

## Flight Fare prediction

```
In [4]: ▶ import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

```
In [5]: ▶ # Reading the dataset
train_df = pd.read_csv("C:/Users/srink/Downloads/Data_Train.csv")
```

```
In [6]: ▶ #displaying the maximum columns
pd.set_option('display.max_columns', None)
```

```
In [7]: ▶ train_df.head()
```

Out[7]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stop
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR ? DEL	22:20	01:10 22 Mar	2h 50m	non-stop
1	Air India	1/05/2019	Kolkata	Banglore	CCU ? IXR ? BBI ? BLR	05:50	13:15	7h 25m	2 stop
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL ? LKO ? BOM ? COK	09:25	04:25 10 Jun	19h	2 stop
3	IndiGo	12/05/2019	Kolkata	Banglore	CCU ? NAG ? BLR	18:05	23:30	5h 25m	1 stop
4	IndiGo	01/03/2019	Banglore	New Delhi	BLR ? NAG ? DEL	16:50	21:35	4h 45m	1 stop



```
In [8]: ▶ train_df.shape
```

Out[8]: (10683, 11)

In [9]: `# Info the dataset`  
`train_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
```

In [10]: `# Description of the dataset`  
`train_df.describe().T`

Out[10]:

	count	mean	std	min	25%	50%	75%	max
Price	10683.0	9087.064121	4611.359167	1759.0	5277.0	8372.0	12373.0	79512.0

## Data Preprocessing

### Missing values

In [11]: `#Finding the null values of the dataset`  
`train_df.isnull().sum()`

```
Out[11]: 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
```

In [12]: `train_df.dropna(inplace=True)`

```
In [13]: train_df.isnull().sum()
```

```
Out[13]: 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
```

Removed the null values in the dataset

### Creating new columns

Creating the new columns in the dataframe to make clear values ,which will be used for analysis

```
In [14]: #Creating the new column journey day
train_df['Journey_Day'] = pd.to_datetime(train_df.Date_of_Journey, format='%d/%m/%Y')
```

```
In [15]: #Creating the new column journey month
train_df['Journey_Month'] = pd.to_datetime(train_df.Date_of_Journey, format='%d/%m/%Y')
```

```
In [16]: train_df.head()
```

```
Out[16]:
```

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR ? DEL	22:20	01:10 22 Mar	2h 50m	non-stop
1	Air India	1/05/2019	Kolkata	Banglore	CCU ? IXR ? BBI ? BLR	05:50	13:15	7h 25m	2 stop
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL ? LKO ? BOM ? COK	09:25	04:25 10 Jun	19h	2 stop
3	IndiGo	12/05/2019	Kolkata	Banglore	CCU ? NAG ? BLR	18:05	23:30	5h 25m	1 stop
4	IndiGo	01/03/2019	Banglore	New Delhi	BLR ? NAG ? DEL	16:50	21:35	4h 45m	1 stop

```
In [17]: #Dropping the date of journey column
train_df.drop(['Date_of_Journey'], axis=1, inplace=True)
```

```
In [18]: train_df.head()
```

Out[18]:

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

```
In [19]: train_df.head()
```

Out[19]:

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

```
In [20]: print(train_df.columns.tolist())
```

['Airline', 'Source', 'Destination', 'Route', 'Dep\_Time', 'Arrival\_Time', 'Duration', 'Total\_Stops', 'Additional\_Info', 'Price', 'Journey\_Day', 'Journey\_Month']

```
In [21]: train_df['Dep_hour'] = pd.to_datetime(train_df['Dep_Time'], format='%H:%M').dt.hour
train_df['Dep_min'] = pd.to_datetime(train_df['Dep_Time'], format='%H:%M').dt.minute

train_df.drop(['Dep_Time'], axis=1, inplace=True)
```

```
In [22]: train_df['Arrival_hour'] = pd.to_datetime(train_df.Arrival_Time).dt.hour
train_df['Arrival_min'] = pd.to_datetime(train_df.Arrival_Time).dt.minute

train_df.drop(['Arrival_Time'], axis=1, inplace=True)
```

```
In [23]: train_df.head()
```

Out[23]:

	Airline	Source	Destination	Route	Duration	Total_Stops	Additional_Info	Price	Journey_Day	Jou
0	IndiGo	Banglore	New Delhi	BLR ? DEL	2h 50m	non-stop	No info	3897	24	
1	Air India	Kolkata	Banglore	CCU ? IXR ? BBI ? BLR	7h 25m	2 stops	No info	7662	1	
2	Jet Airways	Delhi	Cochin	DEL ? LKO ? BOM ? COK	19h	2 stops	No info	13882	9	
3	IndiGo	Kolkata	Banglore	CCU ? NAG ? BLR	5h 25m	1 stop	No info	6218	12	
4	IndiGo	Banglore	New Delhi	BLR ? NAG ? DEL	4h 45m	1 stop	No info	13302	1	

```
In [144]: ▶ # Check the column names in train_df
print(train_df.columns)

# Assuming the column containing duration information is named "Duration" or similar
# Replace "Duration" with the actual column name if it's different
duration_column_name = "Duration"

# Assigning and converting the duration column into a list
if duration_column_name in train_df.columns:
    duration = list(train_df[duration_column_name])

    for i in range(len(duration)):
        # Check if duration contains only hour or minutes
        if len(duration[i].split()) != 2:
            if 'h' in duration[i]:
                # Adding 0 mins
                duration[i] = duration[i].strip() + " 0m"
            else:
                duration[i] = "0h " + duration[i]

    duration_hours = []
    duration_mins = []
    for i in range(len(duration)):
        # Extract hours from Duration
        duration_hours.append(int(duration[i].split("h")[0]))
        # Extract only minutes from Duration
        duration_mins.append(int(duration[i].split('m')[0].split()[-1]))
    else:
        print("Column '{}' not found in DataFrame.".format(duration_column_name))
```

```
Index(['Airline', 'Source', 'Destination', 'Total_Stops', 'Price',
       'Journey_Day', 'Journey_Month', 'Dep_hour', 'Dep_min', 'Arrival_hour',
       'Arrival_min', 'Duration_hours', 'Duration_mins'],
      dtype='object')
Column 'Duration' not found in DataFrame.
```

```
In [25]: ▶ ##Creating the columns duration hours and duration minutes from the duration column
train_df["Duration_hours"] = duration_hours
train_df["Duration_mins"] = duration_mins
train_df.drop(['Duration'], axis=1, inplace=True)
```

```
In [26]: train_df.head()
```

Out[26]:

	Airline	Source	Destination	Route	Total_Stops	Additional_Info	Price	Journey_Day	Journey_Month
0	IndiGo	Banglore	New Delhi	BLR ? DEL	non-stop	No info	3897	24	3
1	Air India	Kolkata	Banglore	CCU ? IXR ? BBI ? BLR	2 stops	No info	7662	1	5
2	Jet Airways	Delhi	Cochin	DEL ? LKO ? BOM ? COK	2 stops	No info	13882	9	6
3	IndiGo	Kolkata	Banglore	CCU ? NAG ? BLR	1 stop	No info	6218	12	5
4	IndiGo	Banglore	New Delhi	BLR ? NAG ? DEL	1 stop	No info	13302	1	3

Handling Categorical Data

The categorical data can't be used for analysis. So this data is converted into some numerical values so that it is used for analysis

The Categorical columns that need to be changed is : Airline, Source, Destination

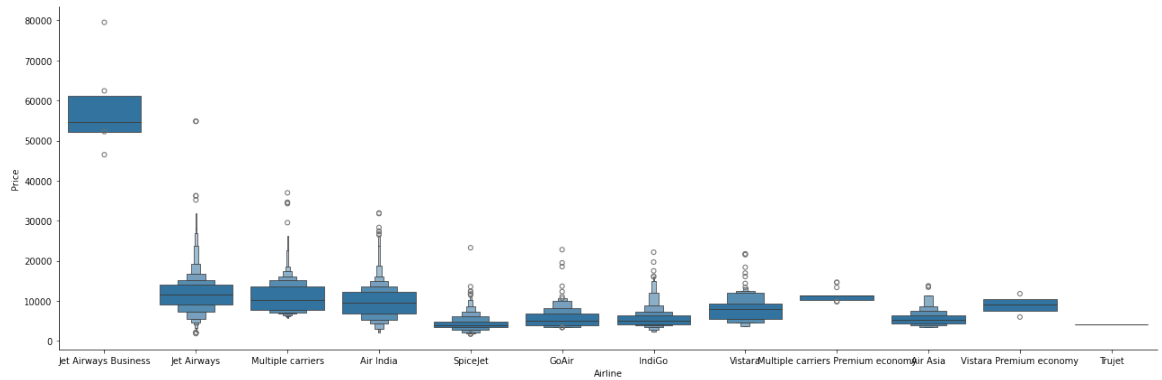
```
In [27]: train_df.Airline.value_counts()
```

Out[27]:

Airline	
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
Vistara Premium economy	3
Trujet	1
Name: count, dtype: int64	

```
In [28]: sns.catplot(y="Price", x="Airline", data=train_df.sort_values("Price", ascending=False))
```

```
Out[28]: <seaborn.axisgrid.FacetGrid at 0x1b8a7df4340>
```



The Airline is plotted according to their price

The most number of bookings are done for Jet Airways. The price for the jet airways business is the highest.

```
In [29]: len(train_df['Airline'].unique())
```

```
Out[29]: 12
```

```
In [30]: Airline = train_df[['Airline']]
Airline = pd.get_dummies(Airline, drop_first=True)
Airline.head()
```

```
Out[30]:
```

	Airline_Air India	Airline_GoAir	Airline_IndiGo	Airline_Jet Airways	Airline_Jet Airways Business	Airline_Multiple carriers	Airline_Multiple carriers Premium economy	Airl
0	False	False	True	False	False	False	False	
1	True	False	False	False	False	False	False	
2	False	False	False	True	False	False	False	
3	False	False	True	False	False	False	False	
4	False	False	True	False	False	False	False	

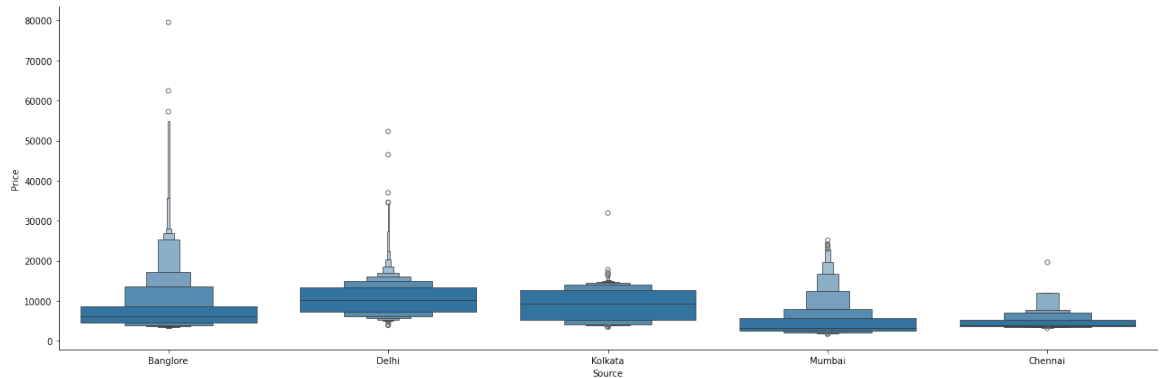
```
In [31]: train_df.Source.value_counts()
```

```
Out[31]: Source
Delhi      4536
Kolkata    2871
Bangalore  2197
Mumbai     697
Chennai    381
Name: count, dtype: int64
```



```
In [32]: sns.catplot(y='Price', x='Source', data=train_df.sort_values('Price', ascending=False))
```

```
Out[32]: <seaborn.axisgrid.FacetGrid at 0x1b8aa110610>
```



The most number of bookings are done from Delhi

```
In [33]: Source = train_df[['Source']]
Source = pd.get_dummies(Source, drop_first=True)
Source.head()
```

```
Out[33]:
```

	Source_Chennai	Source_Delhi	Source_Kolkata	Source_Mumbai
0	False	False	False	False
1	False	False	True	False
2	False	True	False	False
3	False	False	True	False
4	False	False	False	False

```
In [34]: Destination = train_df[['Destination']]
Destination = pd.get_dummies(Destination, drop_first=True)
Destination.head()
```

```
Out[34]:
```

	Destination_Cochin	Destination_Delhi	Destination_Hyderabad	Destination_Kolkata	Destination_New Delhi
0	False	False	False	False	True
1	False	False	False	False	False
2	True	False	False	False	False
3	False	False	False	False	False
4	False	False	False	False	True

```
In [35]: train_df.Route
```

```
Out[35]: 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: 10682, dtype: object
```

The column is route is already defined in the source, destination and total route columns. So we can drop the route column

In [141]: `print(train_df.columns)`

```
Index(['Airline', 'Source', 'Destination', 'Total_Stops', 'Price',
      'Journey_Day', 'Journey_Month', 'Dep_hour', 'Dep_min', 'Arrival_hour',
      'Arrival_min', 'Duration_hours', 'Duration_mins'],
      dtype='object')
```

In [37]: `a.mean()`

Out[37]: 0.781127129750983

As the Additional info is not useful for the analysis as it contains 78% of "No info" values. So we will drop that column.

In [38]: `## Additional_Info conatins almost 80% no_info`  
`# Route and Total_Stops are related to each other`

```
train_df.drop(['Route', 'Additional_Info'], axis=1, inplace=True)
```

In [39]: `train_df.head()`

Out[39]:

	Airline	Source	Destination	Total_Stops	Price	Journey_Day	Journey_Month	Dep_hour	Dep_min
0	IndiGo	Banglore	New Delhi	non-stop	3897	24	3	22	20
1	Air India	Kolkata	Banglore	2 stops	7662	1	5	5	50
2	Jet Airways	Delhi	Cochin	2 stops	13882	9	6	9	25
3	IndiGo	Kolkata	Banglore	1 stop	6218	12	5	18	5
4	IndiGo	Banglore	New Delhi	1 stop	13302	1	3	16	50

In [40]: `train_df.Total_Stops.value_counts()`

```
Out[40]: Total_Stops
1 stop      5625
non-stop    3491
2 stops     1520
3 stops       45
4 stops        1
Name: count, dtype: int64
```

In [41]: `## As this is case of Ordinal Categorical type we perform LabelEncoder`  
`## Here vlaues are assigned with corresponding keys`  
`train_df.replace({"non-stop": 0, "1 stop": 1, "2 stops": 2, "3 stops": 3, "4 stops":`

In [42]: `data_train = pd.concat([train_df, Airline, Source, Destination], axis=1)`

In [43]: `data_train.head()`

Out[43]:

	Airline	Source	Destination	Total_Stops	Price	Journey_Day	Journey_Month	Dep_hour	Dep_min
0	IndiGo	Banglore	New Delhi	0	3897	24	3	22	20
1	Air India	Kolkata	Banglore	2	7662	1	5	5	50
2	Jet Airways	Delhi	Cochin	2	13882	9	6	9	25
3	IndiGo	Kolkata	Banglore	1	6218	12	5	18	5
4	IndiGo	Banglore	New Delhi	1	13302	1	3	16	50

In [44]: `data_train.drop(['Destination', 'Source', 'Airline'], axis=1, inplace=True)`

In [45]: `data_train.head()`

Out[45]:

	Total_Stops	Price	Journey_Day	Journey_Month	Dep_hour	Dep_min	Arrival_hour	Arrival_min	Durat
0	0	3897	24	3	22	20	1	10	
1	2	7662	1	5	5	50	13	15	
2	2	13882	9	6	9	25	4	25	
3	1	6218	12	5	18	5	23	30	
4	1	13302	1	3	16	50	21	35	

In [46]: `data_train.shape`

Out[46]: (10682, 30)

## Test Set

we will make all these in the test set as well

In [47]: `test_data = pd.read_csv("C:/Users/srirk/Downloads/Test_set.csv")`

```
In [48]: test_data.head()
```

Out[48]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops
0	Jet Airways	6/06/2019	Delhi	Cochin	DEL ? BOM ? COK	17:30	04:25 07 Jun	10h 55m	1 stop
1	IndiGo	12/05/2019	Kolkata	Banglore	CCU ? MAA ? BLR	06:20	10:20	4h	1 stop
2	Jet Airways	21/05/2019	Delhi	Cochin	DEL ? BOM ? COK	19:15	19:00 22 May	23h 45m	1 stop
3	Multiple carriers	21/05/2019	Delhi	Cochin	DEL ? BOM ? COK	08:00	21:00	13h	1 stop
4	Air Asia	24/06/2019	Banglore	Delhi	BLR ? DEL	23:55	02:45 25 Jun	2h 50m	non-stop

```
In [49]: test_data.shape
```

Out[49]: (2671, 10)

```
In [50]: print(test_data.columns)
```

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



In [51]:

```

print("Test data Info")
print("-"*75)
print(test_data.info())

print()
print()

print("Null values :")
print("-"*75)
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 minute
        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]))    # Extract 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()

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

print()

```

```
print("Destination")
print("-"*75)
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)
```

## Test data Info

```

-----
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2671 entries, 0 to 2670
Data columns (total 10 columns):
#   Column                Non-Null Count  Dtype
---  -
0   Airline                2671 non-null  object
1   Date_of_Journey       2671 non-null  object
2   Source                 2671 non-null  object
3   Destination            2671 non-null  object
4   Route                 2671 non-null  object
5   Dep_Time               2671 non-null  object
6   Arrival_Time          2671 non-null  object
7   Duration               2671 non-null  object
8   Total_Stops            2671 non-null  object
9   Additional_Info        2671 non-null  object
dtypes: object(10)
memory usage: 208.8+ KB
None

```

## Null values :

```

-----
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

```

```

-----
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: count, dtype: int64

```

## Source

```

-----
Source
Delhi          1145
Kolkata        710
Bangalore      555
Mumbai         186
Chennai        75
Name: count, dtype: int64

```

## Destination

```

-----
Destination
Cochin          1145
Bangalore       710
Delhi           317
New Delhi       238

```





## Model Selection

In [54]: `data_train.head()`

Out[54]:

	Total_Stops	Price	Journey_Day	Journey_Month	Dep_hour	Dep_min	Arrival_hour	Arrival_min	Durat
0	0	3897	24	3	22	20	1	10	
1	2	7662	1	5	5	50	13	15	
2	2	13882	9	6	9	25	4	25	
3	1	6218	12	5	18	5	23	30	
4	1	13302	1	3	16	50	21	35	

In [55]: `data_train.columns`

Out[55]: Index(['Total\_Stops', 'Price', 'Journey\_Day', 'Journey\_Month', 'Dep\_hour', 'Dep\_min', 'Arrival\_hour', 'Arrival\_min', 'Duration\_hours', 'Duration\_mins', 'Airline\_Air India', 'Airline\_GoAir', 'Airline\_IndiGo', 'Airline\_Jet Airways', 'Airline\_Jet Airways Business', 'Airline\_Multiple carriers', 'Airline\_Multiple carriers Premium economy', 'Airline\_SpiceJet', 'Airline\_Trujet', 'Airline\_Vistara', 'Airline\_Vistara Premium economy', 'Source\_Chennai', 'Source\_Delhi', 'Source\_Kolkata', 'Source\_Mumbai', 'Destination\_Cochin', 'Destination\_Delhi', 'Destination\_Hyderabad', 'Destination\_Kolkata', 'Destination\_New Delhi'], dtype='object')

In [56]: `data_train.loc[:, ['Total_Stops', 'Journey_Day', 'Journey_Month', 'Dep_hour', 'Dep_min', 'Duration_hours', 'Duration_mins', 'Airline_Air India', 'Airline_GoAir', 'Airline_Jet Airways', 'Airline_Jet Airways Business', 'Airline_Multiple carriers', 'Airline_Multiple carriers Premium economy', 'Airline_SpiceJet', 'Airline_Trujet', 'Airline_Vistara', 'Airline_Vistara Premium economy', 'Source_Chennai', 'Source_Delhi', 'Source_Kolkata', 'Source_Mumbai', 'Destination_Cochin', 'Destination_Delhi', 'Destination_Hyderabad', 'Destination_Kolkata', 'Destination_New Delhi']]`

In [57]: `X.head()`

Out[57]:

	Total_Stops	Journey_Day	Journey_Month	Dep_hour	Dep_min	Arrival_hour	Duration_hours	Duration_mins
0	0	24	3	22	20	1	2	
1	2	1	5	5	50	13	7	
2	2	9	6	9	25	4	19	
3	1	12	5	18	5	23	5	
4	1	1	3	16	50	21	4	

```
In [58]: ▶ y = data_train.iloc[:, 1]
y.head()
```

```
Out[58]: 0      3897
1      7662
2     13882
3      6218
4     13302
Name: Price, dtype: int64
```

```
In [123]: ▶ from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from sklearn import metrics
```

```
In [124]: ▶ 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_stat
```

```
In [135]: ▶ from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import numpy as np

# Define the function for root mean squared error
def root_mean_squared_error(y_true, y_pred):
    return np.sqrt(mean_squared_error(y_true, y_pred))

# Assuming you have training data 'X_train' and corresponding labels 'y_train'
# Train your linear regression model
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)

# Assuming you have test data 'X_test' and true labels 'y_test'
# Make predictions
lr_preds = lr_model.predict(X_test)

# Calculate RMSE using the function
lr_rmse = root_mean_squared_error(y_test, lr_preds)

# Print the RMSE
print("Linear Regression RMSE:", lr_rmse)
```

Linear Regression RMSE: 2863.574262459376

```
In [137]: > from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
import numpy as np

# Assuming you have your training data X_train and corresponding target values y_train

# Train the decision tree model
dt_model = DecisionTreeRegressor()
dt_model.fit(X_train, y_train)

# Make predictions on the test data
dt_preds = dt_model.predict(X_test)

# Calculate RMSE using root_mean_squared_error function
dt_rmse = np.sqrt(mean_squared_error(y_test, dt_preds))
print("Decision Tree RMSE:", dt_rmse)
```

Decision Tree RMSE: 2419.780061836497

```
In [138]: > from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
import numpy as np

# Assuming you have trained your Random Forest model and obtained predictions
# rf_model = ... # your trained Random Forest model
# X_test = ... # your test feature data
# y_test = ... # your test target data
rf_preds = rf_model.predict(X_test) # Generate predictions using the model

# Calculate RMSE using root_mean_squared_error function
rf_rmse = np.sqrt(mean_squared_error(y_test, rf_preds))
print("Random Forest RMSE:", rf_rmse)
```

Random Forest RMSE: 2091.848833989654

The Random Forest has the less mean square error in it. So the model for the dataset is RandomForest

## Model Evaluation

```
In [88]: > from sklearn.ensemble import RandomForestRegressor
```

```
In [91]: > rf_model = RandomForestRegressor()
rf_model.fit(X_train, y_train)
```

Out[91]:  <https://scikit-learn.org/1.4/modules/generated/sklearn.ensemble.RandomForestRegressor.html>

```
In [92]: > y_pred = rf_model.predict(X_test)
```

```
In [93]: > rf_model.score(X_train, y_train)
```

Out[93]: 0.9534776084567111

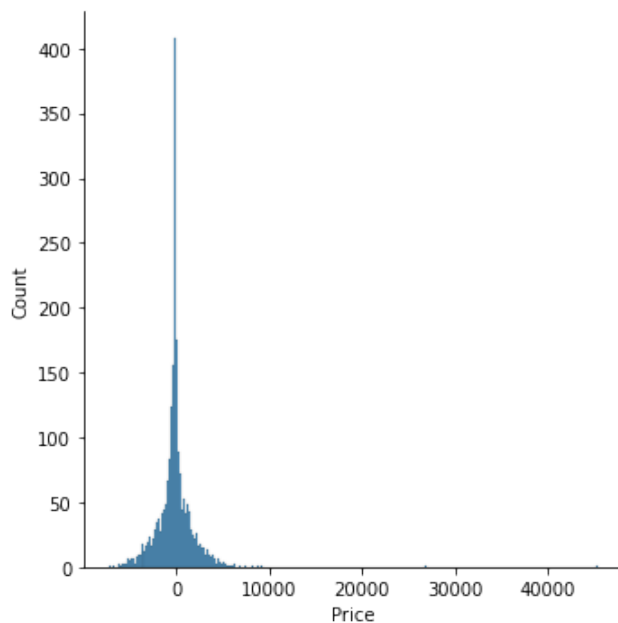
```
In [94]: rf_model.score(X_test, y_test)
```

```
Out[94]: 0.7970587091287554
```

```
In [95]: import seaborn as sns

# Assuming y_test and y_pred are already defined
sns.displot(y_test - y_pred)
```

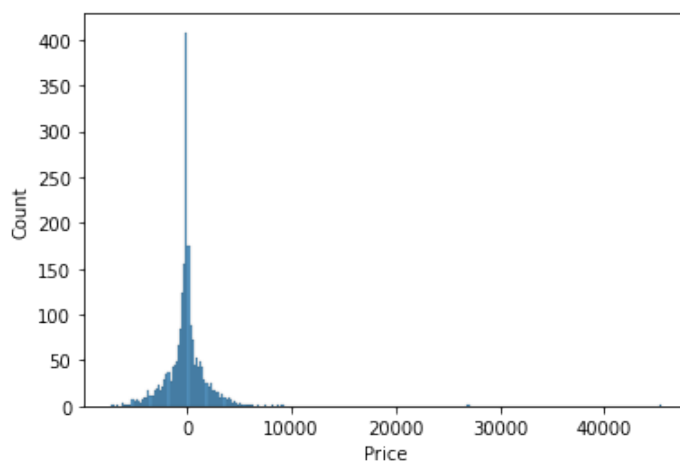
```
Out[95]: <seaborn.axisgrid.FacetGrid at 0x1b8d9a72700>
```



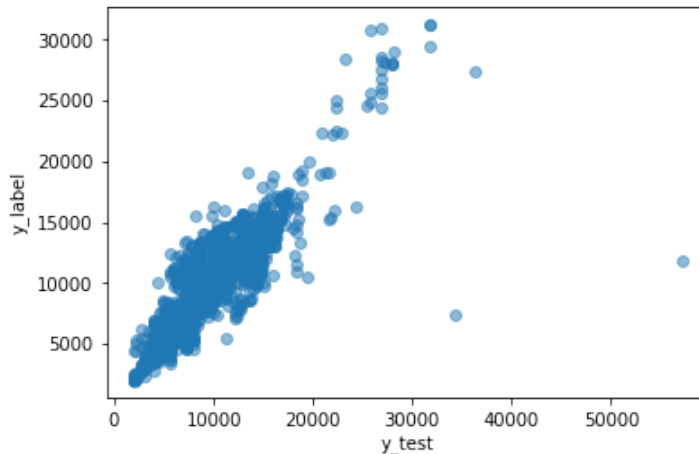
```
In [96]: import seaborn as sns

# Assuming y_test and y_pred are already defined
sns.histplot(y_test - y_pred)
```

```
Out[96]: <Axes: xlabel='Price', ylabel='Count'>
```



```
In [97]: ▶ plt.scatter(y_test, y_pred, alpha=0.5)
plt.xlabel('y_test')
plt.ylabel('y_label')
plt.show()
```



```
In [98]: ▶ from sklearn import metrics
```

```
In [99]: ▶ 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: 1182.1520932525332
MSE: 4375831.544263874
RMSE: 2091.848833989654
```

```
In [100]: ▶ metrics.r2_score(y_test, y_pred)
```

```
Out[100]: 0.7970587091287554
```

The model has the r2 score as 79 percent

To increase the percentage of model evaluation, we check the hyperparameter tuning and find the best parameters for the model training

## Hyperparameter Tuning

```
In [101]: ▶ from sklearn.model_selection import RandomizedSearchCV
```

```
In [139]: ▶ # Randomized Search CV

## Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start=100, stop=1200, num=12)]
## Number of features to consider at every split
max_features = ['auto', 'sqrt']
## Maximum number of level in tree
max_depth = [int(x) for x in np.linspace(5, 30, num=6)]
## Minimum number of samples required to split a node
min_samples_split = [2, 5, 10, 15, 100]
## Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 5, 10]
```

```
In [103]: > ## create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf}
print(random_grid)

{'n_estimators': [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200],
 'max_features': ['auto', 'sqrt'], 'max_depth': [5, 10, 15, 20, 25, 30], 'min_sample
s_split': [2, 5, 10, 15, 100], 'min_samples_leaf': [1, 2, 5, 10]}
```

```
In [108]: > rf_random = RandomizedSearchCV(estimator=rf_model, param_distributions=random_grid,
                                           n_iter=10, cv=5, verbose=2, random_state=42, n_jobs=1)
```

```
In [119]: > print(rf_random.best_params_)

{'n_estimators': 100, 'min_samples_split': 5, 'min_samples_leaf': 1, 'max_feature
s': 'sqrt', 'max_depth': 20, 'bootstrap': False}
```

These are the best parameters for the evaluation

So I will train the data on the best parameters

```
In [118]: > rf_random.fit(X_train, y_train)
```

Fitting 3 folds for each of 10 candidates, totalling 30 fits

```
Out[118]: RandomizedSearchCV
           estimator: RandomForestRegressor
               RandomForestRegressor
```

([https://scikit-learn.org/1.4/modules/generated/sklearn.model\\_selection.RandC](https://scikit-learn.org/1.4/modules/generated/sklearn.model_selection.RandC))  
(<https://scikit-learn.org/1.4/modules/generated/sklearn.ensemble.RandomForestRegri>)

```
In [117]: > rf_random.best_params_

Out[117]: {'n_estimators': 100,
           'min_samples_split': 5,
           'min_samples_leaf': 1,
           'max_features': 'sqrt',
           'max_depth': 20,
           'bootstrap': False}
```

```

In [116]: from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestRegressor
import numpy as np

# Define the hyperparameter grid
param_grid = {
    'n_estimators': [100, 300, 500, 800, 1000],
    'max_features': ['auto', 'sqrt'],
    'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
}

# Create a base model
rf = RandomForestRegressor()

# Random search of parameters
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = param_grid, n_i

# Fit the random search model
rf_random.fit(X_train, y_train)

```

Fitting 3 folds for each of 10 candidates, totalling 30 fits

```

Out[116]: RandomizedSearchCV
           estimator: RandomForestRegressor
             RandomForestRegressor

```

([https://scikit-learn.org/1.4/modules/generated/sklearn.model\\_selection.Randc](https://scikit-learn.org/1.4/modules/generated/sklearn.model_selection.Randc))

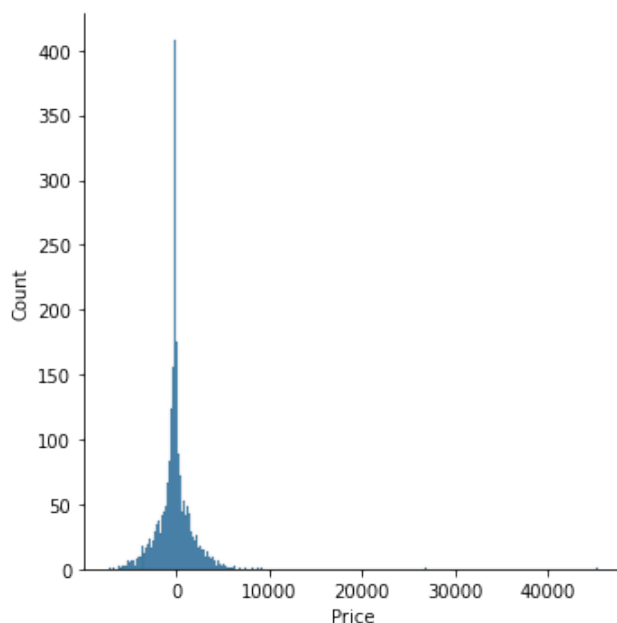
(<https://scikit-learn.org/1.4/modules/generated/sklearn.ensemble.RandomForestRegr>)

```

In [112]: plt.figure(figsize=(8, 8))
sns.displot(y_test-y_pred)
plt.show()

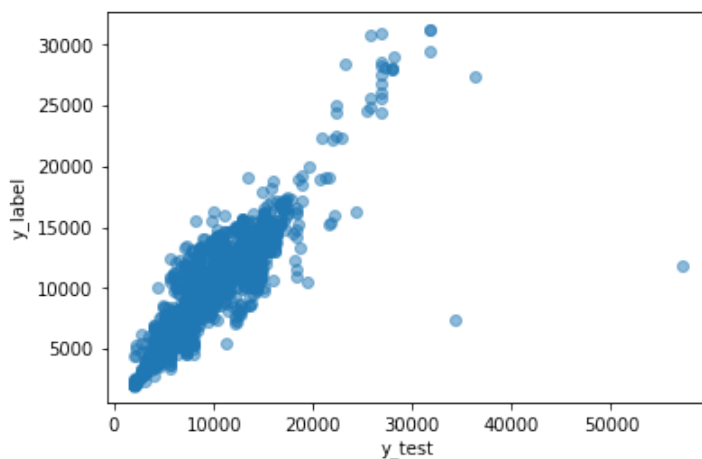
```

<Figure size 576x576 with 0 Axes>





```
In [113]: ▶ plt.scatter(y_test, y_pred, alpha=0.5)
plt.xlabel('y_test')
plt.ylabel('y_label')
plt.show()
```



```
In [114]: ▶ 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: 1182.1520932525332
MSE: 4375831.544263874
RMSE: 2091.848833989654
```

```
In [115]: ▶ metrics.r2_score(y_test, y_pred)
```

```
Out[115]: 0.7970587091287554
```

The model has the r2 score for the prediction is 81 percent