Machine Learning Case Study
Flight Price Analysis

Submitted By,
M.Kavitha.

ACKNOWLEDGMENT

This presentation includes the flight price prediction done by myself with reference to the data analysis prepared by me using web scraping from the website Yatra .Also referred to Google for some detailed learning in the analysis report writing for the completion of the project.

INTRODUCTION

Business Problem Framing

With the covid 19 impact in the market, we have seen lot of changes in the Economics market.People travel for Business around the Country to explore the business .They are in the situation to make their business fast and so they prefer making their trips in short so they avail booking flight for maximum less price.They will be ahead in booking the flight in advance according to their plan.We are going to scrape and find out in the social website how far we can reduce our travel charges by booking in advance.Here we have collected the data in Yatra.com ,finding the flight availability for some source and Destination at less price.With the change in market due to covid 19 impact, our client is facing problems with their previous booked flight changes and cancellation charges. So, they are looking for new machine learning models from new data. We have to make Flight charges according to the collected data and analyze the model and predict the price charges.

Conceptual Background of the Domain Problem

The objective of this article is to predict flight prices given the various parameters. Data used in this article is was scraped from Yatra.com.

The data of Flight services available for few dates from the Location NewDelhi to Mumbai from the website Yatra.com We used web scraping for this. We fetched data for above mentioned locations. The number of columns for data depends on the proper scraping we are doing and also the website which we are scraping .

The dataset contains the data of the used Flight Sevices available on different dates.On the basis of the data we have to predict the price of the Flight charges.The dataset contains the data like details of the 'Airline', 'Date_of_Journey', 'Source', 'Destination', 'Dep_Time','Arrival_Time', 'Duration', 'Total_Stops', 'Price'.

Review of Literature:

I started the Research by first reading and analyzing the data collected by myself . Price is the target

column here. All the features are then analyzed, missing data handling, outlier detection, data cleaning are done. New features are extracted, redundant features dropped and categorical features are encoded accordingly. Then the data is split into train and test data and feature scaling is performed and finally model building is done with Hyper Parameter Tuning done with best parameters observed.

Motivation for the Problem Undertaken

Airline companies use complex algorithms to calculate flight prices given various conditions present at that particular time. These methods take financial, marketing, and various social factors into account to predict flight prices.

Nowadays, the number of people using flights has increased significantly. It is difficult for airlines to maintain prices since prices change dynamically due to different conditions. That's why we will try to use machine learning to solve this problem. This can help airlines by predicting what prices they can maintain. It can also help customers to predict future flight prices and plan their journey accordingly.

The main Objective behind the project is to perform the given task successfully and analyze the dataset thoroughly, learn the objective concepts and perform the prediction according to the provided dataset.
This will be a regression problem since the target or dependent variable is the price (continuous numeric value).

Analytical Problem Framing

· Mathematical/ Analytical Modeling of the Problem

1.Importing modules, Reading the data
2.Analyzing Numerical Features
Checking Statistical summary
Checking Distribution of numerical features .
Inspecting Correlation
Missing Value Handling
Encoding Categorical Features
Correcting data type
Univariate and Bivariate analysis,

DataVisualization.
3.Analyzing Categorical Features
Missing Value Handling
Encoding Categorical Features
Data Visualization
Dropping Redundant Features
4.Splitting data into Train and Test data
5.Comparing model coefficients
6.Model Evaluation
7.Choosing the final model and most significant features.
8.Evaluation of Regressor Models
9.HyperParameter Tuning.
10.Conclusion.

Data Sources and their formats

Data contains 1619 rows × 9 columns entries each having 9 variables. • Data does not have Null values. • Extensive EDA is performed to gain relationships of important variables and price.

• Data contains numerical as well as categorical variables.

• We have to build Machine Learning models, apply Regressor model coefficients and determine the optimal values of Hyper Parameters.
• We need to find important features which affect the price.

Data Description:

The procedure of extracting information from given raw data is called data analysis. Here we will use EDA module of the data-prep library to do this step. First, we will import the required libraries:

Importing Dataset

Importing the Dataset

```
import pandas as pd
import numpy as np
import seaborn as sns
import seaborn as sb
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings('ignore')
```

```
df=pd.read_csv('Flight_price.csv',error_bad_lines=False,delimiter='\t')
df
```

	Airline	Date_of_Journey	Source	Destination	Dep_Time	Arrival_Time	Duration	Total_Stops	Price
0	Air Asia	22-10-2021	New Delhi	Mumbai	08:00	14:35	6h 35m	1 Stop	5,953
1	Air Asia	22-10-2021	New Delhi	Mumbai	12:40	20:15	7h 35m	1 Stop	5,953
2	Air Asia	22-10-2021	New Delhi	Mumbai	11:55	20:15	8h 20m	1 Stop	5,953
3	Air Asia	22-10-2021	New Delhi	Mumbai	08:00	16:35	8h 35m	1 Stop	5,953
4	Air Asia	22-10-2021	New Delhi	Mumbai	04:55	14:15	9h 20m	1 Stop	5,953

Airline: Name of the airline used for traveling

Date_of_Journey: Date at which a person travelled
Source: Starting location of flight
Destination: Ending location of flight
Dep_Time: Departure time of flight from starting location
Arrival_Time: Arrival time of flight at destination
Duration: Duration of flight in hours/minutes
Total_Stops: Number of total stops flight took before landing at the destination.
Price: Price of the flight

Few observations about some of the variables:

1. ‘Price‘ will be our dependent variable and all remaining variables can be used as independent variables.

2. ‘Total_Stops‘ can be used to determine if the flight was direct or connecting.

In [10]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1619 entries, 0 to 1618
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   Airline          1619 non-null   object
 1   Date_of_Journey  1619 non-null   object
 2   Source           1619 non-null   object
 3   Destination      1619 non-null   object
 4   Dep_Time         1619 non-null   object
 5   Arrival_Time     1619 non-null   object
 6   Duration         1619 non-null   object
 7   Total_Stops      1619 non-null   object
 8   Price            1619 non-null   object
dtypes: object(9)
memory usage: 114.0+ KB
```

In [15]:
```
df.columns
```

Out[15]:
```
Index(['Airline', 'Date_of_Journey', 'Source', 'Destination', 'Dep_Time',
       'Arrival_Time', 'Duration', 'Total_Stops', 'Price'],
      dtype='object')
```

:
```
df['Duration'].value_counts()
```

:
```
2h 10m     167
2h 15m     116
2h 05m      98
2h 20m      64
2h 00m      33
          ...
16h 25m      1
9h 30m       1
17h 00m      1
16h 05m      1
25h 10m      1
Name: Duration, Length: 198, dtype: int64
```

:
```
df.shape
```

:
```
(1619, 9)
```

Missing Values

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

Out[13]: Airline            0
         Date_of_Journey    0
         Source             0
         Destination        0
         Dep_Time           0
         Arrival_Time       0
         Duration           0
         Total_Stops        0
         Price              0
         dtype: int64
```

Statistical Analysis

```
In [14]: df.describe()
```

Out[14]:

	Airline	Date_of_Journey	Source	Destination	Dep_Time	Arrival_Time	Duration	Total_Stops	Price
count	1619	1619	1619	1619	1619	1619	1619	1619	1619
unique	6	11	1	1	150	335	198	4	218
top	Vistara	01-11-2021	New Delhi	Mumbai	08:10	23:00	2h 10m	1 Stop	5,955
freq	459	166	1619	1619	64	45	167	1086	406

DataFrame Description

We have the dataset with prices of flight tickets for various airlines and between various cities. We have the features like name of the airline,source from which the service begins,destination where the service ends,route taken by the flight to reach the destination, time when the journey starts from the source,time of arrival at the destination, total duration of the flight,total stops between the source and destination. The price of the ticket is given for the paticular airlines and we have to predict the price of the flight as the target variable.

Exploratory Data Analysis

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

For this we require pandas to_datetime to convert object data type to datetime dtype.

.dt.day method will extract only day of that date\ .dt.month method will extract only month of that date

We have the date of journey with usual date format,we have to extract the format as day and month. Extracting date and month from data of journey and we will drop the Journey day

Handling Date and Time Variables:

We have ‘Date_of_Journey’, a ‘date type variable and ‘Dep_Time’, ‘Arrival_Time’ that captures time information.

We can extract ‘Journey_day’ and ‘Journey_Month’ from the ‘Date_of_Journey’ variable. ‘Journey day’ shows the day of the month on which the journey was started.

```
In [9]: df['Journey_day']=pd.to_datetime(df.Date_of_Journey,format='%d-%m-%Y').dt.day

In [10]: df['Journey_month']=pd.to_datetime(df['Date_of_Journey'],format='%d-%m-%Y').dt.month

In [11]: df.head()
```

Out[11]:

	Airline	Date_of_Journey	Source	Destination	Dep_Time	Arrival_Time	Duration	Total_Stops	Price	Journey_day	Journey_month
0	Air Asia	22-10-2021	New Delhi	Mumbai	08:00	14:35	6h 35m	1 Stop	5,953	22	10
1	Air Asia	22-10-2021	New Delhi	Mumbai	12:40	20:15	7h 35m	1 Stop	5,953	22	10
2	Air Asia	22-10-2021	New Delhi	Mumbai	11:55	20:15	8h 20m	1 Stop	5,953	22	10
3	Air Asia	22-10-2021	New Delhi	Mumbai	08:00	16:35	8h 35m	1 Stop	5,953	22	10
4	Air Asia	22-10-2021	New Delhi	Mumbai	04:55	14:15	9h 20m	1 Stop	5,953	22	10

we are dropping Date_of_Journey since we have converted them into Journey_day and Journey_month.

```
In [12]: df.drop(['Date_of_Journey'],axis=1,inplace=True)
```

We can extract the flight departure time in hours and minutes from the Departure time.

Similarly, we can extract ‘Departure_Hour’ and ‘Departure_Minute’ as well as ‘Arrival_Hour and ‘Arrival_Minute’ from ‘Dep_Time’ and ‘Arrival_Time’ variables respectively.

We can extract the flight departure time in hours and minutes from the Departure time.

In [13]:
```
# Extracting Hours
df["Dep_hour"] = pd.to_datetime(df["Dep_Time"]).dt.hour

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

#Dropping the Dep_Time since we have taken in min and hours.
df.drop(["Dep_Time"], axis = 1, inplace = True)
```

In [14]:
```
df.head()
```

Out[14]:

	Airline	Source	Destination	Arrival_Time	Duration	Total_Stops	Price	Journey_day	Journey_month	Dep_hour	Dep_min
0	Air Asia	New Delhi	Mumbai	14:35	6h 35m	1 Stop	5,953	22	10	8	0
1	Air Asia	New Delhi	Mumbai	20:15	7h 35m	1 Stop	5,953	22	10	12	40
2	Air Asia	New Delhi	Mumbai	20:15	8h 20m	1 Stop	5,953	22	10	11	55
3	Air Asia	New Delhi	Mumbai	16:35	8h 35m	1 Stop	5,953	22	10	8	0
4	Air Asia	New Delhi	Mumbai	14:15	9h 20m	1 Stop	5,953	22	10	4	55

In [15]:
```
#Getting arrival minutes and hours from Arrival time.

#In hours
df['Arrival_hour']=pd.to_datetime(df.Arrival_Time).dt.hour
#In minutes
df['Arrival_min']=pd.to_datetime(df.Arrival_Time).dt.minute
```

In [16]:
```
df
```

Out[16]:

	Airline	Source	Destination	Arrival_Time	Duration	Total_Stops	Price	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	Arrival_min
0	Air Asia	New Delhi	Mumbai	14:35	6h 35m	1 Stop	5,953	22	10	8	0	14	35
1	Air Asia	New Delhi	Mumbai	20:15	7h 35m	1 Stop	5,953	22	10	12	40	20	15
2	Air Asia	New Delhi	Mumbai	20:15	8h 20m	1 Stop	5,953	22	10	11	55	20	15

we are dropping Arrival_Time since we have converted them as Arrival_hour,and Arrival_min.

In [17]:
```
df.drop(['Arrival_Time'],axis=1,inplace=True)
```

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

Out[18]:

	Airline	Source	Destination	Duration	Total_Stops	Price	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	Arrival_min
0	Air Asia	New Delhi	Mumbai	6h 35m	1 Stop	5,953	22	10	8	0	14	35
1	Air Asia	New Delhi	Mumbai	7h 35m	1 Stop	5,953	22	10	12	40	20	15
2	Air Asia	New Delhi	Mumbai	8h 20m	1 Stop	5,953	22	10	11	55	20	15
3	Air Asia	New Delhi	Mumbai	8h 35m	1 Stop	5,953	22	10	8	0	16	35
4	Air Asia	New Delhi	Mumbai	9h 20m	1 Stop	5,953	22	10	4	55	14	15

Data Preprocessing

similarly we are splitting Duration and extracting as hours and minutes separately. so we are grouping duration column as list.duration is the time taken by the plane to reach the destination.duration is the difference between the arrival time and departure time.

In [19]:
```
len("2h 50m".split())
```

Out[19]: 2

We also have duration information on the ‘Duration’ variable. This variable contains both duration hours and minutes information combined.

We can extract ‘Duration_hours’ and ‘Duration_minutes’ separately from the ‘Duration’ variable.

In [20]:
```
duration=list(df['Duration'])
l=[]
for i in range(len(duration)):
    x=duration[i]
    if len(duration[i].split())!=2: #checking for duration is on hours or minutes
        if(x[-1]=="m") :
            l.append(int(x[:-1]))
        else:
            l.append(int(duration[i][:1])*60)
    else:
 #            print(duration[i])
             a = duration[i].split()
 #            print(a[1][:-1])
 #            print(int(a[1][:-1]))
             l.append(int(a[0][:-1])*60+int(a[1][:-1]))
```

#adding duration_hours,duration_mins to df.

#adding duration_hours,duration_mins to df.

In [21]:
```
lh=[]
lm=[]
for x in l :
    lh.append(x//60)
    lm.append(x%60)
```

In [22]:
```
df['Duration_hours'] = lh
df['Duration_mins'] = lm
```

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

Out[23]:

	Airline	Source	Destination	Duration	Total_Stops	Price	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	Arrival_min	Duration_hours	Dur
0	Air Asia	New Delhi	Mumbai	6h 35m	1 Stop	5,953	22	10	8	0	14	35	6	
1	Air Asia	New Delhi	Mumbai	7h 35m	1 Stop	5,953	22	10	12	40	20	15	7	

```
In [24]: df.drop(['Duration'],axis=1,inplace=True)

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

Out[25]:

	Airline	Source	Destination	Total_Stops	Price	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	Arrival_min	Duration_hours	Duration_mins
0	Air Asia	New Delhi	Mumbai	1 Stop	5,953	22	10	8	0	14	35	6	3E
1	Air Asia	New Delhi	Mumbai	1 Stop	5,953	22	10	12	40	20	15	7	3E
2	Air Asia	New Delhi	Mumbai	1 Stop	5,953	22	10	11	55	20	15	8	2C
3	Air Asia	New Delhi	Mumbai	1 Stop	5,953	22	10	8	0	16	35	8	3E
4	Air Asia	New Delhi	Mumbai	1 Stop	5,953	22	10	4	55	14	15	9	2C

Handling Categorical Data

Airline, Source, Destination, Total_Stops are the categorical variables we have in our data. Let’s handle each one by one.

Handling Categorical Data

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

Nominal data --> data are not in any order --> **OneHotEncoder** is used in this case,

Ordinal data --> data are in order --> **LabelEncoder** is used in this case

Categorical Values

```
In [26]: df['Airline'].value_counts()
```

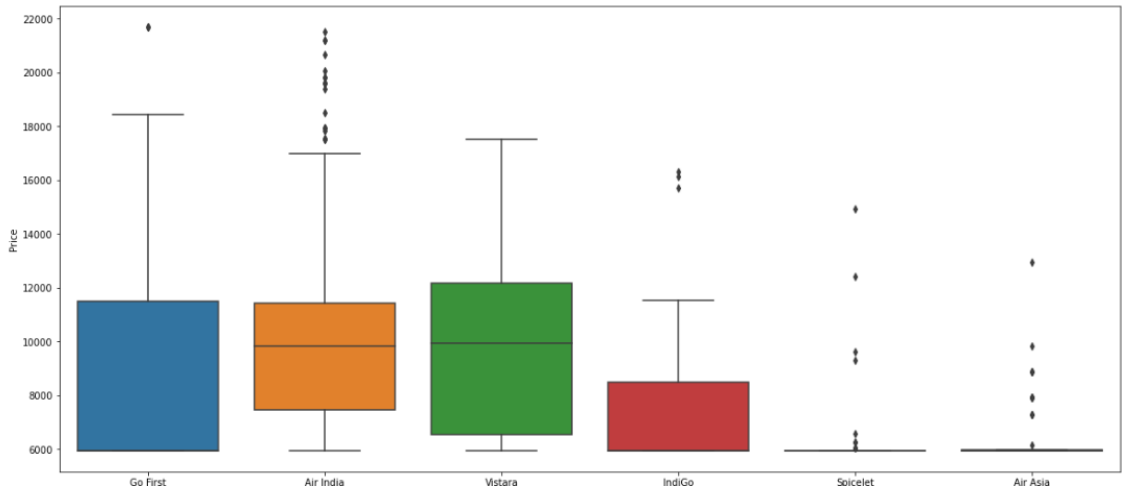
Out[26]:

```
Vistara      459
IndiGo       383
Air India    331
Go First     247
Air Asia     115
SpiceJet      84
Name: Airline, dtype: int64
```

```
In [27]: df['Price']=df['Price'].str.replace("," , "").astype('float')
```

```
In [28]: # jet airways business is having more range of difference compared to other airlines.

         #Airline vs Price
         fig = plt.figure(figsize =(20, 9))

         sns.boxplot(y='Price',x='Airline',data=df.sort_values('Price',ascending=False))
```

```
Out[28]: <AxesSubplot:xlabel='Airline', ylabel='Price'>
```



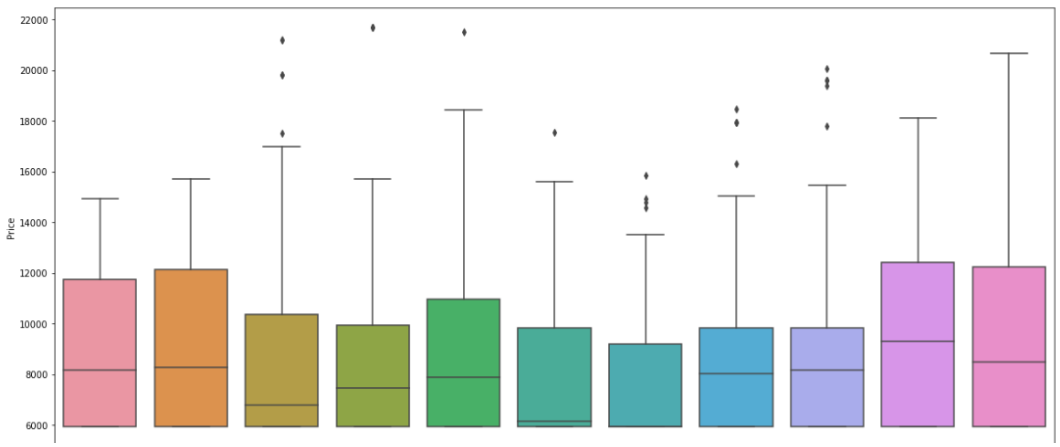
As we can see the name of the airline matters. ‘Vistara’ has the highest price range. Other airlines price also varies.

Since the Airline variable is Nominal Categorical Data (There is no order of any kind in airline names) we will use one-hot encoding to handle this variable.

Airline

In [29]:
```
# jet airways business is having more range of difference compared to other airlines.

#Airline vs Price
fig = plt.figure(figsize =(20,9))

sns.boxplot(x='Journey_day',y='Price',data=df.sort_values('Price',ascending=False))
```

Out[29]: <AxesSubplot:xlabel='Journey_day', ylabel='Price'>



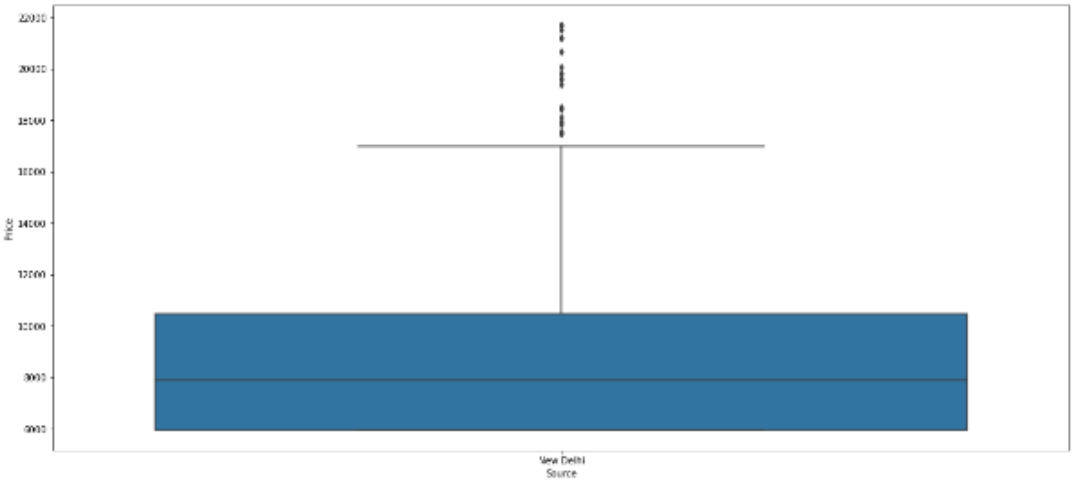
In [30]:
```
#Since airline is categorical data we will perform onehot encoding
Airline=df[['Airline']]
Airline=pd.get_dummies(Airline,drop_first=True)
Airline.head()
```

Out[30]:

	Airline_Air India	Airline_Go First	Airline_IndiGo	Airline_SpiceJet	Airline_Vistara
0	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0

One-Hot encoded ‘Airline’ data is saved in the Airline variable as shown in the above code.

```
In [32]: df['Source'].value_counts()

Out[32]: New Delhi    1619
         Name: Source, dtype: int64

In [33]: #Source Vs Price
         fig = plt.figure(figsize =(20, 9))

         sns.boxplot(y='Price',x='Source',data=df.sort_values('Price',ascending=False))

Out[33]: <AxesSubplot:xlabel='Source', ylabel='Price'>
```



Source and Destination Variable

Again ‘Source’ and ‘Destination’ variables are Nominal Categorical Data. We will use One-Hot encoding again to handle these two variables.

In [34]:
```
#source is nominal category we shall do one hot encoding
Source =df[['Source']]
Source=pd.get_dummies(Source)
Source.head()
```

Out[34]:

	Source_New Delhi
0	1
1	1
2	1
3	1
4	1

In [35]:
```
df['Destination'].value_counts()
```

Out[35]:
```
Mumbai    1619
Name: Destination, dtype: int64
```

In [36]:
```
#As destination is categorical data we will perform one hot encoding
Destination=df[["Destination"]]
Destination=pd.get_dummies(Destination)
Destination.head()
```

Out[36]:

	Destination_Mumbai
0	1
1	1
2	1
3	1
4	1

In [37]:
```
df['Total_Stops'].value_counts()
```

Out[37]:
```
1 Stop       1086
Non-stop      502
2 Stop(s)      29
3 Stop(s)       2
Name: Total_Stops, dtype: int64
```

Here, non-stop means 0 stops which means direct flight. Similarly meaning other values is obvious. We

can see it is an Ordinal Categorical Data so we will use LabelEncoder here to handle this variable.

```
In [38]: #Since this is  ordinal categorical type we perform label encoder
         #we are assigning values accordingly to the keys.
         df.replace({'Non-stop':0,'1 Stop':1,'2 Stop(s)':2,'3 Stop(s)':3,'4 Stop(s)':4},inplace=True)
```

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

Out[39]:

	Airline	Source	Destination	Total_Stops	Price	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	Arrival_min	Duration_hours	Duration_min
0	Air Asia	New Delhi	Mumbai	1	5953.0	22	10	8	0	14	35	6	3
1	Air Asia	New Delhi	Mumbai	1	5953.0	22	10	12	40	20	15	7	3
2	Air Asia	New Delhi	Mumbai	1	5953.0	22	10	11	55	20	15	8	2
3	Air Asia	New Delhi	Mumbai	1	5953.0	22	10	8	0	16	35	8	3
4	Air Asia	New Delhi	Mumbai	1	5953.0	22	10	4	55	14	15	9	2

Final Dataframe

Now we will create the final dataframe by concatenating all the One-hot and Label-encoded features to the original dataframe. We will also remove original variables using which we have prepared new encoded variables.

```
In [40]: df.dtypes
```

```
Out[40]: Airline           object
         Source            object
         Destination       object
         Total_Stops        int64
         Price            float64
         Journey_day        int64
         Journey_month      int64
         Dep_hour           int64
         Dep_min            int64
         Arrival_hour       int64
         Arrival_min        int64
         Duration_hours     int64
         Duration_mins      int64
         dtype: object
```

```
In [41]: # we are grouping to  dataframe the columns Airline,Source,Destination
         df=pd.concat([df,Airline,Source,Destination],axis=1)
```

```
In [42]: df.head()
```

Out[42]:

	Airline	Source	Destination	Total_Stops	Price	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	Arrival_min	Duration_hours	Duration_min
0	Air Asia	New Delhi	Mumbai	1	5953.0	22	10	8	0	14	35	6	
1	Air Asia	New Delhi	Mumbai	1	5953.0	22	10	12	40	20	15	7	
2	Air Asia	New Delhi	Mumbai	1	5953.0	22	10	11	55	20	15	8	
3	Air Asia	New Delhi	Mumbai	1	5953.0	22	10	8	0	16	35	8	
4	Air Asia	New Delhi	Mumbai	1	5953.0	22	10	4	55	14	15	9	

```
In [43]: df.drop(['Airline','Source','Destination'],axis=1,inplace=True)

In [44]: df.head()
```

```
In [44]: df.head()
```

Out[44]:

	Total_Stops	Price	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	Arrival_min	Duration_hours	Duration_mins	Airline_Air India	Airline_Go First	Ai
0	1	5953.0	22	10	8	0	14	35	6	35	0	0	
1	1	5953.0	22	10	12	40	20	15	7	35	0	0	
2	1	5953.0	22	10	11	55	20	15	8	20	0	0	
3	1	5953.0	22	10	8	0	16	35	8	35	0	0	
4	1	5953.0	22	10	4	55	14	15	9	20	0	0	

Let’s see the number of final variables we have in dataframe.

```
In [45]: df.shape
Out[45]: (1619, 17)

In [46]: df.columns
Out[46]: Index(['Total_Stops', 'Price', 'Journey_day', 'Journey_month', 'Dep_hour',
                'Dep_min', 'Arrival_hour', 'Arrival_min', 'Duration_hours',
                'Duration_mins', 'Airline_Air India', 'Airline_Go First',
                'Airline_IndiGo', 'Airline_SpiceJet', 'Airline_Vistara',
                'Source_New Delhi', 'Destination_Mumbai'],
               dtype='object')
```

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,

heatmap

feature_importance_

SelectKBest

```
In [51]: x=df.iloc[0:,2:]

In [55]: x=df.drop(['Price'],axis=1)

In [56]: x.head()
```

In [55]:
```
x=df.drop(['Price'],axis=1)
```

In [56]:
```
x.head()
```

Out[56]:

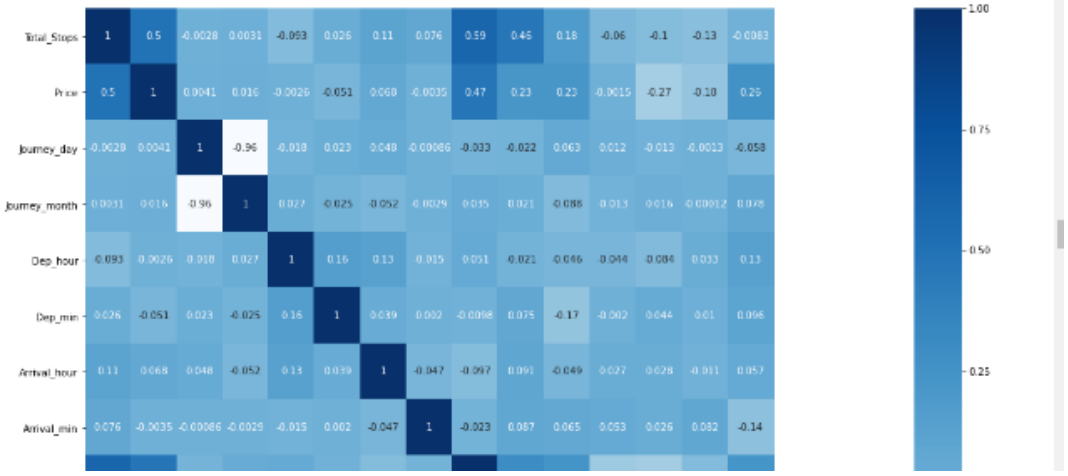
	Total_Stops	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	Arrival_min	Duration_hours	Duration_mins	Airline_Air India	Airline_Go First	Airline_Ind
0	1	22	10	8	0	14	35	6	35	0	0	
1	1	22	10	12	40	20	15	7	35	0	0	
2	1	22	10	11	55	20	15	8	20	0	0	
3	1	22	10	8	0	16	35	8	35	0	0	
4	1	22	10	4	55	14	15	9	20	0	0	

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

Out[57]:
```
0    5953.0
1    5953.0
2    5953.0
3    5953.0
4    5953.0
Name: Price, dtype: float64
```

Correlation

In [58]:
```
# correlation between Independent and dependent values.
plt.figure(figsize = (18,18))
sns.heatmap(df.corr(), annot = True, cmap = "Blues")
plt.show()
```



Model Building

Split dataset into train and test set in order to prediction w.r.t X_test.

If needed do scaling of data

Scaling is not done in Random forest

Import model

Fit the data

Predict w.r.t X_test

In regression check RSME Score

Plot graph

In [59]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import GridSearchCV
from sklearn.feature_selection import RFE
import statsmodels.api as sm
# for model evaluation
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
# for suppressing warnings
import warnings
warnings.filterwarnings("ignore")
```

In [68]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=0)
```

In [69]:
```
lr=LinearRegression(normalize=True)
lr.fit(x_train,y_train)
```

Out[69]:
```
LinearRegression(normalize=True)
```

In [70]:
```
lr_pred=lr.predict(x_test)
```

In [71]:
```
lr_pred
```

Out[71]:
```
array([ 6976.49243002, 10967.15900427, 11588.70012789, 10375.88657698,
       11320.61361189, 11266.61116216,  7711.43645031,  7698.45233164,
       10900.74316998, 10874.07966392,  9683.83150692,  7236.70198069,
       12399.73026665,  5582.44985724, 10681.93754043,  9635.73987587,
        8141.59814846, 11694.28404857,  8139.98066848, 10926.7242701 ,
        5327.6013816 ,  9819.54125536,  7197.45834198,  6609.28808527,
        7038.24307457, 10759.97544663,  7434.05206278, 10737.79988674,
        9906.80338195, 10405.12827813,  9944.03490583, 10687.96224065,
       10816.95989504,  5439.25337753, 11558.09363678,  7426.50509472,
        9627.39487263, 11411.6051887 ,  9388.64783537,  9810.1111007 ,
        8003.16314959, 10496.95401777,  9969.94507053,  9956.19860859,
        6347.08805785, 11250.90466388,  4660.76771877,  6831.78833277,
        9848.84731914,  5503.28984894,  5248.14785128,  6396.20325349,
        8017.7027761 ,  8820.18233053,  7133.18531492, 11378.00900549,
       12124.16066877,  9794.05989247,  8510.11031911,  2998.30131406,
       11325.28414766, 12040.85209554, 11075.1533242 ,  9355.62799328,
        8298.30068508, 10373.29014056,  7045.39989508,  8294.46570992,
       11899.51197709, 10461.53811762, 13841.15209988,  7975.51644984,
        5759.5275553 ,  6523.00805504,  6687.69623648,  8759.1870338 ,
        4914.62547176,  7677.55081199, 11043.35872964, 11170.77442597,
```

In [72]:
```
lr_accuracy=round(lr.score(x_train,y_train)*100)
lr_accuracy
```

Out[72]:
```
45
```

In [73]:
```
from sklearn.metrics import mean_absolute_error
mean_absolute_error(y_test, lr_pred)
```

Out[73]:
```
1731.5628951197891
```

In [74]:
```
print("RMSE VALUE = ",mean_squared_error(y_test, lr_pred,squared = False))
```

```
RMSE VALUE =  2322.357397767956
```

In [75]:
```
import numpy as np
o = np.array(y_test)
```

In [76]:
```
print("original price  is " ,o[0])
print("predicted average price is " , lr_pred[0])
```

```
original price  is  8790.0
predicted average price is  6976.492430015496
```

In [77]:
```
# using ExtraTreesRegressor

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

Out[77]:
```
ExtraTreesRegressor()
```

In [78]:
```
print(selection.feature_importances_)
```

```
[0.15740734 0.17655217 0.01140203 0.05207631 0.04985157 0.05306593
 0.05387805 0.16755495 0.04505817 0.0703772  0.05315068 0.03710376
 0.00473619 0.06778565 0.         0.        ]
```

XGBRegressor

In [102]:
```
from xgboost import XGBRegressor
model =  XGBRegressor()
model.fit(x_train,y_train)
```

Out[102]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.300000012, max_delta_step=0, max_depth=6,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=100, n_jobs=4, num_parallel_tree=1, random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
```

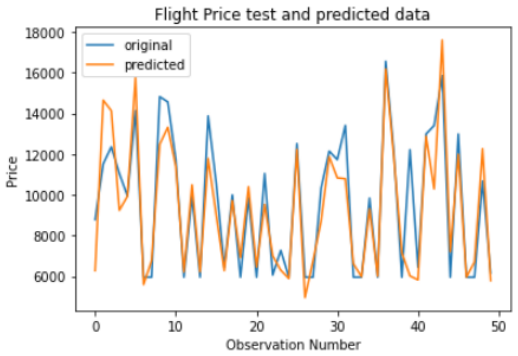
In [104]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.300000012, max_delta_step=0, max_depth=6,
             min_child_weight=1, monotone_constraints='()',
             n_estimators=100, n_jobs=0, num_parallel_tree=1, random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
```

Out[104]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.300000012, max_delta_step=0, max_depth=6,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=100, n_jobs=0, num_parallel_tree=1, random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
```

In [105]:
```
y_pred =  model.predict(x_test)
print('Training Score :',model.score(x_train, y_train))
print('Test Score     :',model.score(x_test, y_test))
```

```
Training Score : 0.9951869440807274
Test Score     : 0.6992801591514202
```

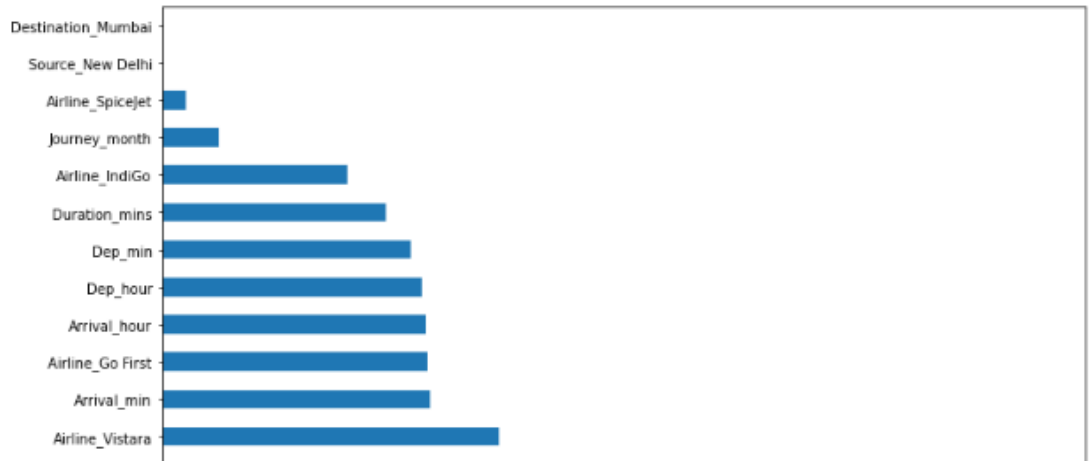
In [106]:
```
number_of_observations=50

x_ax = range(len(y_test[:number_of_observations]))

plt.plot(x_ax, y_test[:number_of_observations], label="original")

plt.plot(x_ax, y_pred[:number_of_observations], label="predicted")

plt.title("Flight Price test and predicted data")

plt.xlabel('Observation Number')

plt.ylabel('Price')

plt.legend()

plt.show()
```



As we can observe in the above figure, model predictions and original prices are overlapping. This visual result confirms the high model score .

Data Visualisation

[79]:
```
# visualization

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



Random Forest Regressor

In [80]:
```
from sklearn.ensemble import RandomForestRegressor
reg_rf = RandomForestRegressor()
reg_rf.fit(x_train, y_train)
```

Out[80]:
```
RandomForestRegressor()
```

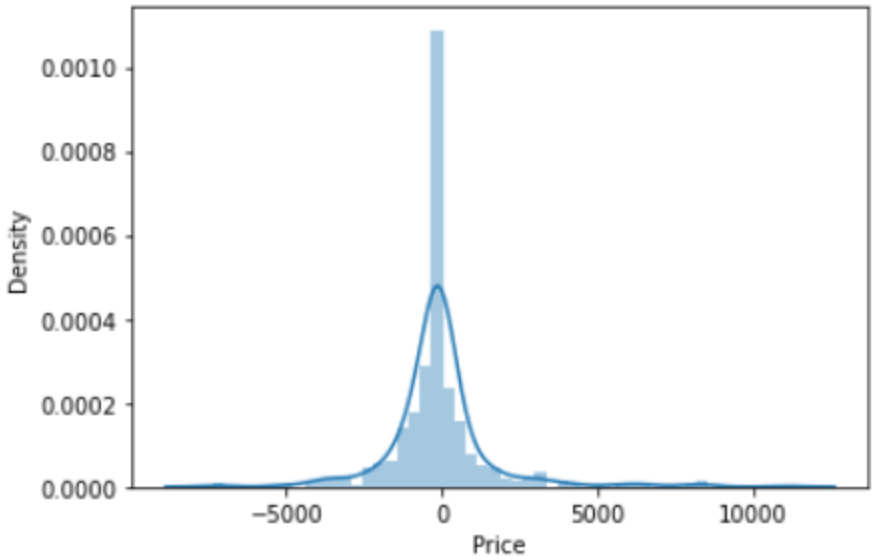
In [81]:
```
y_pred = reg_rf.predict(x_test)
```

In [82]:
```
reg_rf.score(x_train, y_train)
```

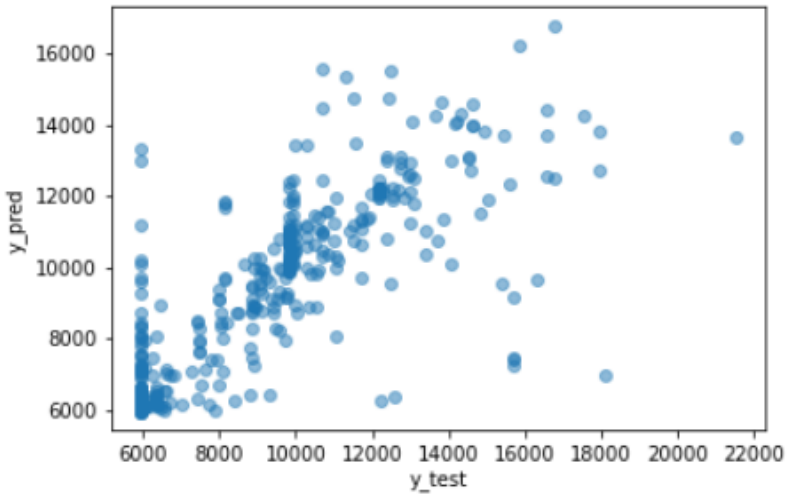
Out[82]:
```
0.9621161936558771
```

In [83]:
```
sns.distplot(y_test-y_pred)
plt.show()
```

In [83]:
```
sns.distplot(y_test-y_pred)
plt.show()
```



In [84]:
```
plt.scatter(y_test, y_pred, alpha = 0.5)
plt.xlabel("y_test")
plt.ylabel("y_pred")
plt.show()
```



In [85]:
```
from sklearn import metrics
```

In [86]:
```
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: 920.3339218106996
MSE: 2925711.3723442755
RMSE: 1710.4710966117714
```

In [87]:
```
# RMSE/(max(DV)-min(DV))

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

Out[87]:
```
0.13285148068124047
```

In [88]:
```
metrics.r2_score(y_test, y_pred)
```

Out[88]:
```
0.682963885601845
```

KNeighborsRegressor,SVR,DecisionTreeRegressor,RandomForestRegressor

]:
```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor

#for RMSLE we will create  own scorer
from sklearn.metrics import make_scorer
```

In [97]:
```
def score(y_pred,y):
    y_pred = np.log(y_pred)
    y = np.log(y)
    return 1 - ((np.sum((y_pred-y)**2))/len(y))**1/2    # 1-RMSLE

# make  own scorer
scorer = make_scorer(score,greater_is_better=True, needs_proba=False)
```

In [98]:
```
knn_reg = KNeighborsRegressor()
svm_reg = SVR(gamma='scale')
dt_reg = DecisionTreeRegressor()
rf_reg = RandomForestRegressor()
```

In [100]:
```
#Training,Testing
for reg in (knn_reg, svm_reg, dt_reg, rf_reg):
    reg.fit(x_train, y_train)

    y_pred = reg.predict(x_test)

    print(reg, score(y_pred,y_test))
```

```
KNeighborsRegressor() 0.9680692208598336
SVR() 0.9452027111266905
DecisionTreeRegressor() 0.9797783487389925
RandomForestRegressor() 0.9858572125190296
```

Hyperparameter Tuning

Choose following method for hyperparameter tuning.

RandomizedSearchCV

GridSearchCV

Assign hyperparameters in form of dictionary

Fit the model

Check best paramters and best score

]:
```
from sklearn.model_selection import RandomizedSearchCV
```

]:
```
#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']
```

In [89]:
```
from sklearn.model_selection import RandomizedSearchCV
```

In [90]:
```
#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 [91]:
```
# 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 [92]:
```
# Random search of parameters, using 5 fold cross validation,
# search across 100 different combinations
rf_random = RandomizedSearchCV(estimator = reg_rf, param_distributions = random_grid,scoring='neg_mean_squared_error', n_iter = 1
rf_random.fit(x_train,y_train)
```

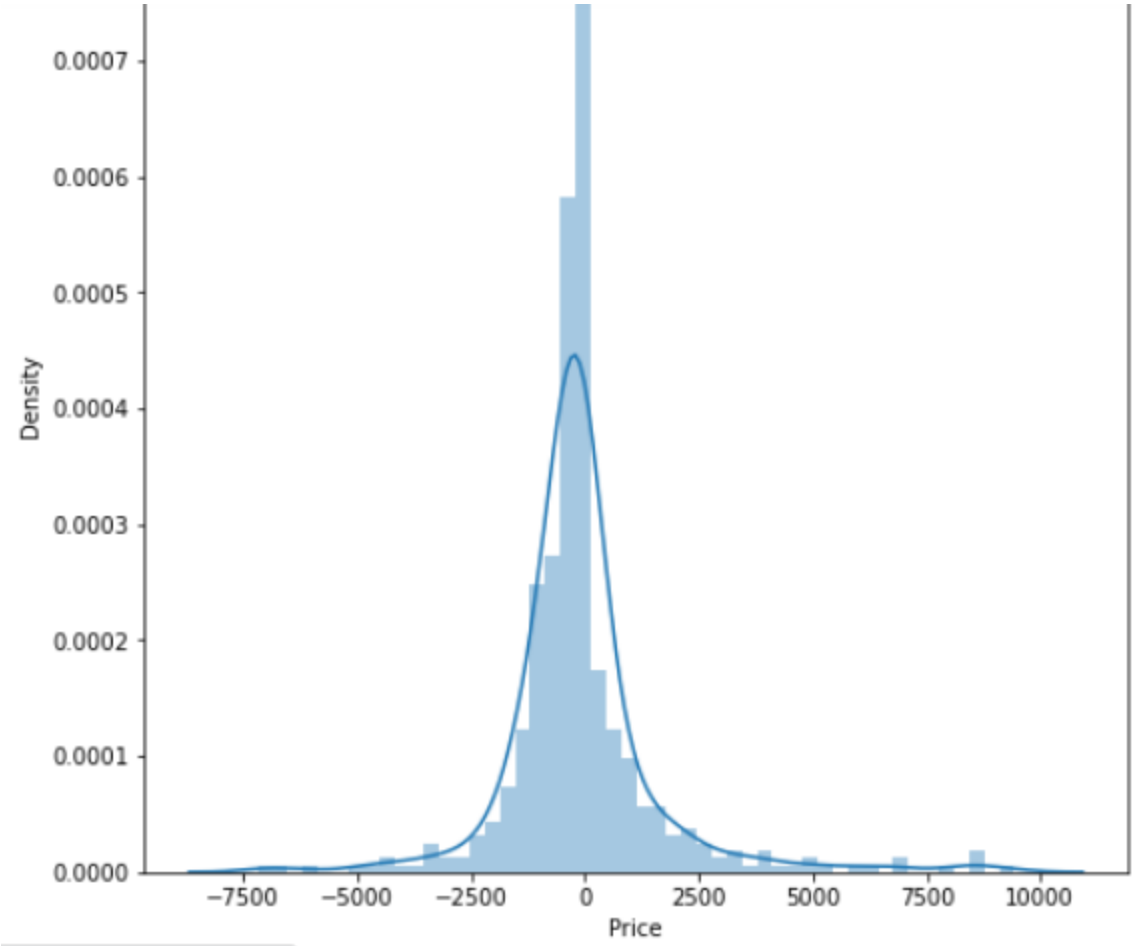
```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[CV] n_estimators=900, min_samples_split=5, min_samples_leaf=5, max_features=sqrt, max_depth=10
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[CV]  n_estimators=900, min_samples_split=5, min_samples_leaf=5, max_features=sqrt, max_depth=10, total=   2.4s
[CV] n_estimators=900, min_samples_split=5, min_samples_leaf=5, max_features=sqrt, max_depth=10
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    2.3s remaining:    0.0s
[CV]  n_estimators=900, min_samples_split=5, min_samples_leaf=5, max_features=sqrt, max_depth=10, total=   2.2s
[CV] n_estimators=900, min_samples_split=5, min_samples_leaf=5, max_features=sqrt, max_depth=10
[CV]  n_estimators=900, min_samples_split=5, min_samples_leaf=5, max_features=sqrt, max_depth=10, total=   2.3s
[CV] n_estimators=900, min_samples_split=5, min_samples_leaf=5, max_features=sqrt, max_depth=10
[CV]  n_estimators=900, min_samples_split=5, min_samples_leaf=5, max_features=sqrt, max_depth=10, total=   2.3s
[CV] n_estimators=900, min_samples_split=5, min_samples_leaf=5, max_features=sqrt, max_depth=10
[CV]  n_estimators=900, min_samples_split=5, min_samples_leaf=5, max_features=sqrt, max_depth=10, total=   2.5s
[CV] n_estimators=1100, min_samples_split=10, min_samples_leaf=2, max_features=sqrt, max_depth=15
[CV]  n_estimators=1100, min_samples_split=10, min_samples_leaf=2, max_features=sqrt, max_depth=15, total=   2.7s
[CV] n_estimators=1100, min_samples_split=10, min_samples_leaf=2, max_features=sqrt, max_depth=15
```

In [93]:
```
rf_random.best_params_
```

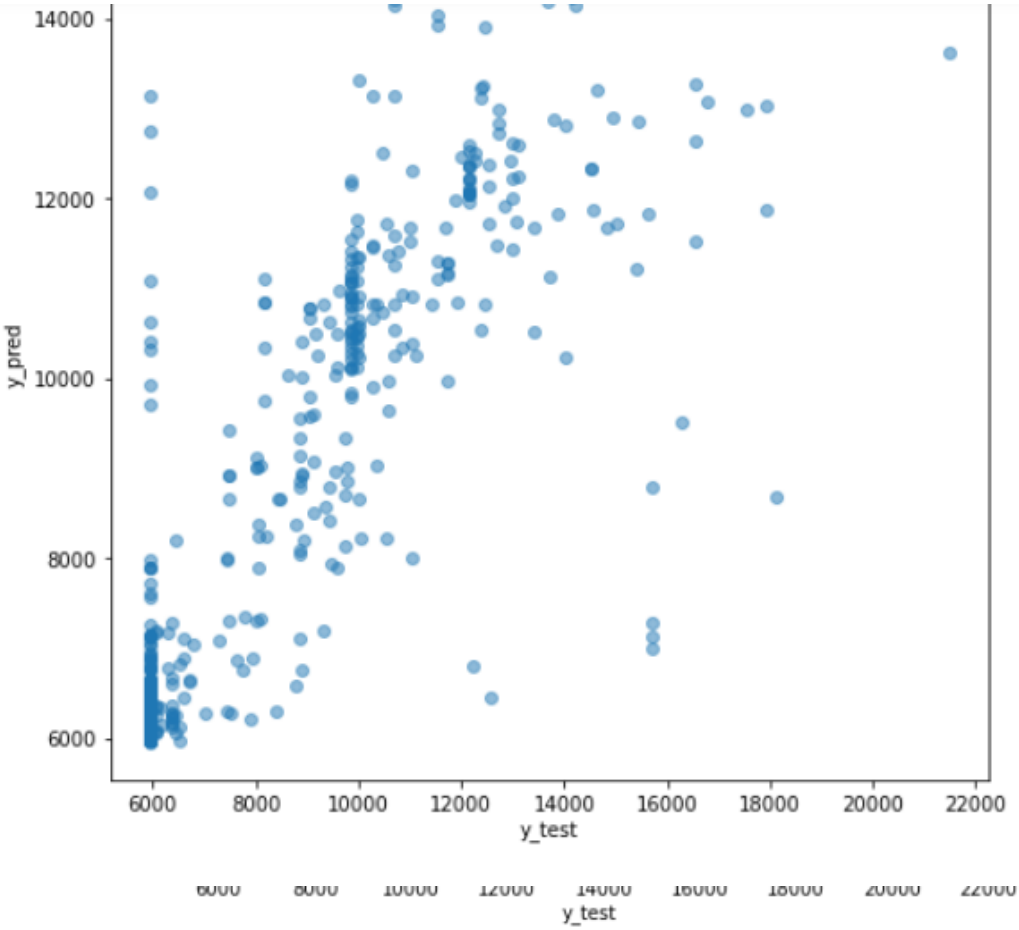
Out[93]:
```
{'n_estimators': 700,
 'min_samples_split': 15,
 'min_samples_leaf': 1,
 'max_features': 'auto',
 'max_depth': 20}
```

In [94]:
```
prediction = rf_random.predict(x_test)
```

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



In [96]:
```
plt.figure(figsize = (8,8))
plt.scatter(y_test, prediction, alpha = 0.5)
plt.xlabel("y_test")
plt.ylabel("y_pred")
plt.show()
```



In [97]:
```
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: 984.3028188174584
MSE: 2966117.767030634
RMSE: 1722.2420756184754
```

Conclusion

In [98]:
```
import pickle
# open a file, where you ant to store the data
file = open('flight_rf.pkl', 'wb')

# dump information to that file
pickle.dump(rf_random, file)
```

In [99]:
```
model = open('flight_rf.pkl','rb')
forest = pickle.load(model)
```

In [100]:
```
y_prediction = forest.predict(x_test)
```

In [101]:
```
metrics.r2_score(y_test, y_prediction)
```

Out[101]: 0.6785853653932925

Interpretation of the Results Summary:

First the Flight dataset is read and analyzed reading the features we analyze ,Price is the target column here. All the features are then analyzed, missing data handling, outlier detection, data cleaning are done. Trend of Price is observed for change in individual features. New features are extracted, redundant features dropped and categorical features are encoded accordingly.
I take a look at the first few rows using the data dictionary provided.I get a sense of what each column represents,I identify the predictors and response variables.I check if there is any unique identifier for each column.
made a note of all the observations as a part of data understanding,in the data there are 1619 variables out of

which price is the responsible variable and all the predictors.
Name of the Airlines and Journey Date is the unique identifier for each record as a part of data preparation. I start with data cleaning,i start with null values and data types of each column,i bring all the string values together.
Creating dummy variables increases the number of features greatly, highly imbalanced columns are dropped.
created dummy variables to convert categorical variables to numeric variables.
We need this because linear regression can only be done with numeric variables.
Before modelling, we divided the model into training and testing data.
70% random samples for training data and remaining 30% for testing data.
I began data modelling by creating the first model as Linear regression with all the variables of training data and noted the results of the first model.
Then calculated the Original price and Predicted price from the values observed.

Plotted the graph of feature importances for better visualization and the residual of the final to ensure that normal distribution.

Using the Regressor models calculated the KNeighborsRegressor,SVR,DecisionTreeRegressor,RandomForestRegressor, and observed that the RandomForestRegressor is performing the good precision and calculated the Hyper parameter tuning for RandomizedSearchCV.Finally calculated the rf_random best score.

Plotted the distplot and scatter plot of Y test predictions,and saved the model.