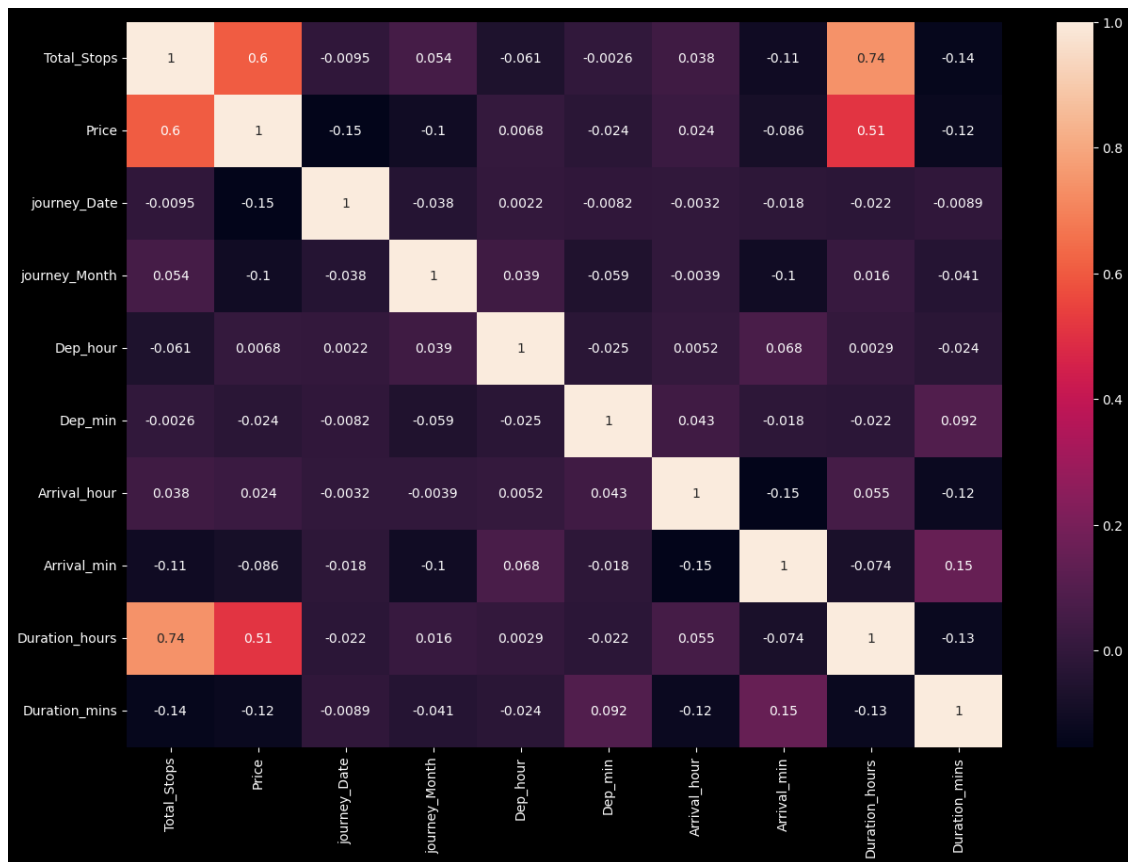
  

	Duration_hours	Duration_mins
0	2	50
1	7	25
2	19	0
3	5	25
4	4	45

**Correlation:** \* Correlation is a technique for determining the link between two variables, which is useful in real life since it allows us to forecast the value of one variable using other factors that are connected with it. Because two variables are involved, it is a sort of bivariate statistic.

```
[51]: # Heatmap  
plt.figure(figsize=(15,10))  
plt.style.use('dark_background')  
plt.rcParams.update({'text.color':'white'})  
sns.heatmap(df.corr(),annot = True,) #cmap = "RdYlGn")
```

```
[51]: <AxesSubplot: >
```



**Final Dataframe:** \* Now we'll join all of the One-hot and Label-encoded features to the original data frame to make the final data frame. We'll also get rid of the old variables that we used to create the new encoded variables.

```
[52]: df.head(3)
```

```
[52]:
```

	Airline	Source	Destination	Total_Stops	Price	journey_Date \
0	IndiGo	Banglore	New Delhi	0	3897	24
1	Air India	Kolkata	Banglore	2	7662	1
2	Jet Airways	Delhi	Cochin	2	13882	9

	journey_Month	Dep_hour	Dep_min	Arrival_hour	Arrival_min \
0	3	22	20	1	10
1	5	5	50	13	15
2	6	9	25	4	25

	Duration_hours	Duration_mins
0	2	50
1	7	25
2	19	0

```
[53]: df.isnull().sum()
```

```
[53]: Airline      0
      Source      0
      Destination  0
      Total_Stops  0
      Price      0
      journey_Date  0
      journey_Month  0
      Dep_hour     0
      Dep_min      0
      Arrival_hour  0
      Arrival_min   0
      Duration_hours  0
      Duration_mins  0
      dtype: int64
```

```
[54]: #Concatenate dataframe --> df+ Airline + Source + Destination
      data_train=pd.concat([df,Airline , Source, Destination],axis=1)
      # we have drop the variables
      data_train.drop(["Airline","Source","Destination"],axis=1,inplace=True)
      data_train.head()
```

```
[54]:   Total_Stops  Price  journey_Date  journey_Month  Dep_hour  Dep_min  \
0           0   3897           24           3         22        20
1           2   7662            1           5          5        50
2           2  13882            9           6          9        25
3           1   6218           12           5         18         5
4           1  13302            1           3         16        50

      Arrival_hour  Arrival_min  Duration_hours  Duration_mins  ...  \
0                1           10                2             50  ...
1               13           15                7             25  ...
2                4           25               19              0  ...
3               23           30                5             25  ...
4               21           35                4             45  ...

      Vistara Premium economy  Chennai  Delhi  Kolkata  Mumbai  \
0                          0         0      0         0         0
1                          0         0      0         1         0
2                          0         0      1         0         0
3                          0         0      0         1         0
4                          0         0      0         0         0

      Destination_Cochin  Destination_Delhi  Destination_Hyderabad  \
0                      0                 0                     0
1                      0                 0                     0
```

2	1	0	0
3	0	0	0
4	0	0	0

	Destination_Kolkata	Destination_New Delhi
0	0	1
1	0	0
2	0	0
3	0	0
4	0	1

[5 rows x 31 columns]

```
[55]: data_train.columns.unique()
```

```
[55]: Index(['Total_Stops', 'Price', 'journey_Date', 'journey_Month', 'Dep_hour',
        'Dep_min', 'Arrival_hour', 'Arrival_min', 'Duration_hours',
        'Duration_mins', 'Air Asia', 'Air India', 'GoAir', 'IndiGo',
        'Jet Airways', 'Jet Airways Business', 'Multiple carriers',
        'Multiple carriers Premium economy', 'SpiceJet', 'Trujet', 'Vistara',
        'Vistara Premium economy', 'Chennai', 'Delhi', 'Kolkata', 'Mumbai',
        'Destination_Cochin', 'Destination_Delhi', 'Destination_Hyderabad',
        'Destination_Kolkata', 'Destination_New Delhi'],
        dtype='object')
```

```
[56]: data_train.shape
```

```
[56]: (10682, 31)
```

```
[57]: data_train.isnull().sum()
```

```
[57]: Total_Stops          0
Price                  0
journey_Date          0
journey_Month         0
Dep_hour              0
Dep_min              0
Arrival_hour         0
Arrival_min          0
Duration_hours        0
Duration_mins         0
Air Asia              0
Air India             0
GoAir                 0
IndiGo                0
Jet Airways           0
Jet Airways Business  0
```

```

Multiple carriers          0
Multiple carriers Premium economy  0
SpiceJet                  0
Trujet                   0
Vistara                   0
Vistara Premium economy   0
Chennai                   0
Delhi                     0
Kolkata                   0
Mumbai                    0
Destination_Cochin        0
Destination_Delhi         0
Destination_Hyderabad     0
Destination_Kolkata       0
Destination_New Delhi     0
dtype: int64

```

```
[58]: data_train.shape
```

```
[58]: (10682, 31)
```

As a result, the final data frame has 32 variables, including the dependent variable 'Price.' For training, there are only 29 variables.

As a result, the final data frame has 30 variables, including the dependent variable 'Price.' For training, there are only 29 variables.

#### 4.1.1 Importing DataSet

```
[59]: test_data=pd.read_excel("Test_set.xlsx")
test_data.head()
```

```
[59]:
```

	Airline	Date_of_Journey	Source	Destination	Route \
0	Jet Airways	6/06/2019	Delhi	Cochin	DEL → BOM → COK
1	IndiGo	12/05/2019	Kolkata	Banglore	CCU → MAA → BLR
2	Jet Airways	21/05/2019	Delhi	Cochin	DEL → BOM → COK
3	Multiple carriers	21/05/2019	Delhi	Cochin	DEL → BOM → COK
4	Air Asia	24/06/2019	Banglore	Delhi	BLR → DEL

	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info
0	17:30	04:25 07 Jun	10h 55m	1 stop	No info
1	06:20	10:20	4h	1 stop	No info
2	19:15	19:00 22 May	23h 45m	1 stop	In-flight meal not included
3	08:00	21:00	13h	1 stop	No info
4	23:55	02:45 25 Jun	2h 50m	non-stop	No info

```
[60]: test_data.isnull().sum()
```



```
[60]: Airline          0
      Date_of_Journey  0
      Source          0
      Destination     0
      Route           0
      Dep_Time        0
      Arrival_Time    0
      Duration        0
      Total_Stops     0
      Additional_Info  0
      dtype: int64
```

Doing All above steps for Test Data

```
[61]: # Preprocessing

print(test_data.info())

test_data.dropna(inplace = True)
print(test_data.isnull().sum())

# EDA

# Date_of_Journey
test_data["Journey_day"] = pd.to_datetime(test_data.Date_of_Journey, format="%d/
    ↪%m/%Y").dt.day
test_data["Journey_month"] = pd.to_datetime(test_data["Date_of_Journey"],
    ↪format = "%d/%m/%Y").dt.month
test_data.drop(["Date_of_Journey"], axis = 1, inplace = True)

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

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

# Duration
duration = list(test_data["Duration"])

for i in range(len(duration)):
```

```

        if len(duration[i].split()) != 2:    # Check if duration contains only hour
        ↪ or mins
            if "h" in duration[i]:
                duration[i] = duration[i].strip() + " 0m"    # Adds 0 minute
            else:
                duration[i] = "0h " + duration[i]            # Adds 0 hour

duration_hours = []
duration_mins = []
for i in range(len(duration)):
    duration_hours.append(int(duration[i].split(sep = "h")[0]))    # Extract
    ↪ hours from duration
    duration_mins.append(int(duration[i].split(sep = "m")[0].split()[-1]))    #
    ↪ Extracts only minutes from duration

# Adding Duration column to test set
test_data["Duration_hours"] = duration_hours
test_data["Duration_mins"] = duration_mins
test_data.drop(["Duration"], axis = 1, inplace = True)

# Categorical data

print("Airline")
print("-"*75)
print(test_data["Airline"].value_counts())
Airline = pd.get_dummies(test_data["Airline"], drop_first= True)

print(test_data["Source"].value_counts())
Source = pd.get_dummies(test_data["Source"], drop_first= True)

print(test_data["Destination"].value_counts())
Destination = pd.get_dummies(test_data["Destination"], drop_first = True)

# Additional_Info contains almost 80% no_info
# Route and Total_Stops are related to each other
test_data.drop(["Route", "Additional_Info"], axis = 1, inplace = True)

# Replacing Total_Stops
test_data.replace({"non-stop": 0, "1 stop": 1, "2 stops": 2, "3 stops": 3, "4
    ↪ stops": 4}, inplace = True)

```

```

# Concatenate dataframe --> test_data + Airline + Source + Destination
data_test = pd.concat([test_data, Airline, Source, Destination], axis = 1)

data_test.drop(["Airline", "Source", "Destination"], axis = 1, inplace = True)

print()
print()

print("Shape of test data : ", data_test.shape)

```

```
<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
```

```
None
```

Airline	0
Date_of_Journey	0
Source	0
Destination	0
Route	0
Dep_Time	0
Arrival_Time	0
Duration	0
Total_Stops	0
Additional_Info	0

```
dtype: int64
```

```
Airline
```

Jet Airways	897
IndiGo	511
Air India	440
Multiple carriers	347
SpiceJet	208
Vistara	129

```

Air Asia      86
GoAir         46
Multiple carriers Premium economy    3
Vistara Premium economy              2
Jet Airways Business                 2
Name: Airline, dtype: int64
Delhi      1145
Kolkata     710
Bangalore   555
Mumbai     186
Chennai     75
Name: Source, dtype: int64
Cochin     1145
Bangalore   710
Delhi       317
New Delhi   238
Hyderabad   186
Kolkata     75
Name: Destination, dtype: int64

```

Shape of test data : (2671, 28)

## 5 4. Building Machine Learning Model using Random Forest Algorithm

```
[62]: data_train.columns.unique()
```

```
[62]: Index(['Total_Stops', 'Price', 'journey_Date', 'journey_Month', 'Dep_hour',
            'Dep_min', 'Arrival_hour', 'Arrival_min', 'Duration_hours',
            'Duration_mins', 'Air Asia', 'Air India', 'GoAir', 'IndiGo',
            'Jet Airways', 'Jet Airways Business', 'Multiple carriers',
            'Multiple carriers Premium economy', 'SpiceJet', 'Trujet', 'Vistara',
            'Vistara Premium economy', 'Chennai', 'Delhi', 'Kolkata', 'Mumbai',
            'Destination_Cochin', 'Destination_Delhi', 'Destination_Hyderabad',
            'Destination_Kolkata', 'Destination_New Delhi'],
            dtype='object')
```

```
[63]: x= data_train[['Total_Stops', 'journey_Date', 'journey_Month', 'Dep_hour',
                    'Dep_min', 'Arrival_hour', 'Arrival_min', 'Duration_hours',
                    'Duration_mins', 'Air India', 'GoAir', 'IndiGo', 'Jet Airways',
                    'Jet Airways Business', 'Multiple carriers',
                    'Multiple carriers Premium economy', 'SpiceJet', 'Trujet', 'Vistara',
                    'Vistara Premium economy', 'Chennai', 'Delhi', 'Kolkata', 'Mumbai',
                    'Destination_Cochin', 'Destination_Delhi', 'Destination_Hyderabad',
                    'Destination_Kolkata', 'Destination_New Delhi']]
```

```
x.head()
```

```
[63]:
```

	Total_Stops	journey_Date	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	Duration_hours	Duration_mins	Air India	...	\
0	10	2	50	0	...	
1	15	7	25	1	...	
2	25	19	0	0	...	
3	30	5	25	0	...	
4	35	4	45	0	...	

	Vistara Premium economy	Chennai	Delhi	Kolkata	Mumbai	\
0		0	0	0	0	
1		0	0	1	0	
2		0	0	1	0	
3		0	0	1	0	
4		0	0	0	0	

	Destination_Cochin	Destination_Delhi	Destination_Hyderabad	\
0	0	0	0	
1	0	0	0	
2	1	0	0	
3	0	0	0	
4	0	0	0	

	Destination_Kolkata	Destination_New Delhi
0	0	1
1	0	0
2	0	0
3	0	0
4	0	1

[5 rows x 29 columns]

```
[64]: x.head()
```

```
[64]:
```

	Total_Stops	journey_Date	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	Duration_hours	Duration_mins	Air India	...	\
0	10	2	50	0	...	
1	15	7	25	1	...	
2	25	19	0	0	...	
3	30	5	25	0	...	
4	35	4	45	0	...	

	Vistara Premium economy	Chennai	Delhi	Kolkata	Mumbai	\
0	0	0	0	0	0	
1	0	0	0	1	0	
2	0	0	1	0	0	
3	0	0	0	1	0	
4	0	0	0	0	0	

	Destination_Cochin	Destination_Delhi	Destination_Hyderabad	\
0	0	0	0	
1	0	0	0	
2	1	0	0	
3	0	0	0	
4	0	0	0	

	Destination_Kolkata	Destination_New Delhi
0	0	1
1	0	0
2	0	0
3	0	0
4	0	1

[5 rows x 29 columns]

```
[65]: x.isnull().sum()
```

```
[65]: Total_Stops          0
      journey_Date         0
      journey_Month        0
      Dep_hour             0
      Dep_min              0
      Arrival_hour         0
      Arrival_min          0
      Duration_hours       0
      Duration_mins        0
      Air India            0
      GoAir                0
      IndiGo               0
      Jet Airways          0
      Jet Airways Business 0
```

Multiple carriers	0
Multiple carriers Premium economy	0
SpiceJet	0
Trujet	0
Vistara	0
Vistara Premium economy	0
Chennai	0
Delhi	0
Kolkata	0
Mumbai	0
Destination_Cochin	0
Destination_Delhi	0
Destination_Hyderabad	0
Destination_Kolkata	0
Destination_New Delhi	0
dtype:	int64

```
[66]: # dependent Feature
y=data_train['Price']
```

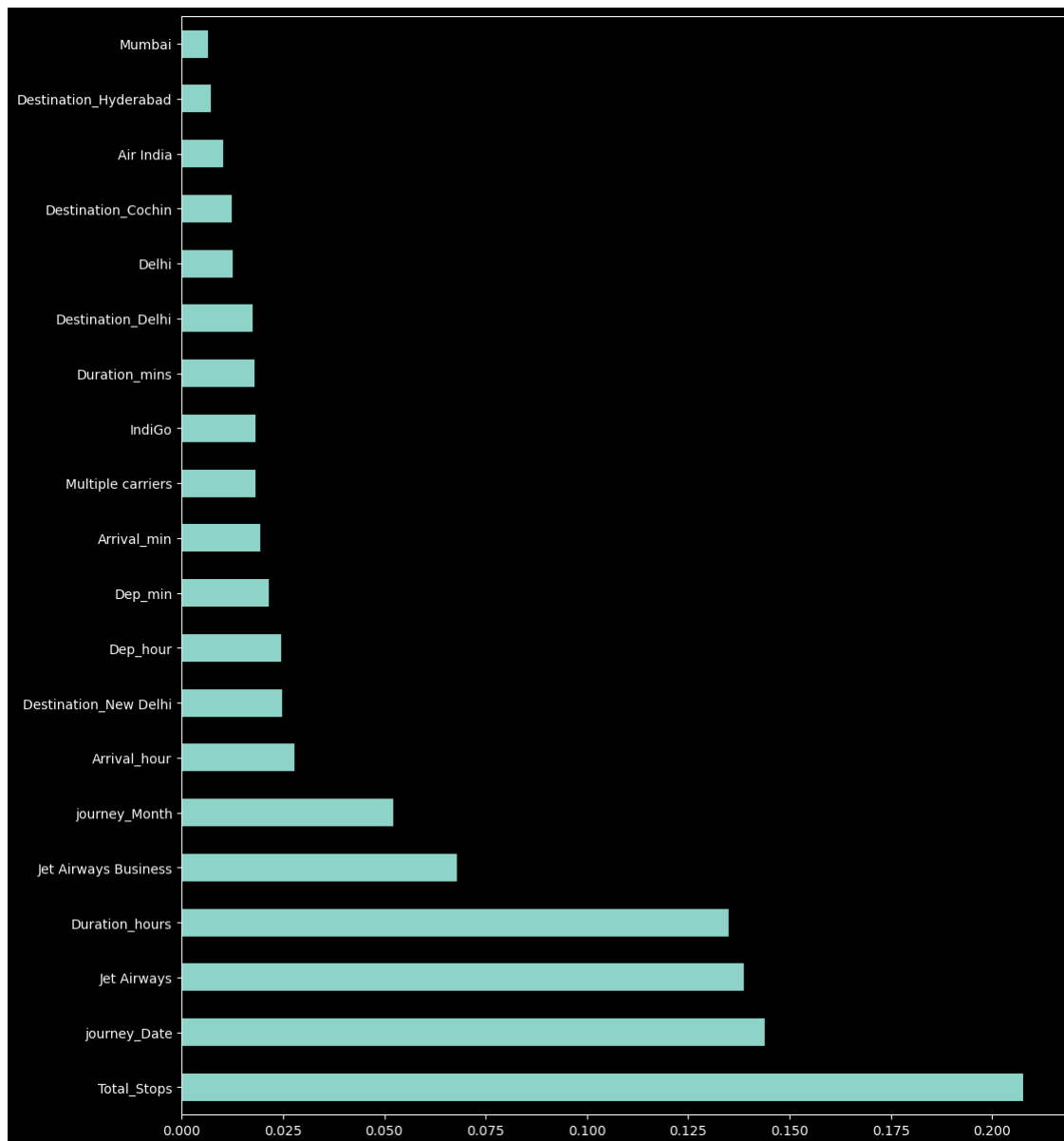
### 5.0.1 Feature Importance

- In Machine Learning, The purpose of feature selection is to discover the best set of characteristic that allows one to develop usable models of the phenomena being examined

```
[67]: # Feature selection
from sklearn.ensemble import ExtraTreesRegressor
selection=ExtraTreesRegressor()
selection.fit(x,y) # train model
```

```
[67]: ExtraTreesRegressor()
```

```
[68]: # plot graph of important features
plt.figure(figsize=(12,15))
feat_importances=pd.Series(selection.feature_importances_,index=x.columns)
feat_importances.nlargest(20).plot(kind='barh')
plt.style.use('dark_background')
plt.rcParams.update({'text.color':'white'})
plt.show()
```

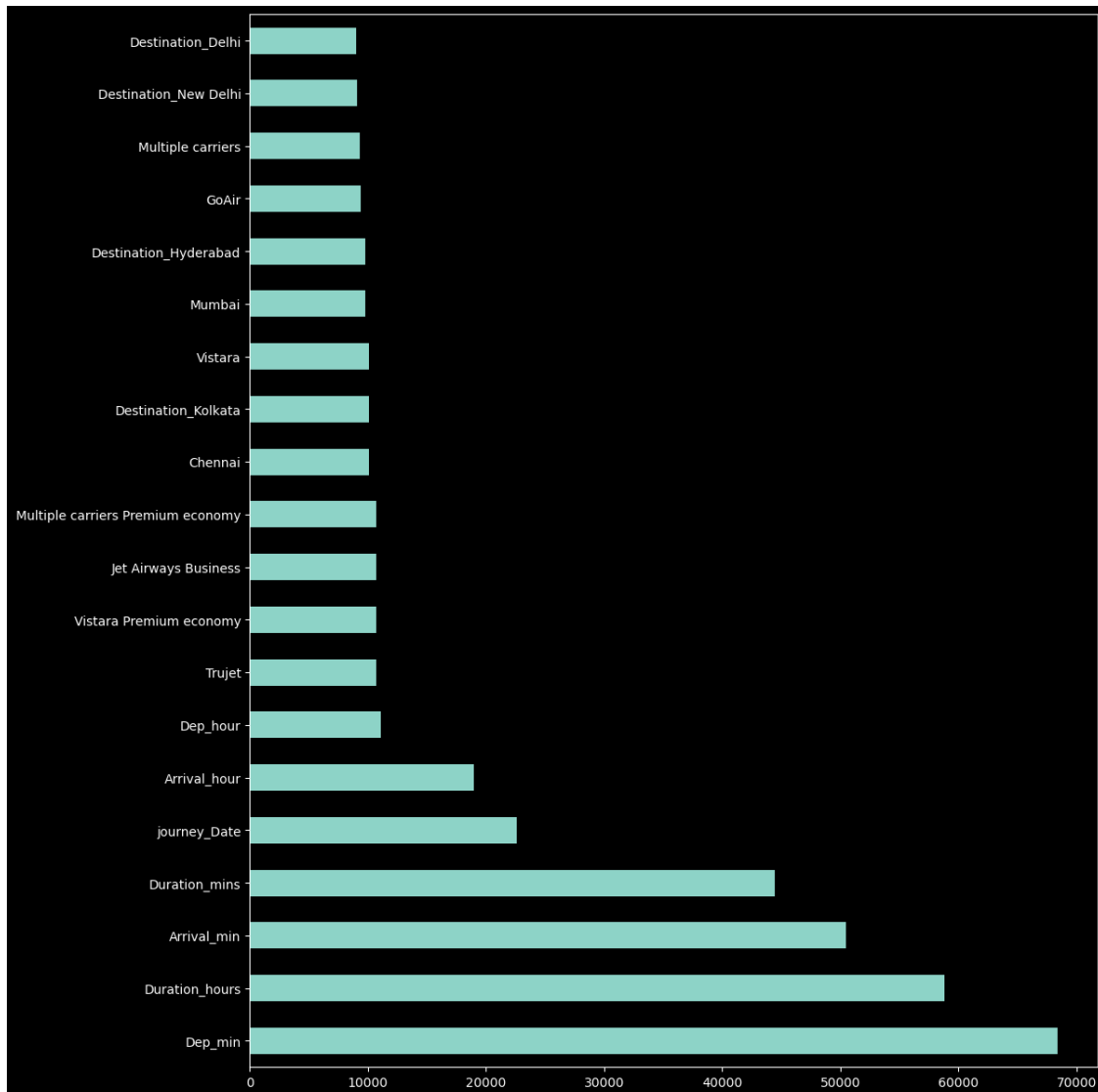


```
[69]: # import ilbrarry for feature selction
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
fs=SelectKBest(score_func=chi2)
X_slected=fs.fit(x,y)
```

```
[70]: # plot graph of important features
plt.figure(figsize=(12,15))
feat_importances=pd.Series(X_slected.scores_,index=x.columns)
feat_importances.nlargest(20).plot(kind='barh')
plt.style.use('dark_background')
```



```
plt.rcParams.update({'text.color': 'white'})
plt.show()
```



```
[71]: # splitting data in train and test data
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.
↪25,random_state=42)
```

```
[72]: from sklearn.ensemble import RandomForestRegressor
random_forest=RandomForestRegressor()
```

```
[73]: random_forest.fit(x_train,y_train)
```

```
[73]: RandomForestRegressor()
```

### R2 SCORE

```
[74]: random_forest.score(x_test,y_test)
```

```
[74]: 0.7980742619212899
```

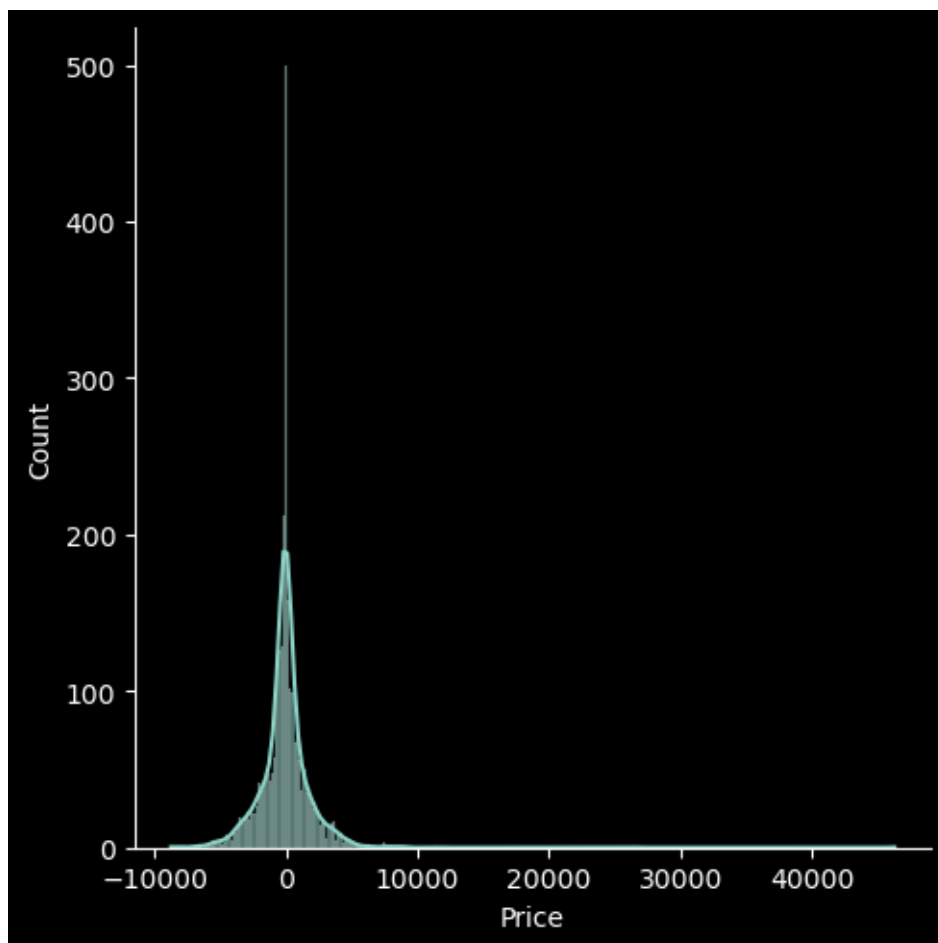
```
[75]: random_forest.score(x_train,y_train)
```

```
[75]: 0.9540259412679606
```

```
[76]: y_pred=random_forest.predict(x_test)  
y_pred
```

```
[76]: array([16908.39,  5558.54,  8843.7 , ...,  3528.19,  6333.17,  6877.61])
```

```
[77]: # Ploting The error graph and should mean is zero  
sns.displot(y_test-y_pred,kde=True)  
plt.style.use('dark_background')  
plt.rcParams.update({'text.color':'white'})
```

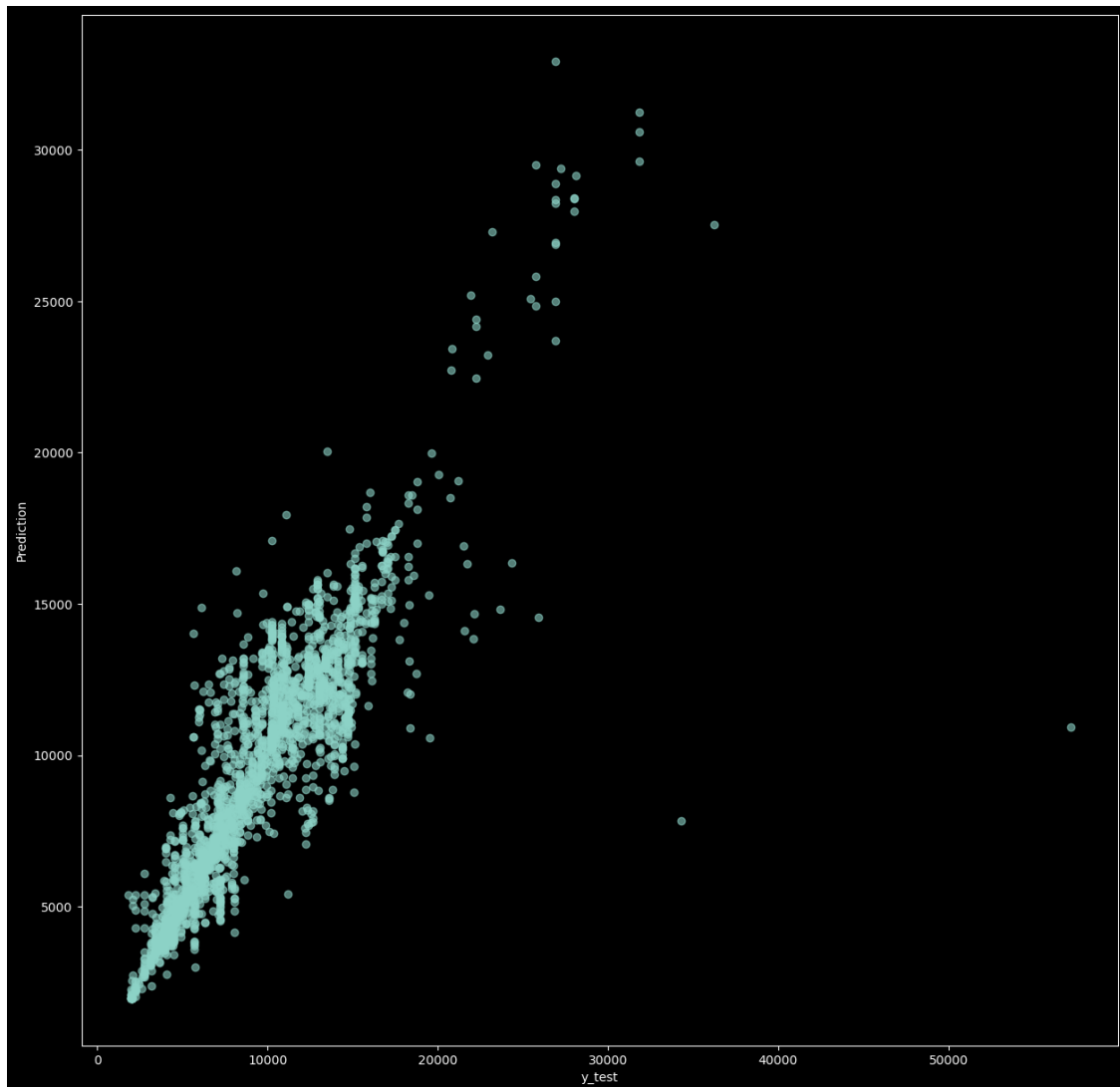


With an R2 score of 81 percent, With this model, we can also calculate the minimal values for mean absolute error, mean squared error, and root mean squared error (regression metrics). We will try to improve the accuracy by doing hyperparameter tuning.

```
[78]: from sklearn import metrics
print('MAE:', metrics.mean_absolute_error(y_test, y_pred))
print('MSE:', metrics.mean_squared_error(y_test, y_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
MAE: 1165.0942441090046
MSE: 4162300.5736071332
RMSE: 2040.1717019915586
```

```
[79]: # plotting graph for check linear regression
plt.figure(figsize=(15,15))
plt.scatter(y_test,y_pred,alpha=0.6)
plt.xlabel('y_test')
plt.ylabel('Prediction')
plt.style.use("dark_background")
plt.rcParams.update({'text.color':'white'})
```



Performing Hyperparameter Tuning for better Accuracy, it can be done using:-

- RandomizedSearchCV
- GridSearchCV
- Here We will be using RandomizedSearchCV

```
[80]: n_estimators = [int(x) for x in np.linspace(start = 100, stop = 1200, num = 12)]
max_features = ['auto', 'sqrt']
max_depth = [int(x) for x in np.linspace(5, 30, num = 8)]
min_samples_split = [2, 5, 10, 15, 100, 120, 150, 200, 250]
min_samples_leaf = [1, 2, 5, 10, 15, 25, 30, 35]
```

```
[81]: random_grid_params = {'n_estimators': n_estimators,
                           'max_features': max_features,
                           'max_depth': max_depth,
                           'min_samples_split': min_samples_split,
                           'min_samples_leaf': min_samples_leaf}
```

```
[82]: from sklearn.model_selection import RandomizedSearchCV, GridSearchCV, \
      ↪ train_test_split
```

We will pass these parameter into our random forest classifier

```
[83]: random_forest_regressor=RandomForestRegressor(n_estimators=300,
            min_samples_split= 10,
            min_samples_leaf= 2,
            max_features= 'auto',
            max_depth= 15)
```

```
[84]: random_forest_regressor.fit(x_train,y_train)
```

```
[84]: RandomForestRegressor(max_depth=15, max_features='auto', min_samples_leaf=2,
            min_samples_split=10, n_estimators=300)
```

```
[85]: random_forest_regressor.score(x_train,y_train)
```

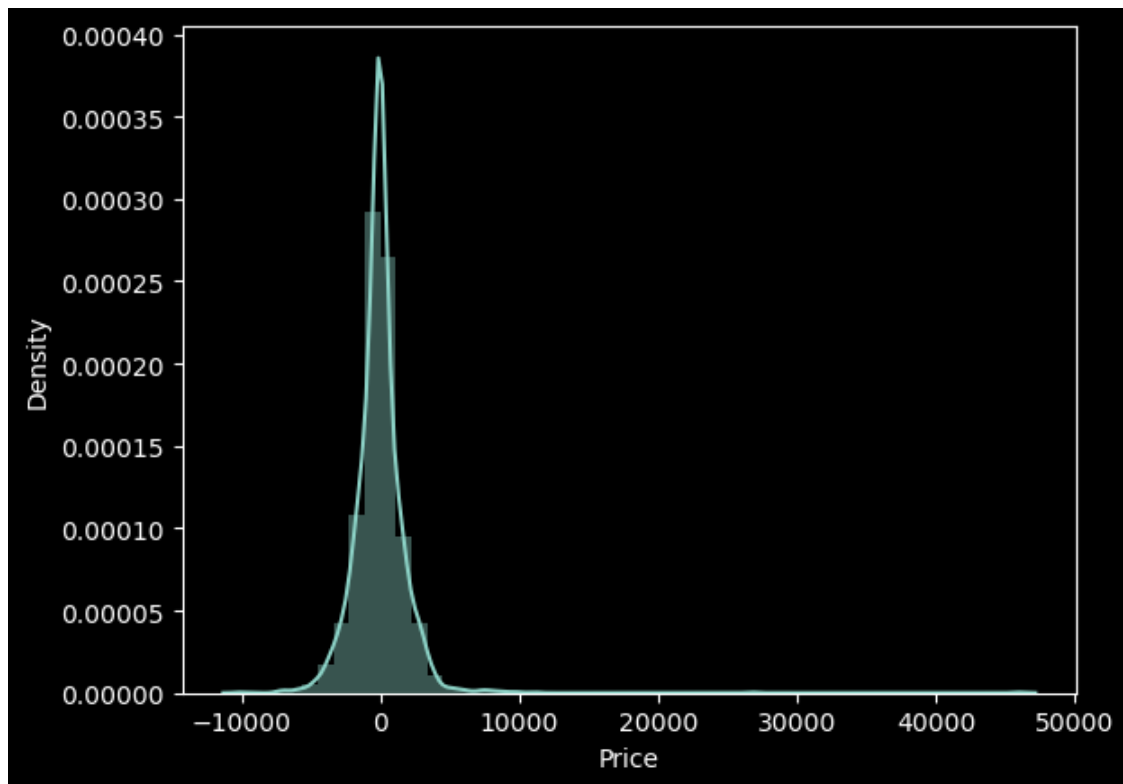
```
[85]: 0.9021654637090928
```

```
[86]: random_forest_regressor.score(x_test,y_test)
```

```
[86]: 0.8156563330644699
```

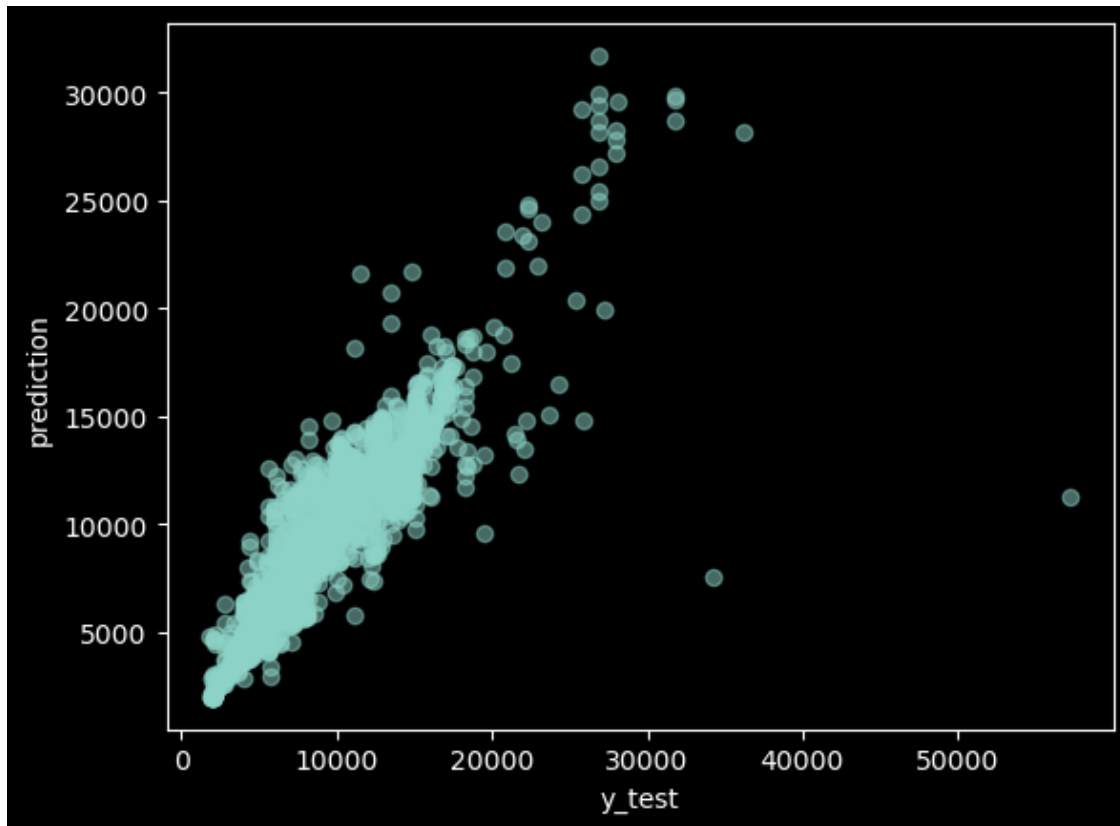
```
[87]: prediction=random_forest_regressor.predict(x_test)
```

```
[88]: #Plotting the error graph and should be mean=0
      sns.distplot(y_test-prediction,kde=True)
      plt.style.use("dark_background")
      plt.rcParams.update({'text.color': 'white'})
```



```
[89]: #Plotting scatter graph to check linear relations
plt.scatter(y_test,prediction,alpha=0.5)
plt.style.use("dark_background")
plt.rcParams.update({'text.color':'white'})
plt.xlabel('y_test')
plt.ylabel('prediction')
```

```
[89]: Text(0, 0.5, 'prediction')
```



```
[90]: from sklearn import metrics
print('MAE:', metrics.mean_absolute_error(y_test, prediction))
print('MSE:', metrics.mean_squared_error(y_test, prediction))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, prediction)))
```

MAE: 1142.2782907830003  
MSE: 3799880.8766394616  
RMSE: 1949.3283142250466

After hyper tuning, the R2 score for random forest Regressor is 84 percent, whereas, before hyper tuning, the R2 score for random forest Regressor was 81 percent. The value of MAE drops as well, indicating that we were successful in tuning our model.

**Conclusion:** \* So, we have used a random forest model for this data and improved accuracy by doing hyperparameter tuning. \* As a result, we were able to successfully train our regression model, the 'Random forest model,' to forecast fares of flight tickets with an R2 score of 84 percent and complete the required work.

## 5.0.2 Model Saving in Pickle Format

```
[92]: import pickle
      file=open('Flight_Price_Pridiction.pkl','wb')
      pickle.dump(random_forest_regresor,file)
```

## 5.0.3 Loading The Model Saved in Pickle Format

```
[93]: model=open('Flight_Price_Pridiction.pkl','rb')
      flight_fare_predictor=pickle.load(model)
```

# 6 Predicting Use Loaded Model

```
[94]: flight_fare_predictor.score(x_test,y_test)
```

```
[94]: 0.8156563330644699
```

```
[95]: x_test
```

```
[95]:      Total_Stops  journey_Date  journey_Month  Dep_hour  Dep_min  \
6075             2             21             5         15         5
3544             1              3             6         10        35
9291             1              9             5         20        20
5032             0             24             5         14        45
2483             1             21             5         22        50
...             ...             ...             ...         ...         ...
3022             1             18             3         16        55
8416             2              3             3          9        45
2605             0             21             6          0        25
1392             1             27             6         17        45
8535             1              3             6          5          5

      Arrival_hour  Arrival_min  Duration_hours  Duration_mins  Air India  \
6075             1           30             10           25         0
3544            19           35              9            0         0
9291             9            5             12           45         0
5032            17            5              2           20         0
2483             4           25              5           35         0
...             ...             ...             ...         ...         ...
3022            23            0              6            5         0
8416             9           25             23           40         1
2605             3           15              2           50         0
1392             1           30              7           45         0
8535            12           10              7            5         0

      ...  Vistara Premium economy  Chennai  Delhi  Kolkata  Mumbai  \
6075  ...                        0         0         1         0         0
```



3544	...	0	0	1	0	0
9291	...	0	0	0	1	0
5032	...	0	1	0	0	0
2483	...	0	0	1	0	0
...	...	...	...	...	...	...
3022	...	0	0	0	0	0
8416	...	0	0	1	0	0
2605	...	0	0	0	0	0
1392	...	0	0	1	0	0
8535	...	0	0	1	0	0

	Destination_Cochin	Destination_Delhi	Destination_Hyderabad	\
6075	1	0		0
3544	1	0		0
9291	0	0		0
5032	0	0		0
2483	1	0		0
...	...	...	...	...
3022	0	0		0
8416	1	0		0
2605	0	1		0
1392	1	0		0
8535	1	0		0

	Destination_Kolkata	Destination_New Delhi
6075	0	0
3544	0	0
9291	0	0
5032	1	0
2483	0	0
...	...	...
3022	0	1
8416	0	0
2605	0	0
1392	0	0
8535	0	0

[2671 rows x 29 columns]

```
[96]: y_prediction=flight_fare_predictor.predict(x_test)
      y_prediction
```

```
[96]: array([16559.86333548,  5509.48618671,  8691.03972869, ...,
            3870.10572319,  7054.6589413 ,  6784.51269835])
```

```
<h2 style='padding: 20px;
        color:red;
        text-align:center;'>
```

```
        END OF THE PROJECT !  
</h2>  
</div>
```

[ ]: