Flight Price Prediction

AI Project

Team

Shreyas Palod RA1911003010793

Kritika Agarwal RA1911003010808

Shubham Kumar RA1911003010794

Akash Roy Choudhary RA191003010795

In [1]:

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

sns.set()
```

In [2]:

```
train_data = pd.read_csv("Data_Train.csv")
```

In [3]:

```
pd.set_option('display.max_columns', None)
```

In [4]:

```
train_data.head()
```

Out[4]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	T
0	IndiGo	24-03-2019	Banglore	New Delhi	BLR ? DEL	22:20	22-03-2022 01:10	2h 50m	
1	Air India	01-05-2019	Kolkata	Banglore	CCU ? IXR ? BBI ? BLR	05:50	13:15	7h 25m	
2	Jet Airways	09-06-2019	Delhi	Cochin	DEL ? LKO ? BOM ? COK	09:25	10-06-2022 04:25	19h	
3	IndiGo	12-05-2019	Kolkata	Banglore	CCU ? NAG ? BLR	18:05	23:30	5h 25m	
4	IndiGo	01-03-2019	Banglore	New Delhi	BLR ? NAG ? DEL	16:50	21:35	4h 45m	

In [5]:

```
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10683 entries, 0 to 10682
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   Airline          10683 non-null  object
 1   Date_of_Journey  10683 non-null  object
 2   Source           10683 non-null  object
 3   Destination      10683 non-null  object
 4   Route            10682 non-null  object
 5   Dep_Time         10683 non-null  object
 6   Arrival_Time     10683 non-null  object
 7   Duration         10683 non-null  object
 8   Total_Stops      10682 non-null  object
 9   Additional_Info  10683 non-null  object
 10  Price            10683 non-null  int64 
dtypes: int64(1), object(10)
memory usage: 918.2+ KB
```

In [6]:

```
train_data["Duration"].value_counts()
```

Out[6]:

```
2h 50m     550
1h 30m     386
2h 45m     337
2h 55m     337
2h 35m     329
          ...
13h 35m      1
29h 10m      1
29h 40m      1
30h 15m      1
33h 20m      1
Name: Duration, Length: 368, dtype: int64
```

In [7]:

```
train_data.dropna(inplace = True)
```

In [8]:

```
train_data.isnull().sum()
```

Out[8]:

```
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
```

EDA

From description we can see that Date_of_Journey is a object data type,
Therefore, we have to convert this datatype into timestamp so as to use this column properly for prediction

In [9]:

```
train_data["Journey_day"] = pd.to_datetime(train_data.Date_of_Journey, format="%d-%m-%Y").d
```

In [10]:

```
train_data["Journey_month"] = pd.to_datetime(train_data["Date_of_Journey"], format = "%d-%m
```

In [11]:

```
train_data.head()
```

Out[11]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	T
0	IndiGo	24-03-2019	Banglore	New Delhi	BLR ? DEL	22:20	22-03-2022 01:10	2h 50m	
1	Air India	01-05-2019	Kolkata	Banglore	CCU ? IXR ? BBI ? BLR	05:50	13:15	7h 25m	
2	Jet Airways	09-06-2019	Delhi	Cochin	DEL ? LKO ? BOM ? COK	09:25	10-06-2022 04:25	19h	
3	IndiGo	12-05-2019	Kolkata	Banglore	CCU ? NAG ? BLR	18:05	23:30	5h 25m	
4	IndiGo	01-03-2019	Banglore	New Delhi	BLR ? NAG ? DEL	16:50	21:35	4h 45m	

In [12]:

```
# Since we have converted Date_of_Journey column into integers, Now we can drop as it is of

train_data.drop(["Date_of_Journey"], axis = 1, inplace = True)
```

In [13]:

```
# Departure time is when a plane leaves the gate.
# Similar to Date_of_Journey we can extract values from Dep_Time

# Extracting Hours
train_data["Dep_hour"] = pd.to_datetime(train_data["Dep_Time"]).dt.hour

# Extracting Minutes
train_data["Dep_min"] = pd.to_datetime(train_data["Dep_Time"]).dt.minute

# Now we can drop Dep_Time as it is of no use
train_data.drop(["Dep_Time"], axis = 1, inplace = True)
```

In [14]:

```
train_data.head()
```

Out[14]:

	Airline	Source	Destination	Route	Arrival_Time	Duration	Total_Stops	Additional_Info	
0	IndiGo	Banglore	New Delhi	BLR ? DEL	22-03-2022 01:10	2h 50m	non-stop	No info	
1	Air India	Kolkata	Banglore	CCU ? IXR ? BBI ? BLR	13:15	7h 25m	2 stops	No info	
2	Jet Airways	Delhi	Cochin	DEL ? LKO ? BOM ? COK	10-06-2022 04:25	19h	2 stops	No info	1
3	IndiGo	Kolkata	Banglore	CCU ? NAG ? BLR	23:30	5h 25m	1 stop	No info	
4	IndiGo	Banglore	New Delhi	BLR ? NAG ? DEL	21:35	4h 45m	1 stop	No info	1

In [15]:

```
# Arrival time is when the plane pulls up to the gate.
# Similar to Date_of_Journey we can extract values from Arrival_Time

# Extracting Hours
train_data["Arrival_hour"] = pd.to_datetime(train_data.Arrival_Time).dt.hour

# Extracting Minutes
train_data["Arrival_min"] = pd.to_datetime(train_data.Arrival_Time).dt.minute

# Now we can drop Arrival_Time as it is of no use
train_data.drop(["Arrival_Time"], axis = 1, inplace = True)
```

In [16]:

```
train_data.head()
```

Out[16]:

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

In [17]:

```
# Time taken by plane to reach destination is called Duration
# It is the differnce betwwen Departure Time and Arrival time


# Assigning and converting Duration column into list
duration = list(train_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 dur
    duration_mins.append(int(duration[i].split(sep = "m")[0].split()[-1]))   # Extracts onl
```

In [18]:

```
# Adding duration_hours and duration_mins list to train_data dataframe

train_data["Duration_hours"] = duration_hours
train_data["Duration_mins"] = duration_mins
```

In [19]:

```
train_data.drop(["Duration"], axis = 1, inplace = True)
```

In [20]:

```
train_data.head()
```

Out[20]:

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

Handling Categorical Data

One can find many ways to handle categorical data. Some of them categorical data are,

1. **Nominal data** --> data are not in any order --> **OneHotEncoder** is used in this case
2. **Ordinal data** --> data are in order --> **LabelEncoder** is used in this case

In [21]:

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

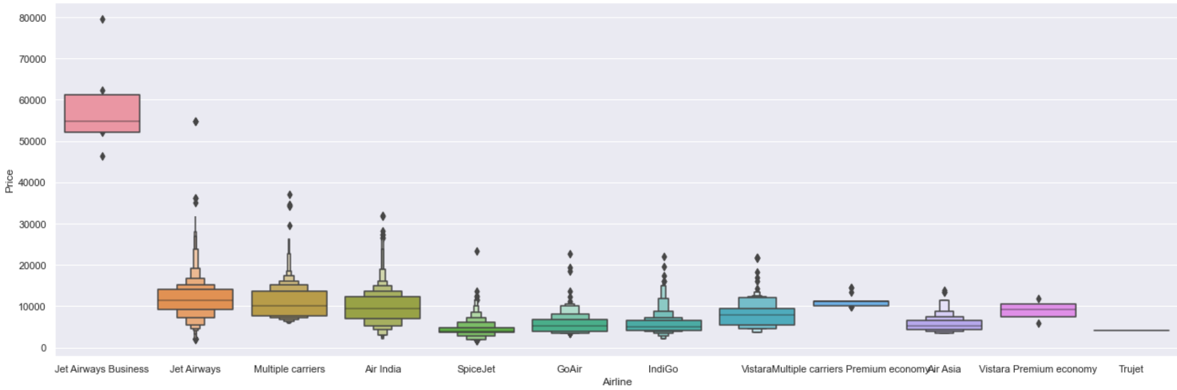
Out[21]:

```
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: Airline, dtype: int64
```

In [22]:

```
# From graph we can see that Jet Airways Business have the highest Price.
# Apart from the first Airline almost all are having similar median

# Airline vs Price
sns.catplot(y = "Price", x = "Airline", data = train_data.sort_values("Price", ascending =
plt.show()
```



In [23]:

```
# As Airline is Nominal Categorical data we will perform OneHotEncoding

Airline = train_data[["Airline"]]

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

Airline.head()
```

Out[23]:

	Airline_Air India	Airline_GoAir	Airline_IndiGo	Airline_Jet Airways	Airline_Jet Airways Business	Airline_Multiple carriers	Airline_Mul car Pren econ
0	0	0	1	0	0	0	
1	1	0	0	0	0	0	
2	0	0	0	1	0	0	
3	0	0	1	0	0	0	
4	0	0	1	0	0	0	

In [24]:

```
train_data["Source"].value_counts()
```

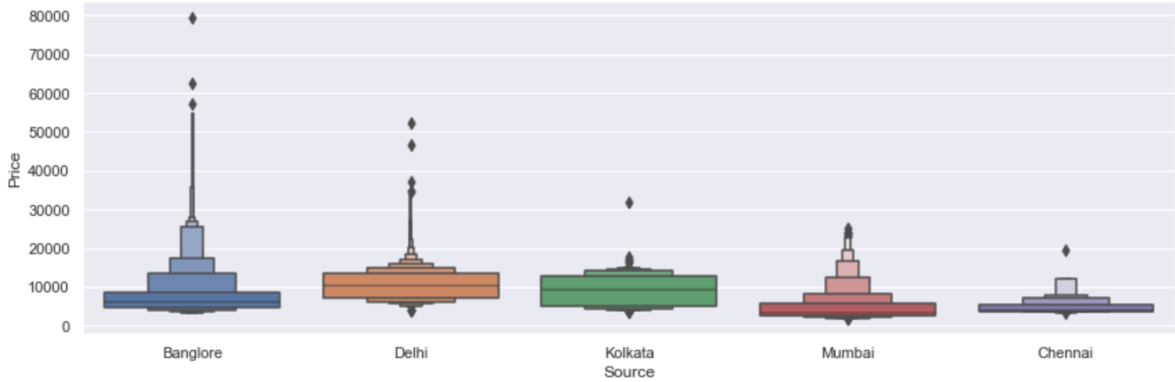
Out[24]:

```
Delhi       4536
Kolkata     2871
Banglore    2197
Mumbai       697
Chennai      381
Name: Source, dtype: int64
```

In [25]:

```
# Source vs Price

sns.catplot(y = "Price", x = "Source", data = train_data.sort_values("Price", ascending = F
plt.show()
```



In [26]:

```
# As Source is Nominal Categorical data we will perform OneHotEncoding

Source = train_data[["Source"]]

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

Source.head()
```

Out[26]:

	Source_Chennai	Source_Delhi	Source_Kolkata	Source_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

In [27]:

```
train_data["Destination"].value_counts()
```

Out[27]:

```
Cochin       4536
Banglore     2871
Delhi        1265
New Delhi     932
Hyderabad     697
Kolkata       381
Name: Destination, dtype: int64
```

In [28]:

```
# As Destination is Nominal Categorical data we will perform OneHotEncoding

Destination = train_data[["Destination"]]

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

Destination.head()
```

Out[28]:

	Destination_Cochin	Destination_Delhi	Destination_Hyderabad	Destination_Kolkata	Destinatio
0	0	0	0	0	
1	0	0	0	0	
2	1	0	0	0	
3	0	0	0	0	
4	0	0	0	0	

In [29]:

```
train_data["Route"]
```

Out[29]:

```
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
```

In [30]:

```
# Additional_Info contains almost 80% no_info
# Route and Total_Stops are related to each other

train_data.drop(["Route", "Additional_Info"], axis = 1, inplace = True)
```

In [31]:

```
train_data["Total_Stops"].value_counts()
```

Out[31]:

```
1 stop      5625
non-stop    3491
2 stops     1520
3 stops       45
4 stops        1
Name: Total_Stops, dtype: int64
```

In [32]:

```
# As this is case of Ordinal Categorical type we perform LabelEncoder
# Here Values are assigned with corresponding keys

train_data.replace({"non-stop": 0, "1 stop": 1, "2 stops": 2, "3 stops": 3, "4 stops": 4},
```

In [33]:

```
train_data.head()
```

Out[33]:

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

In [34]:

```
# Concatenate dataframe --> train_data + Airline + Source + Destination

data_train = pd.concat([train_data, Airline, Source, Destination], axis = 1)
```

In [35]:

```
data_train.head()
```

Out[35]:

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

In [36]:

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

In [37]:

```
data_train.head()
```

Out[37]:

	Total_Stops	Price	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	Arrival_
0	0	3897	24	3	22	20	1	
1	2	7662	1	5	5	50	13	
2	2	13882	9	6	9	25	4	
3	1	6218	12	5	18	5	23	
4	1	13302	1	3	16	50	21	

In [38]:

```
data_train.shape
```

Out[38]:

```
(10682, 30)
```

Test set

In [39]:

```
test_data = pd.read_csv("Test_set.csv")
```

In [40]:

```
test_data.head()
```

Out[40]:

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

In [41]:

```
# Preprocessing

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.
test_data["Journey_month"] = pd.to_datetime(test_data["Date_of_Journey"], format = "%d/%m/%
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 dur
    duration_mins.append(int(duration[i].split(sep = "m")[0].split()[-1]))   # Extracts onl

# 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)
```

```python
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}, i

# 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
---------------------------------------------------------------------------
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

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

Destination
---------------------------------------------------------------------------
Cochin       1145
Banglore      710
Delhi         317
New Delhi     238
Hyderabad     186
Kolkata        75
Name: Destination, dtype: int64


Shape of test data :  (2671, 28)
```

In [42]:

```
data_test.head()
```

Out[42]:

	Total_Stops	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	Arrival_min	D
0	1	6	6	17	30	4	25	
1	1	12	5	6	20	10	20	
2	1	21	5	19	15	19	0	
3	1	21	5	8	0	21	0	
4	0	24	6	23	55	2	45	

Feature Selection

Finding out the best feature which will contribute and have good relation with target variable. Following are some of the feature selection methods,

1. **heatmap**
2. **feature_importance_**
3. **SelectKBest**

In [43]:

```
data_train.shape
```

Out[43]:

```
(10682, 30)
```

In [44]:

```
data_train.columns
```

Out[44]:

```
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_IndiG
o',
       'Airline_Jet Airways', 'Airline_Jet Airways Business',
       'Airline_Multiple carriers',
       'Airline_Multiple carriers Premium economy', 'Airline_SpiceJet',
       'Airline_Trujet', 'Airline_Vistara', 'Airline_Vistara Premium econom
y',
       'Source_Chennai', 'Source_Delhi', 'Source_Kolkata', 'Source_Mumbai',
       'Destination_Cochin', 'Destination_Delhi', 'Destination_Hyderabad',
       'Destination_Kolkata', 'Destination_New Delhi'],
      dtype='object')
```

In [45]:

```
X = data_train.loc[:, ['Total_Stops', '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']]
X.head()
```

Out[45]:

	Total_Stops	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	Arrival_min	D
0	0	24	3	22	20	1	10	
1	2	1	5	5	50	13	15	
2	2	9	6	9	25	4	25	
3	1	12	5	18	5	23	30	
4	1	1	3	16	50	21	35	

In [46]:

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

Out[46]:

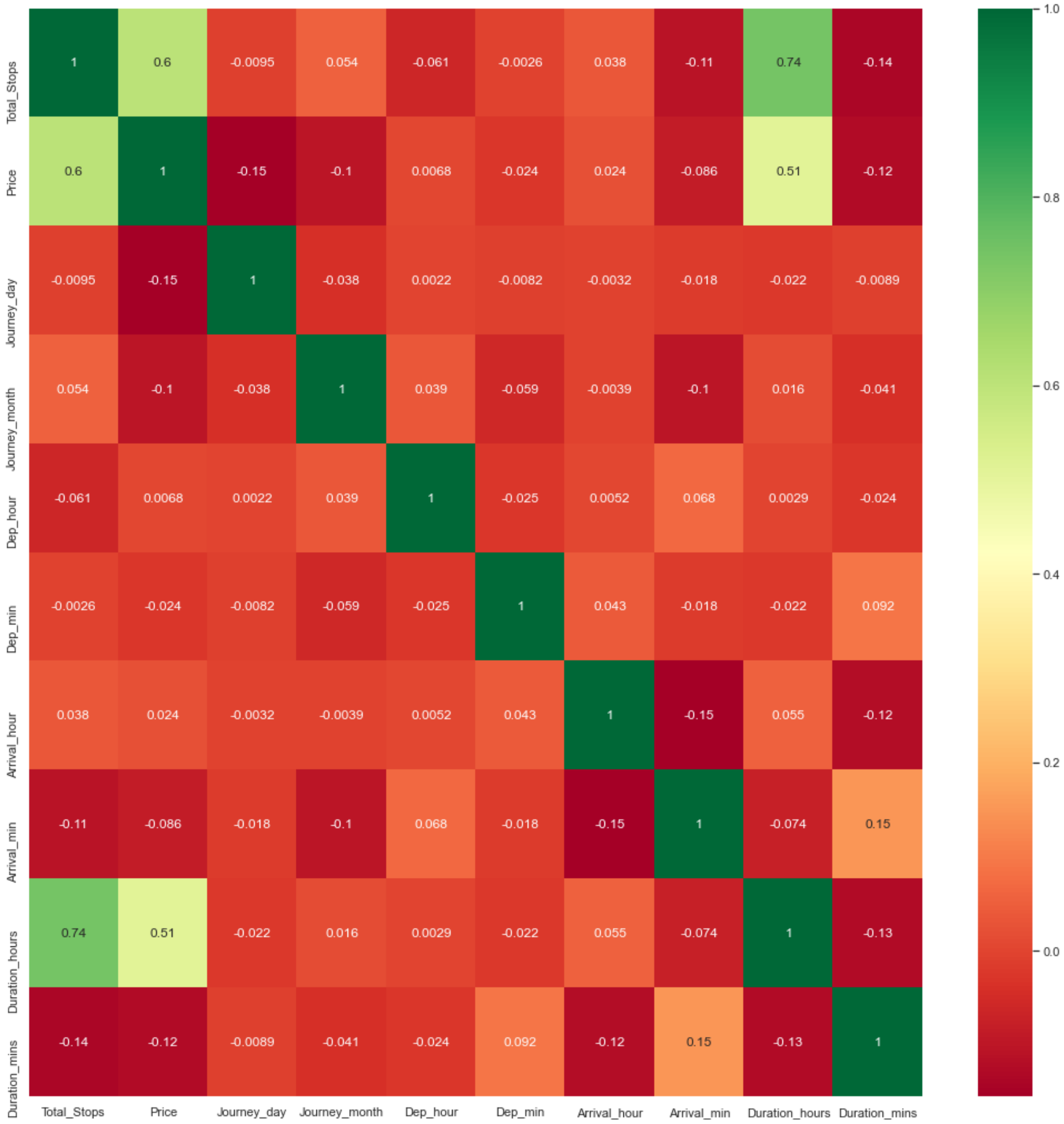
```
0     3897
1     7662
2    13882
3     6218
4    13302
Name: Price, dtype: int64
```

In [47]:

```
# Finds correlation between Independent and dependent attributes

plt.figure(figsize = (18,18))
sns.heatmap(train_data.corr(), annot = True, cmap = "RdYlGn")

plt.show()
```



In [48]:

```
# Important feature using ExtraTreesRegressor

from sklearn.ensemble import ExtraTreesRegressor
selection = ExtraTreesRegressor()
selection.fit(X, y)
```

Out[48]:

```
ExtraTreesRegressor()
```

In [49]:

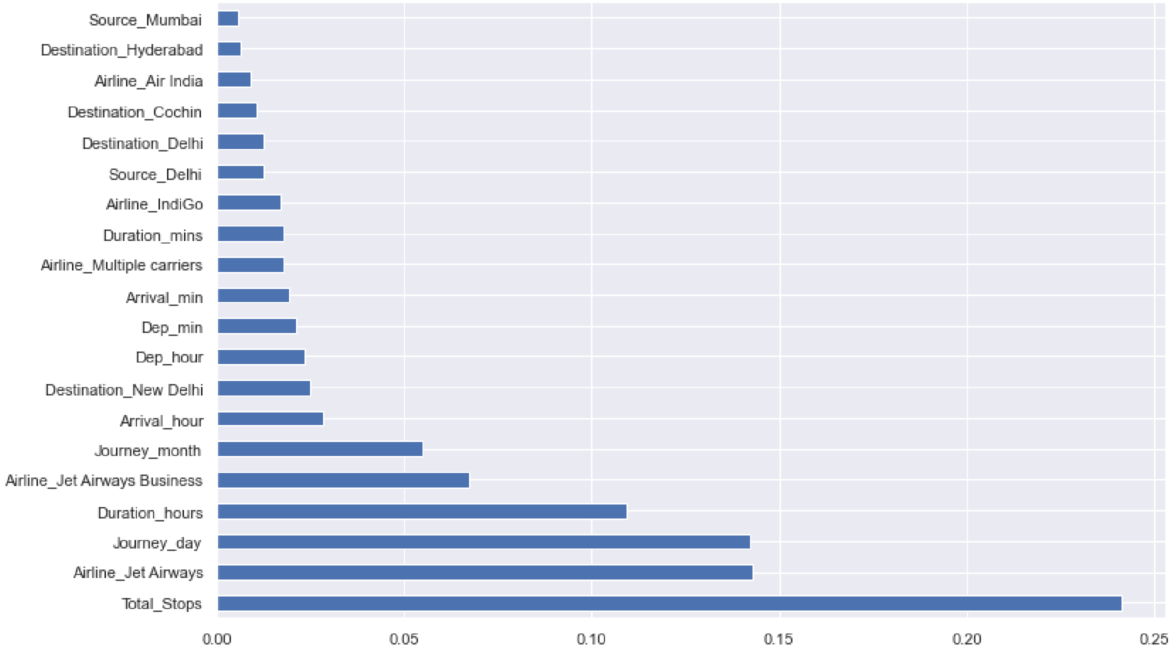
```
print(selection.feature_importances_)
```

```
[2.41001729e-01 1.42132136e-01 5.46805823e-02 2.36014299e-02
 2.12523285e-02 2.82643793e-02 1.91978118e-02 1.09236940e-01
 1.77896558e-02 9.13745435e-03 1.93484570e-03 1.70050697e-02
 1.43061897e-01 6.73873570e-02 1.78506227e-02 8.82456771e-04
 3.13228105e-03 1.03000131e-04 5.12590416e-03 8.11249931e-05
 5.10834842e-04 1.26719805e-02 3.20225814e-03 5.56697231e-03
 1.06136533e-02 1.25250515e-02 6.47943446e-03 4.47129085e-04
 2.51236791e-02]
```

In [50]:

```
#plot graph of feature importances for better visualization

plt.figure(figsize = (12,8))
feat_importances = pd.Series(selection.feature_importances_, index=X.columns)
feat_importances.nlargest(20).plot(kind='barh')
plt.show()
```



Fitting model using Random Forest

1. Split dataset into train and test set in order to prediction w.r.t X_test
2. If needed do scaling of data
    - Scaling is not done in Random forest
3. Import model
4. Fit the data
5. Predict w.r.t X_test
6. In regression check **RSME** Score
7. Plot graph

In [51]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 4
```

In [52]:

```
from sklearn.ensemble import RandomForestRegressor
reg_rf = RandomForestRegressor()
reg_rf.fit(X_train, y_train)
```

Out[52]:

```
RandomForestRegressor()
```

In [53]:

```
y_pred = reg_rf.predict(X_test)
```

In [54]:

```
reg_rf.score(X_train, y_train)
```

Out[54]:

```
0.9525507165440114
```

In [55]:

```
reg_rf.score(X_test, y_test)
```

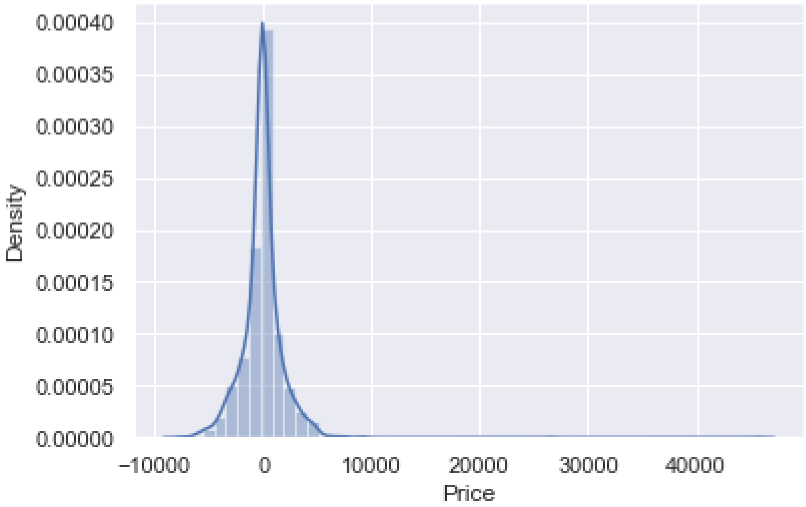
Out[55]:

```
0.7981962137917512
```

In [56]:

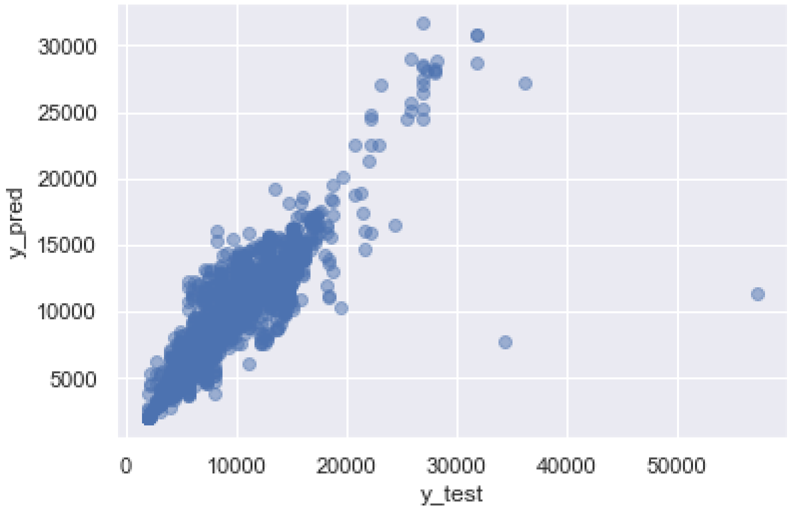
```
sns.distplot(y_test-y_pred)
plt.show()
```

```
C:\Users\LENOVO\anaconda3\lib\site-packages\seaborn\distributions.py:2557: F
utureWarning: `distplot` is a deprecated function and will be removed in a f
uture version. Please adapt your code to use either `displot` (a figure-leve
l function with similar flexibility) or `histplot` (an axes-level function f
or histograms).
  warnings.warn(msg, FutureWarning)
```



In [57]:

```
plt.scatter(y_test, y_pred, alpha = 0.5)
plt.xlabel("y_test")
plt.ylabel("y_pred")
plt.show()
```



In [58]:

```
from sklearn import metrics
```

In [59]:

```
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: 1178.386659050589
MSE: 4351304.604651364
RMSE: 2085.978093042054
```

In [60]:

```
# RMSE/(max(DV)-min(DV))

2090.5509/(max(y)-min(y))
```

Out[60]:

```
0.026887077025966846
```

In [61]:

```
metrics.r2_score(y_test, y_pred)
```

Out[61]:

```
0.7981962137917512
```

In [ ]:

```

```

Hyperparameter Tuning

- Choose following method for hyperparameter tuning
    1. **RandomizedSearchCV** --> Fast
    2. **GridSearchCV**
- Assign hyperparameters in form of dictionery
- Fit the model
- Check best paramters and best score

In [62]:

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

In [63]:

```
#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 levels 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 [64]:

```
# 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}
```

In [65]:

```
# Random search of parameters, using 5 fold cross validation,
# search across 100 different combinations
rf_random = RandomizedSearchCV(estimator = reg_rf, param_distributions = random_grid,scorin
```

In [66]:

```
rf_random.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_sp
lit=5, n_estimators=900; total time=   3.1s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_sp
lit=5, n_estimators=900; total time=   3.1s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_sp
lit=5, n_estimators=900; total time=   3.1s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_sp
lit=5, n_estimators=900; total time=   3.1s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_sp
lit=5, n_estimators=900; total time=   3.1s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_sp
lit=10, n_estimators=1100; total time=   4.9s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_sp
lit=10, n_estimators=1100; total time=   5.2s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_sp
lit=10, n_estimators=1100; total time=   4.9s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_sp
lit=10, n_estimators=1100; total time=   4.8s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_sp
lit=10, n_estimators=1100; total time=   5.0s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_sp
lit=100, n_estimators=300; total time=   3.1s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_sp
lit=100, n_estimators=300; total time=   3.1s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_sp
lit=100, n_estimators=300; total time=   3.0s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_sp
lit=100, n_estimators=300; total time=   2.9s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_sp
lit=100, n_estimators=300; total time=   3.0s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_sp
lit=5, n_estimators=400; total time=   5.4s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_sp
lit=5, n_estimators=400; total time=   5.3s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_sp
lit=5, n_estimators=400; total time=   5.3s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_sp
lit=5, n_estimators=400; total time=   5.3s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_sp
lit=5, n_estimators=400; total time=   5.2s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_s
plit=5, n_estimators=700; total time=   8.3s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_s
plit=5, n_estimators=700; total time=   8.2s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_s
plit=5, n_estimators=700; total time=   8.4s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_s
plit=5, n_estimators=700; total time=   8.3s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_s
plit=5, n_estimators=700; total time=   8.1s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_sp
lit=2, n_estimators=1000; total time=   7.4s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_sp
lit=2, n_estimators=1000; total time=   7.5s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_sp
lit=2, n_estimators=1000; total time=   7.2s
```

```
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_sp
lit=2, n_estimators=1000; total time=   7.2s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_sp
lit=2, n_estimators=1000; total time=   7.4s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_sp
lit=15, n_estimators=1100; total time=   2.5s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_sp
lit=15, n_estimators=1100; total time=   2.7s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_sp
lit=15, n_estimators=1100; total time=   2.7s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_sp
lit=15, n_estimators=1100; total time=   2.6s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_sp
lit=15, n_estimators=1100; total time=   2.6s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_sp
lit=15, n_estimators=300; total time=   1.3s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_sp
lit=15, n_estimators=300; total time=   1.2s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_sp
lit=15, n_estimators=300; total time=   1.2s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_sp
lit=15, n_estimators=300; total time=   1.2s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_sp
lit=15, n_estimators=300; total time=   1.2s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_spl
it=10, n_estimators=700; total time=   1.6s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_spl
it=10, n_estimators=700; total time=   1.5s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_spl
it=10, n_estimators=700; total time=   1.5s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_spl
it=10, n_estimators=700; total time=   1.5s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_spl
it=10, n_estimators=700; total time=   1.5s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_sp
lit=15, n_estimators=700; total time=   9.8s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_sp
lit=15, n_estimators=700; total time=   9.5s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_sp
lit=15, n_estimators=700; total time=  10.1s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_sp
lit=15, n_estimators=700; total time=   9.8s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_sp
lit=15, n_estimators=700; total time=  10.0s
```

Out[66]:

```
RandomizedSearchCV(cv=5, estimator=RandomForestRegressor(), n_jobs=1,
                   param_distributions={'max_depth': [5, 10, 15, 20, 25, 3
0],
                                        'max_features': ['auto', 'sqrt'],
                                        'min_samples_leaf': [1, 2, 5, 10],
                                        'min_samples_split': [2, 5, 10, 15,
                                                              100],
                                        'n_estimators': [100, 200, 300, 400,
                                                         500, 600, 700, 800,
                                                         900, 1000, 1100,
                                                         1200]},
                   random_state=42, scoring='neg_mean_squared_error',
                   verbose=2)
```

In [67]:

```
rf_random.best_params_
```

Out[67]:

```
{'n_estimators': 700,
 'min_samples_split': 15,
 'min_samples_leaf': 1,
 'max_features': 'auto',
 'max_depth': 20}
```

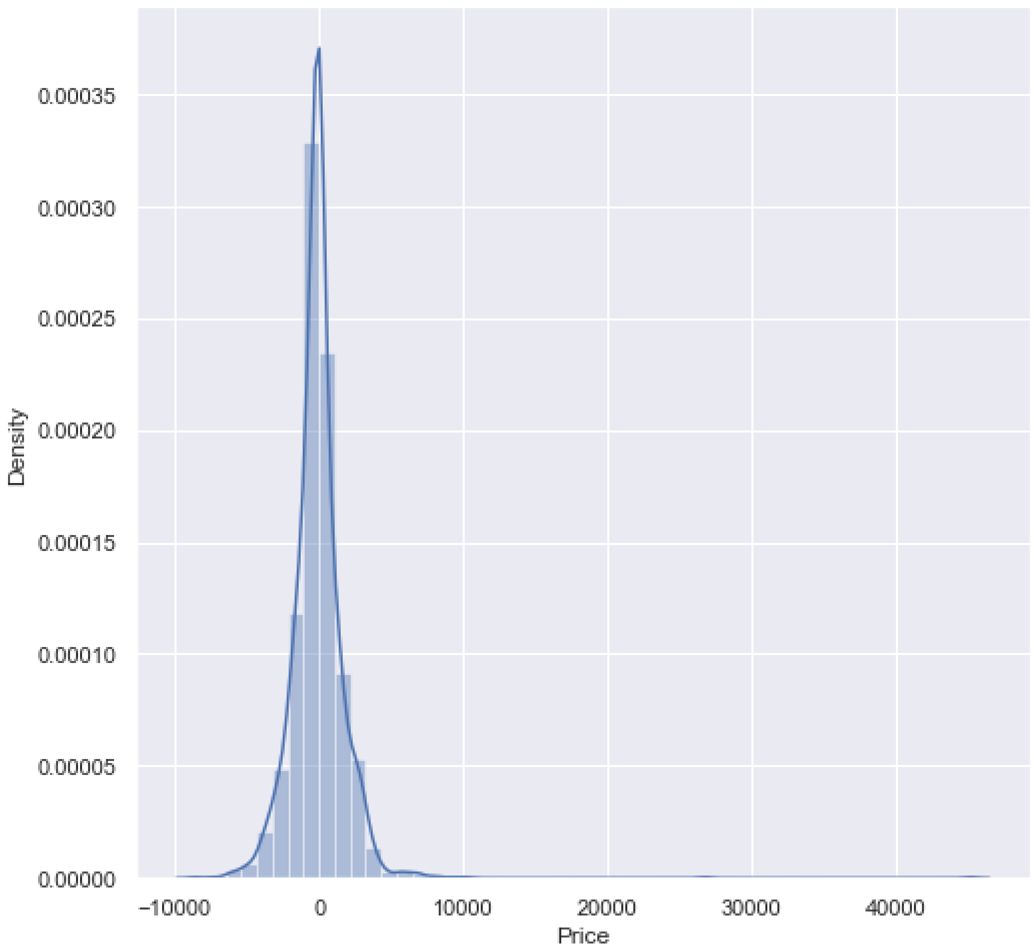
In [68]:

```
prediction = rf_random.predict(X_test)
```

In [69]:

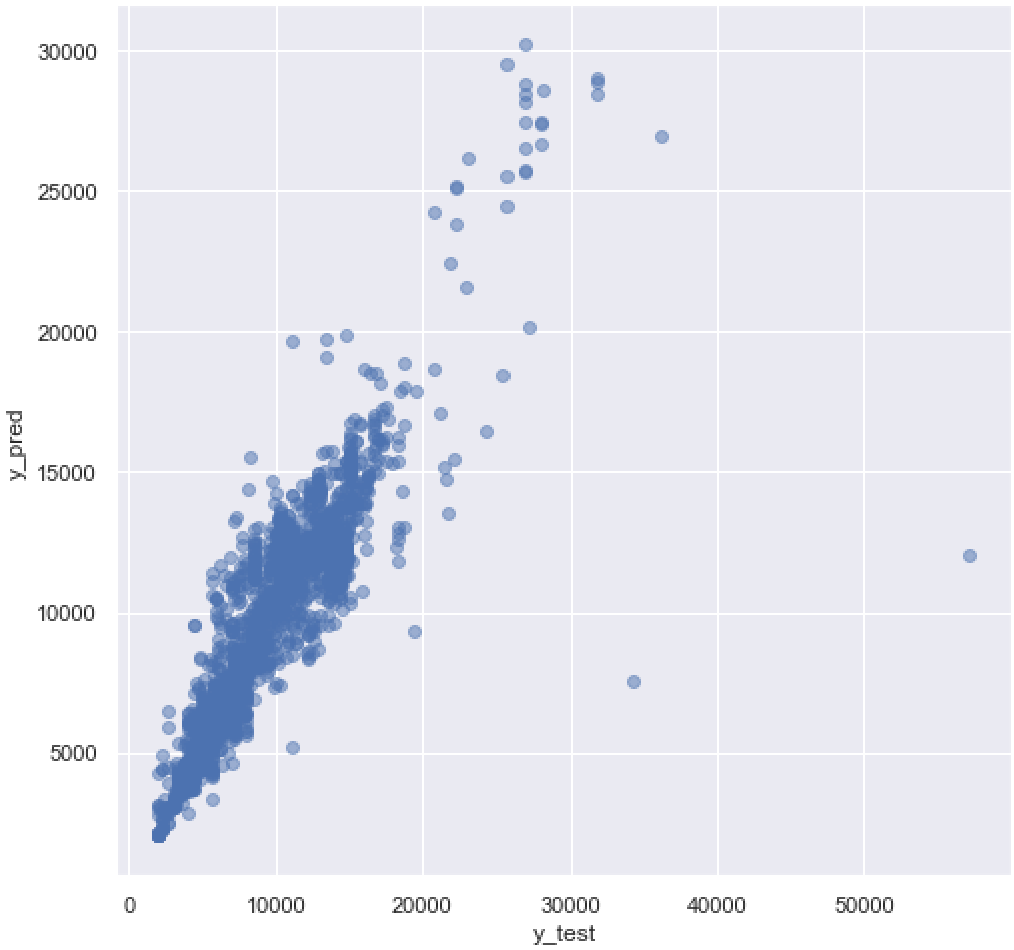
```
plt.figure(figsize = (8,8))
sns.distplot(y_test-prediction)
plt.show()
```

```
C:\Users\LENOVO\anaconda3\lib\site-packages\seaborn\distributions.py:2557: F
utureWarning: `distplot` is a deprecated function and will be removed in a f
uture version. Please adapt your code to use either `displot` (a figure-leve
l function with similar flexibility) or `histplot` (an axes-level function f
or histograms).
  warnings.warn(msg, FutureWarning)
```



In [70]:

```
plt.figure(figsize = (8,8))
plt.scatter(y_test, prediction, alpha = 0.5)
plt.xlabel("y_test")
plt.ylabel("y_pred")
plt.show()
```



In [71]:

```
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: 1162.5465497521143
MSE: 4033404.726960512
RMSE: 2008.3338186069846
```

In [75]: