Flight Ticket Price Precition

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
    Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_
```

```
df=pd.read_excel('/content/drive/MyDrive/Luminar/Data_Train.xlsx')
df
```

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additi
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 stops	
					DEL					

2	Jet Airways	9/06/2019	Delhi	Cochin	→ LKO → BOM → COK	09:25	04:25 10 Jun	19h	2 stops
3	IndiGo	12/05/2019	Kolkata	Banglore	CCU → NAG →	18:05	23:30	5h 25m	1 stop

```
df.head()
```

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	non-stop	N
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	2 stops	N

```
df.tail()
```

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additi
10678	Air Asia	9/04/2019	Kolkata	Banglore	CCU → BLR	19:55	22:25	2h 30m	non-stop	
10679	Air India	27/04/2019	Kolkata	Banglore	CCU → BLR	20:45	23:20	2h 35m	non-stop	
	Jet				BLR					

10680	Jet	27/04/2019	Banglore	Delhi	→	08:20	11:20	3h	non-stop

```
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
```

```
df.shape
```

```
(10683, 11)
```

```
df.info()  #price is dependent features and others are indipendent features
```

```
<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
```

Checking for null values

```
df.isna().sum()
```

```
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
```

Here we are having only 2 null values in Route and Total stops so we are dropping it

```
df.dropna(inplace=True) #drop null values
```

```
df.isna().sum()
```

```
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
```

```
df.shape
```

```
(10682, 11)
```

EDA/FEATURE ENGINEERING/PREPROCESSING STEPS - is done to make data more informative

```
df['Date_of_Journey'].value_counts
```

```
<bound method IndexOpsMixin.value_counts of 0        24/03/2019
1         1/05/2019
2         9/06/2019
3        12/05/2019
4        01/03/2019
            ...
10678     9/04/2019
10679    27/04/2019
10680    27/04/2019
10681    01/03/2019
10682     9/05/2019
Name: Date_of_Journey, Length: 10682, dtype: object>
```

Extracting Date and Month From Date of Journey columns

Converting into Datetime

we are splitting the jounery date into day and month using the function to_datetime in pandas

dt.date will get the date dt.month will get the date

```
df['Journey_day'] = pd.to_datetime(df["Date_of_Journey"], format = "%d/%m/%Y").dt.day  #dt.day is used to get the date

df['Journey_month'] = pd.to_datetime(df["Date_of_Journey"],format ='%d/%m/%Y').dt.month #dt.month is used to get the mo
```

```
df.head()
```

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	non-stop	N
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	2 stops	N

```
#dropping date_of _journey as we have already gathered the day and month

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

```
df.head()
```

Airline	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price	Jour

	Airline	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price	Jour
0	IndiGo	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	non-stop	No info	3897	
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	2 stops	No info	7662	

```
#Converting the Departure Time in Hours and Minutes

df['Dep_hour']=pd.to_datetime(df['Dep_Time']).dt.hour       #.dt.hour is used to get hours
df['Dep_Min']=pd.to_datetime(df['Dep_Time']).dt.minute      #dt.minute is used to get mins
```

```
df.head()
```

	Airline	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price	Jour
0	IndiGo	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	non-stop	No info	3897	
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	2 stops	No info	7662	

```
#dropping Dep Time as it is also converted to Hours and Minutes
```

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

```
df.head()
```

	Airline	Source	Destination	Route	Arrival_Time	Duration	Total_Stops	Additional_Info	Price	Journey_day	J
0	IndiGo	Banglore	New Delhi	BLR → DEL	01:10 22 Mar	2h 50m	non-stop	No info	3897	24	
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	13:15	7h 25m	2 stops	No info	7662	1	

```
#similarly Arrival Time is also converted into Hours and Minutes

df['Arrival_hour']=pd.to_datetime(df['Arrival_Time']).dt.hour
df['Arrival_minutes']=pd.to_datetime(df['Arrival_Time']).dt.minute
```

```
df.head()
```

	Airline	Source	Destination	Route	Arrival_Time	Duration	Total_Stops	Additional_Info	Price	Journey_day	J
0	IndiGo	Banglore	New Delhi	BLR → DEL	01:10 22 Mar	2h 50m	non-stop	No info	3897	24	
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	13:15	7h 25m	2 stops	No info	7662	1	

```
#dropping arrival time as we have  already extracted it to hours and minutes

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

```
df.head()
```

	Airline	Source	Destination	Route	Duration	Total_Stops	Additional_Info	Price	Journey_day	Journey_month
0	IndiGo	Banglore	New Delhi	BLR → DEL	2h 50m	non-stop	No info	3897	24	3
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	7h 25m	2 stops	No info	7662	1	5

From the Duration column we are extracting hours and minutes

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

```
2h 50m     550
1h 30m     386
2h 45m     337
2h 55m     337
2h 35m     329
          ...
31h 30m      1
30h 25m      1
42h 5m       1
4h 10m       1
47h 40m      1
Name: Duration, Length: 368, dtype: int64
```

```
len("4h 45m".split())
```

```
2
```

```
#time taken by plane to reach destination is called duration
#it is defference between departure time and arrival time

duration=list(df["Duration"])
for i in range(len(duration)):
```

```
  if len(duration[i].split())!=2:  #check duration contains only hours or mins
    if "h" in duration[i]:
      duration[i]=duration[i].strip()+" 0m"  #Adds 0 minutes
    else:
      duration[i]="0h "+duration[i]
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 only
```

As done earlier here too we are Adding "duration_hours" and "duration_mins" list to data frame and droping the columns "duration" from it

```
df['Duration_hours']=duration_hours
df['Duration_mins']=duration_mins
```

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

```
df.head()
```

	Airline	Source	Destination	Route	Total_Stops	Additional_Info	Price	Journey_day	Journey_month	Dep_hour
0	IndiGo	Banglore	New Delhi	BLR → DEL	non-stop	No info	3897	24	3	22
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	2 stops	No info	7662	1	5	5

Handling Categorical Data

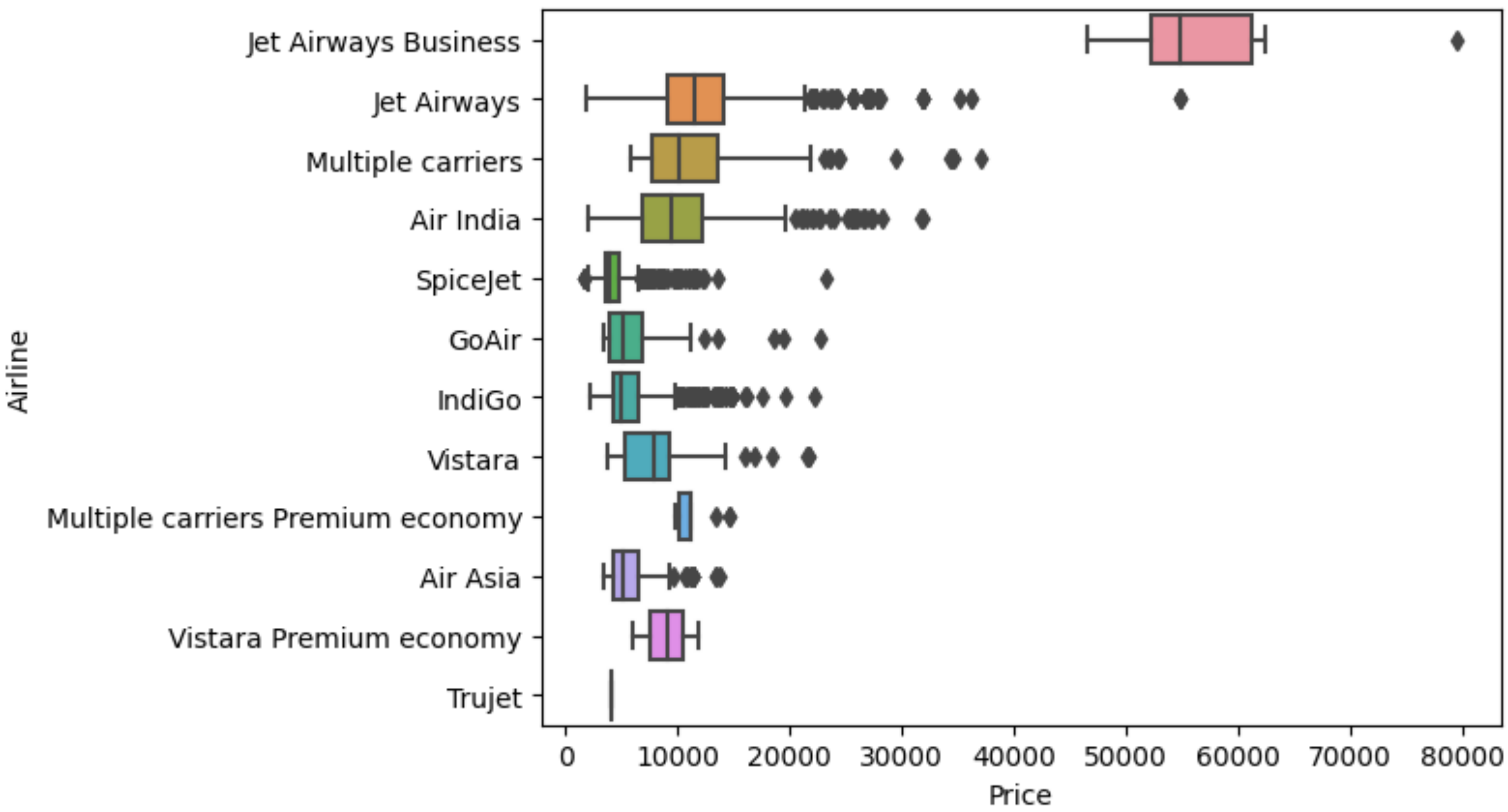
Handling categorical data means converting object data types into numerical values using techniques like Label Encoding,One Hot Encoding,..etc

```
df['Airline'].value_counts()
```

```
    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
```

```
Multiple carriers Premium economy      13
Jet Airways Business                    6
Vistara Premium economy                 3
Trujet                                  1
Name: Airline, dtype: int64
```

```
#Airline Vs Price
sns.boxplot(x = 'Price' , y = 'Airline' ,data=df.sort_values("Price",ascending=False))
plt.show()
```



Here we will be changing the ‘Airline’ feature into integer format using one hot encoding

```
Airline=pd.get_dummies(df['Airline'],drop_first=True)
Airline.head()
```

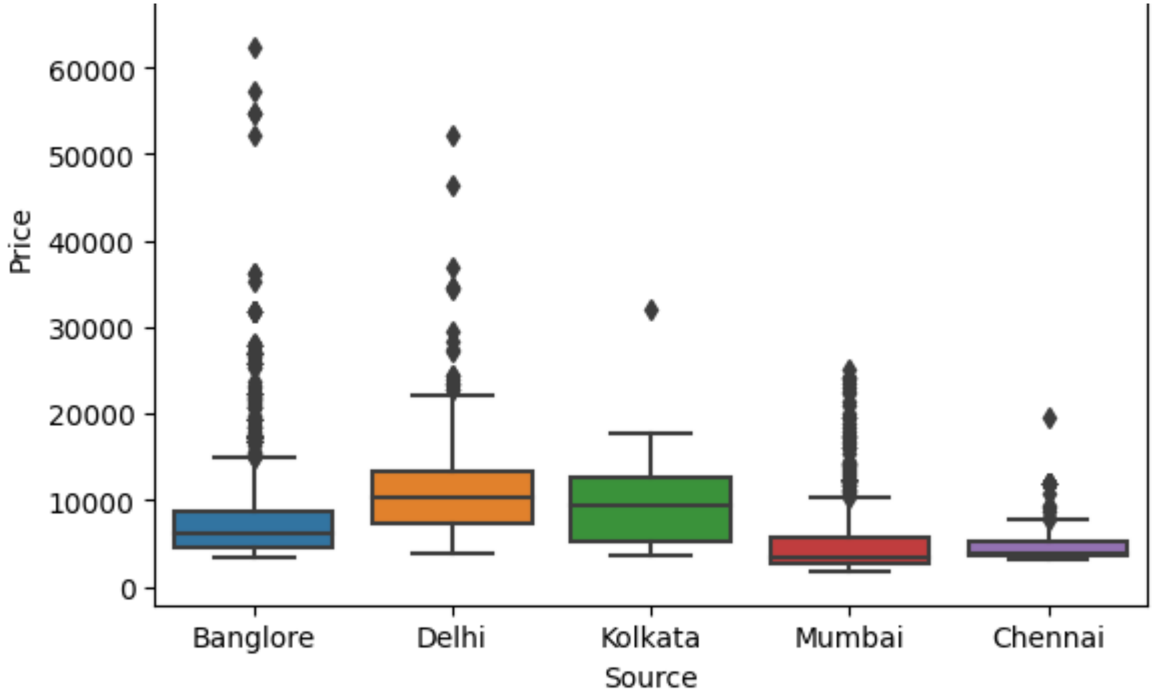
	Air India	GoAir	IndiGo	Jet Airways	Jet Airways Business	Multiple carriers	Multiple carriers Premium economy	SpiceJet	Trujet	Vistara	Vistara Premium economy
0	0	0	1	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	1	0	0	0	0	0	0	0
3	0	0	1	0	0	0	0	0	0	0	0

We will apply the get_dummies function on the 'Source' and 'Destination' column as well.

```
df['Source'].value_counts()
```

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

```
#source vs price
sns.boxplot(x='Source',y='Price',data=df.sort_values("Price",ascending=False))
plt.show()
```




```
#source is nominal data,so we use OneHotEncoding
Source=pd.get_dummies(df[['Source']],drop_first=True)
Source.head()
```

	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

```
df['Destination'].value_counts()
```

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

```
#destination is also nominal data
Destination=pd.get_dummies(df[['Destination']],drop_first=True)
Destination.head()
```

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

```
df['Route'].value_counts()
```

```
DEL → BOM → COK          2376
BLR → DEL                1552
CCU → BOM → BLR           979
CCU → BLR                 724
BOM → HYD                 621
                         ...
CCU → VTZ → BLR             1
CCU → IXZ → MAA → BLR       1
BOM → COK → MAA → HYD       1
BOM → CCU → HYD             1
BOM → BBI → HYD             1
```

```
Name: Route, Length: 128, dtype: int64
```

```
df['Additional_Info'].value_counts()
```

```
No info                         8344
In-flight meal not included     1982
No check-in baggage included     320
1 Long layover                    19
Change airports                    7
Business class                     4
No Info                            3
1 Short layover                    1
Red-eye flight                     1
2 Long layover                     1
Name: Additional_Info, dtype: int64
```

```
#route and total_stops are related to each other and 80% of Additional info contains no_info
#Route and total stops are related to each other
#so we drop these two features
df.drop(['Route','Additional_Info'],axis=1,inplace=True)
```

```
df.head()
```

	Airline	Source	Destination	Total_Stops	Price	Journey_day	Journey_month	Dep_hour	Dep_Min	Arrival_hour	A
0	IndiGo	Banglore	New Delhi	non-stop	3897	24	3	22	20	1	
1	Air India	Kolkata	Banglore	2 stops	7662	1	5	5	50	13	
2	Jet Airways	Delhi	Cochin	2 stops	13882	9	6	9	25	4	
3	IndiGo	Kolkata	Banglore	1 stop	6218	12	5	18	5	23	
4	IndiGo	Banglore	New Delhi	1 stop	13302	1	3	16	50	21	

```
df['Total_Stops'].value_counts()
```

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

Total stops have 5unique value and it is Ordinal categorical type we perform Label Encoder (total stops increasess so price will be increases)

```
df.replace({'non-stop':0,'1 stop':1,'2 stops':2,'3 stops':3,'4 stops':4},inplace=True)
df
```

	Airline	Source	Destination	Total_Stops	Price	Journey_day	Journey_month	Dep_hour	Dep_Min	Arrival_ho
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	
...	...	...	...	...	...	...	...	...	...	
10678	Air Asia	Kolkata	Banglore	0	4107	9	4	19	55	
10679	Air India	Kolkata	Banglore	0	4145	27	4	20	45	
10680	Jet Airways	Banglore	Delhi	0	7229	27	4	8	20	
10681	Vistara	Banglore	New Delhi	0	12648	1	3	11	30	

10682	Air India	Delhi	Cochin	2	11753	9	5	10	55	

```
#concatenate dataframe by adding df,airline,source,destination
df_train=pd.concat([df,Airline,Source,Destination],axis=1)
df_train.head()
```

	Airline	Source	Destination	Total_Stops	Price	Journey_day	Journey_month	Dep_hour	Dep_Min	Arrival_hour	.
0	IndiGo	Banglore	New Delhi	0	3897	24	3	22	20	1	
1	Air India	Kolkata	Banglore	2	7662	1	5	5	50	13	
2	Jet Airways	Delhi	Cochin	2	13882	9	6	9	25	4	
3	IndiGo	Kolkata	Banglore	1	6218	12	5	18	5	23	
4	IndiGo	Banglore	New Delhi	1	13302	1	3	16	50	21	

5 rows × 33 columns

```
df_train.drop(['Airline','Source','Destination'],axis=1,inplace=True)
df_train.head()
```

	Total_Stops	Price	Journey_day	Journey_month	Dep_hour	Dep_Min	Arrival_hour	Arrival_minutes	Duration_hours
0	0	3897	24	3	22	20	1	10	2
1	2	7662	1	5	5	50	13	15	7
2	2	13882	9	6	9	25	4	25	19

3	1	6218	12	5	18	5	23	30	5
4	1	13302	1	3	16	50	21	35	4

5 rows × 30 columns

```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10682 entries, 0 to 10682
Data columns (total 30 columns):
 #   Column                             Non-Null Count  Dtype
---  ------                             --------------  -----
 0   Total_Stops                        10682 non-null  int64
 1   Price                              10682 non-null  int64
 2   Journey_day                        10682 non-null  int64
 3   Journey_month                      10682 non-null  int64
 4   Dep_hour                           10682 non-null  int64
 5   Dep_Min                            10682 non-null  int64
 6   Arrival_hour                       10682 non-null  int64
 7   Arrival_minutes                    10682 non-null  int64
 8   Duration_hours                     10682 non-null  int64
 9   Duration_mins                      10682 non-null  int64
 10  Air India                          10682 non-null  uint8
 11  GoAir                              10682 non-null  uint8
 12  IndiGo                             10682 non-null  uint8
 13  Jet Airways                        10682 non-null  uint8
 14  Jet Airways Business               10682 non-null  uint8
 15  Multiple carriers                  10682 non-null  uint8
 16  Multiple carriers Premium economy  10682 non-null  uint8
 17  SpiceJet                           10682 non-null  uint8
 18  Trujet                             10682 non-null  uint8
 19  Vistara                            10682 non-null  uint8
 20  Vistara Premium economy            10682 non-null  uint8
 21  Source_Chennai                     10682 non-null  uint8
 22  Source_Delhi                       10682 non-null  uint8
 23  Source_Kolkata                     10682 non-null  uint8
 24  Source_Mumbai                      10682 non-null  uint8
```

```
 25  Destination_Cochin                 10682 non-null  uint8
 26  Destination_Delhi                  10682 non-null  uint8
 27  Destination_Hyderabad              10682 non-null  uint8
 28  Destination_Kolkata                10682 non-null  uint8
 29  Destination_New Delhi              10682 non-null  uint8
dtypes: int64(10), uint8(20)
memory usage: 1.1 MB
```

```
df_train.shape
```

```
(10682, 30)
```

Test Data

in this project we are preprocessing training and testing data supperately.bacause data leakage

```
ds=pd.read_excel('/content/drive/MyDrive/Luminar/Test_set.xlsx')
ds
```

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additio
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	21/05/2019	Delhi	Cochin	DEL → BOM	19:15	19:00 22 May	23h 45m	1 stop	In-flight

	Airways				→ COK				
3	Multiple carriers	21/05/2019	Delhi	Cochin	DEL → BOM → COK	08:00	21:00	13h	1 stop

```
ds.head()
```

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_
0	Jet Airways	6/06/2019	Delhi	Cochin	DEL → BOM → COK	17:30	04:25 07 Jun	10h 55m	1 stop	No
1	IndiGo	12/05/2019	Kolkata	Banglore	CCU → MAA	06:20	10:20	4h	1 stop	No

```
ds.tail()
```

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additiona
2666	Air India	6/06/2019	Kolkata	Banglore	CCU → DEL → BLR	20:30	20:25 07 Jun	23h 55m	1 stop	
2667	IndiGo	27/03/2019	Kolkata	Banglore	CCU → BLR	14:20	16:55	2h 35m	non-stop	

```
ds.shape
```

```
(2671, 10)
```

```
#preprocessing
ds.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
```

```
ds.isna().sum()
```

```
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
```

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



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


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



# Duration
duration = list(ds["Duration"])
for i in range(len(duration)):
    if len(duration[i].split()) != 2:
        if "h" in duration[i]:
            duration[i] = duration[i].strip() + " 0m"
        else:
            duration[i] = "0h " + duration[i]
duration_hours = []
duration_mins = []
for i in range(len(duration)):
    duration_hours.append(int(duration[i].split(sep = "h")[0]))
    duration_mins.append(int(duration[i].split(sep = "m")[0].split()[-1]))


# Adding Duration column to test set
ds["Duration_hours"] = duration_hours
ds["Duration_mins"] = duration_mins
```

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

```
# Categorical data
ds["Airline"].value_counts()
```

```
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
```

```
Airline = pd.get_dummies(ds["Airline"], drop_first= True)
Airline.head()
```

	Air India	GoAir	IndiGo	Jet Airways	Jet Airways Business	Multiple carriers	Multiple carriers Premium economy	SpiceJet	Vistara	Vistara Premium economy
0	0	0	0	1	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0
2	0	0	0	1	0	0	0	0	0	0
3	0	0	0	0	0	1	0	0	0	0

```
ds['Source'].value_counts()
```

```
Delhi      1145
Kolkata     710
```

```
Banglore     555
Mumbai       186
Chennai       75
Name: Source, dtype: int64
```

```
Source=pd.get_dummies(ds["Source"],drop_first=True)
Source.head()
```

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

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

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

```
Destination = pd.get_dummies(ds["Destination"], drop_first = True)
Destination.head()
```

	Cochin	Delhi	Hyderabad	Kolkata	New Delhi
0	1	0	0	0	0

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

```
#Route and Total_Stops are related to each other and 80% of additional info contains no_info
# So we drop these two features
ds.drop(["Route", "Additional_Info"], axis = 1, inplace = True)
```

```
ds
```

	Airline	Source	Destination	Total_Stops	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	Arri
0	Jet Airways	Delhi	Cochin	1 stop	6	6	17	30	4	
1	IndiGo	Kolkata	Banglore	1 stop	12	5	6	20	10	
2	Jet Airways	Delhi	Cochin	1 stop	21	5	19	15	19	
3	Multiple carriers	Delhi	Cochin	1 stop	21	5	8	0	21	
4	Air Asia	Banglore	Delhi	non-stop	24	6	23	55	2	
...	...	...	...	...	...	...	...	...	...	
2666	Air India	Kolkata	Banglore	1 stop	6	6	20	30	20	
2667	IndiGo	Kolkata	Banglore	non-stop	27	3	14	20	16	
2668	Jet Airways	Delhi	Cochin	1 stop	6	3	21	50	4	

2669	Air India	Delhi	Cochin	1 stop	6	3	4	0	19

```
ds.replace({"non-stop": 0, "1 stop": 1, "2 stops": 2, "3 stops":3, "4 stops": 4},inplace=True)
ds
```

	Airline	Source	Destination	Total_Stops	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	Arri
0	Jet Airways	Delhi	Cochin	1	6	6	17	30	4	
1	IndiGo	Kolkata	Banglore	1	12	5	6	20	10	
2	Jet Airways	Delhi	Cochin	1	21	5	19	15	19	
3	Multiple carriers	Delhi	Cochin	1	21	5	8	0	21	
4	Air Asia	Banglore	Delhi	0	24	6	23	55	2	
...	...	...	...	...	...	...	...	...	...	
2666	Air India	Kolkata	Banglore	1	6	6	20	30	20	
2667	IndiGo	Kolkata	Banglore	0	27	3	14	20	16	
2668	Jet Airways	Delhi	Cochin	1	6	3	21	50	4	
2669	Air India	Delhi	Cochin	1	6	3	4	0	19	

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

	Airline	Source	Destination	Total_Stops	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	Arri

0	Jet Airways	Delhi	Cochin	1	6	6	17	30	4
1	IndiGo	Kolkata	Banglore	1	12	5	6	20	10
2	Jet Airways	Delhi	Cochin	1	21	5	19	15	19
3	Multiple carriers	Delhi	Cochin	1	21	5	8	0	21
4	Air Asia	Banglore	Delhi	0	24	6	23	55	2
...	...	...	...	...	...	...	...	...	...
2666	Air India	Kolkata	Banglore	1	6	6	20	30	20
2667	IndiGo	Kolkata	Banglore	0	27	3	14	20	16
2668	Jet Airways	Delhi	Cochin	1	6	3	21	50	4
2669	Air India	Delhi	Cochin	1	6	3	4	0	19
2670	Multiple carriers	Delhi	Cochin	1	15	6	4	55	19

2671 rows × 31 columns

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

	Total_Stops	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	Arrival_min	Duration_hours	Duration_m
0	1	6	6	17	30	4	25	10	
1	1	12	5	6	20	10	20	4	

2	1	21	5	19	15	19	0	23
3	1	21	5	8	0	21	0	13
4	0	24	6	23	55	2	45	2

5 rows × 28 columns

```
df_test.shape
```

```
(2671, 28)
```

Separating independent and dependent features.

Now that all our data is numerical after label encoding so we split the data into test and train and drop the price column from the test set because we have to predict the price with our test data set

```
df_train.shape
```

```
(10682, 30)
```

```
df_train.columns
```

```
Index(['Total_Stops', 'Price', 'Journey_day', 'Journey_month', 'Dep_hour',
       'Dep_Min', 'Arrival_hour', 'Arrival_minutes', 'Duration_hours',
       'Duration_mins', 'Air India', 'GoAir', 'IndiGo', 'Jet Airways',
       'Jet Airways Business', 'Multiple carriers',
       'Multiple carriers Premium economy', 'SpiceJet', 'Trujet', 'Vistara',
       'Vistara Premium economy', 'Source_Chennai', 'Source_Delhi',
       'Source_Kolkata', 'Source_Mumbai', 'Destination_Cochin',
       'Destination_Delhi', 'Destination_Hyderabad', 'Destination_Kolkata',
       'Destination_New Delhi'],
      dtype='object')
```

```
x = df_train.loc[:, ['Total_Stops','Journey_day', 'Journey_month', 'Dep_hour',
'Dep_Min', 'Arrival_hour', 'Arrival_minutes', 'Duration_hours',
'Duration_mins', 'Air India', 'GoAir', 'IndiGo', 'Jet Airways',
'Jet Airways Business', 'Multiple carriers',
'Multiple carriers Premium economy', 'SpiceJet', 'Trujet', 'Vistara',
'Vistara Premium economy', 'Source_Chennai', 'Source_Delhi',
'Source_Kolkata', 'Source_Mumbai', 'Destination_Cochin',
'Destination_Delhi', 'Destination_Hyderabad', 'Destination_Kolkata',
'Destination_New Delhi']]
```

```
x.head()
```

	Total_Stops	Journey_day	Journey_month	Dep_hour	Dep_Min	Arrival_hour	Arrival_minutes	Duration_hours	Durati
0	0	24	3	22	20	1	10	2	
1	2	1	5	5	50	13	15	7	
2	2	9	6	9	25	4	25	19	
3	1	12	5	18	5	23	30	5	
4	1	1	3	16	50	21	35	4	

5 rows × 29 columns

```
y=df_train['Price']
```

```
y.head()
```

```
0     3897
1     7662
```

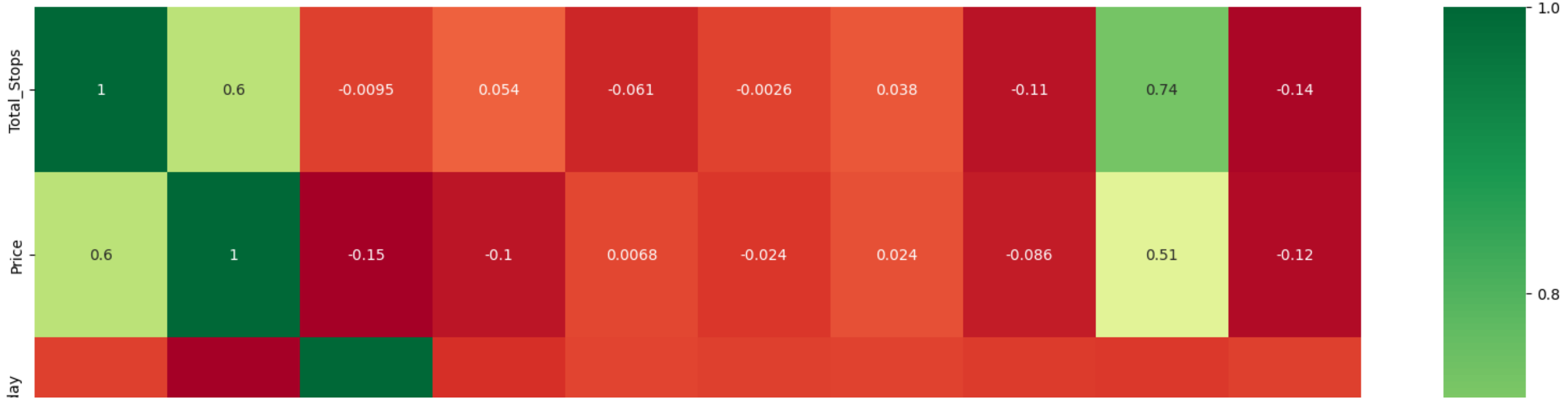
```
2    13882
3     6218
4    13302
Name: Price, dtype: int64
```

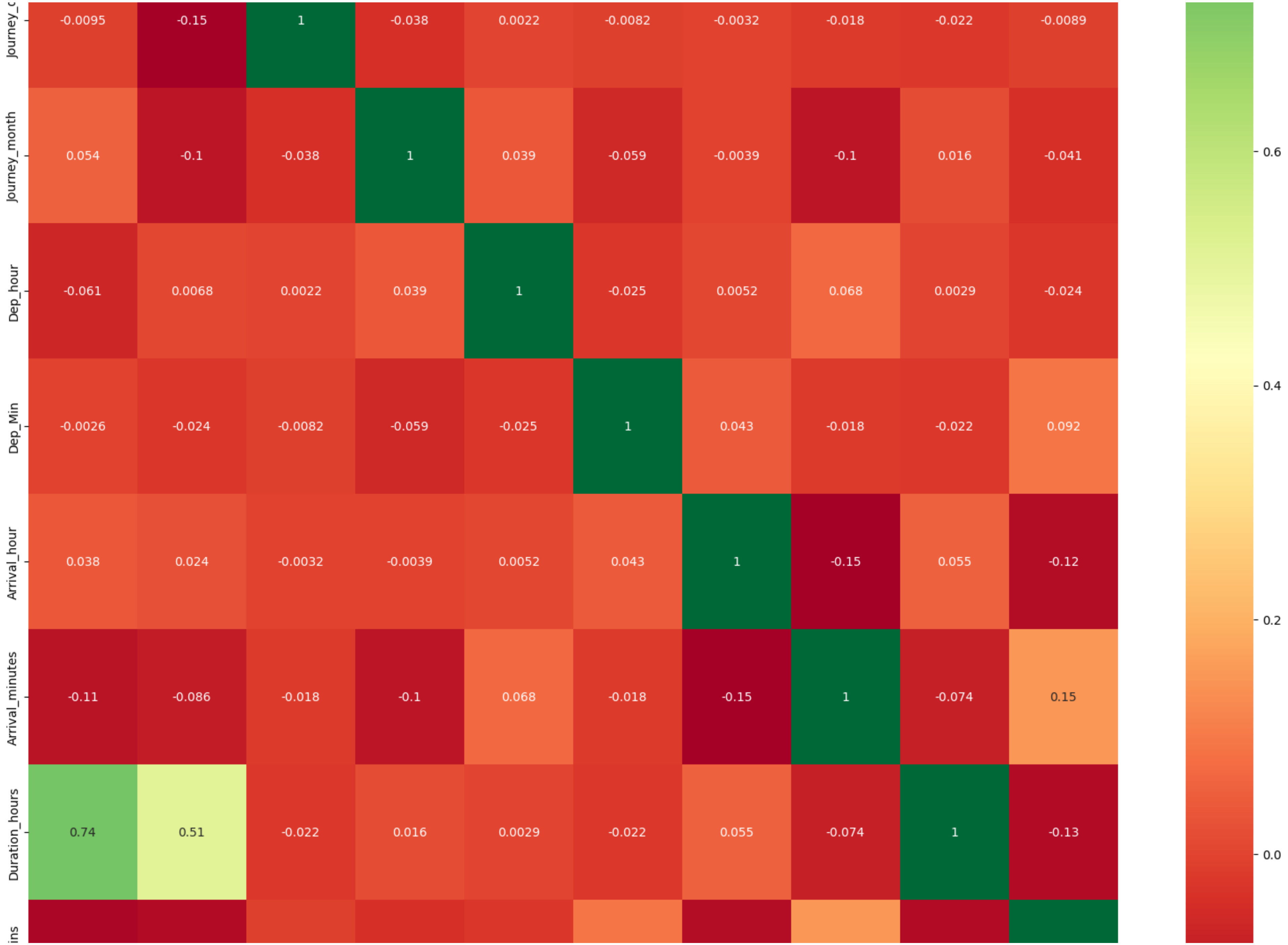
Feature selection

it is nothing but to find the best feature that contributes the most and that has good relationship with the target values. The main reason to apply feature selection is to select important feature so that we don't face the problem of multiple dimensions.

```
#heatmap-finding correlation betwwen independent and dependent features
plt.figure(figsize = (20,20))
sns.heatmap(df.corr(),annot = True,cmap = "RdYlGn")
plt.show()
```

```
#    (df.corr)data: The matrix or dataset you want to visualize as a heatmap.
#    annot: If set to True, it will annotate each cell of the heatmap with the numeric value from the data.
#   cmap: The color map used for the heatmap. "coolwarm" is one of many available colormaps.
```




Journey_d
Journey_month
Dep_hour
Dep_Min
Arrival_hour
Arrival_minutes
Duration_hours
ins
-0.0095 -0.15 1 -0.038 0.0022 -0.0082 -0.0032 -0.018 -0.022 -0.0089
0.054 -0.1 -0.038 1 0.039 -0.059 -0.0039 -0.1 0.016 -0.041
-0.061 0.0068 0.0022 0.039 1 -0.025 0.0052 0.068 0.0029 -0.024
-0.0026 -0.024 -0.0082 -0.059 -0.025 1 0.043 -0.018 -0.022 0.092
0.038 0.024 -0.0032 -0.0039 0.0052 0.043 1 -0.15 0.055 -0.12
-0.11 -0.086 -0.018 -0.1 0.068 -0.018 -0.15 1 -0.074 0.15
0.74 0.51 -0.022 0.016 0.0029 -0.022 0.055 -0.074 1 -0.13
0.6
0.4
0.2
0.0

using heatmap we can see that the "Total_Stops'is positively correlated with 'Price' which leads to increasesin cost of fuel so increase price.Also Total_Stops is highly correlated with Duration_hours means if the no.of stops would increasess,the duration of hours of the flight will also increase.

x

	Total_Stops	Journey_day	Journey_month	Dep_hour	Dep_Min	Arrival_hour	Arrival_minutes	Duration_hours	D
0	0	24	3	22	20	1	10	2	
1	2	1	5	5	50	13	15	7	
2	2	9	6	9	25	4	25	19	
3	1	12	5	18	5	23	30	5	
4	1	1	3	16	50	21	35	4	
...	...	...	...	...	...	...	...	...	
10678	0	9	4	19	55	22	25	2	
10679	0	27	4	20	45	23	20	2	
10680	0	27	4	8	20	11	20	3	
10681	0	1	3	11	30	14	10	2	
10682	2	9	5	10	55	19	15	8	

10682 rows × 29 columns

In machine learning, ExtraTreeRegressor refers to the Extra Trees Regressor, which is a type of ensemble learning algorithm used for regression tasks.Ensemble learning algorithms means it is a technique that involves combining the predictions of multiple individual models

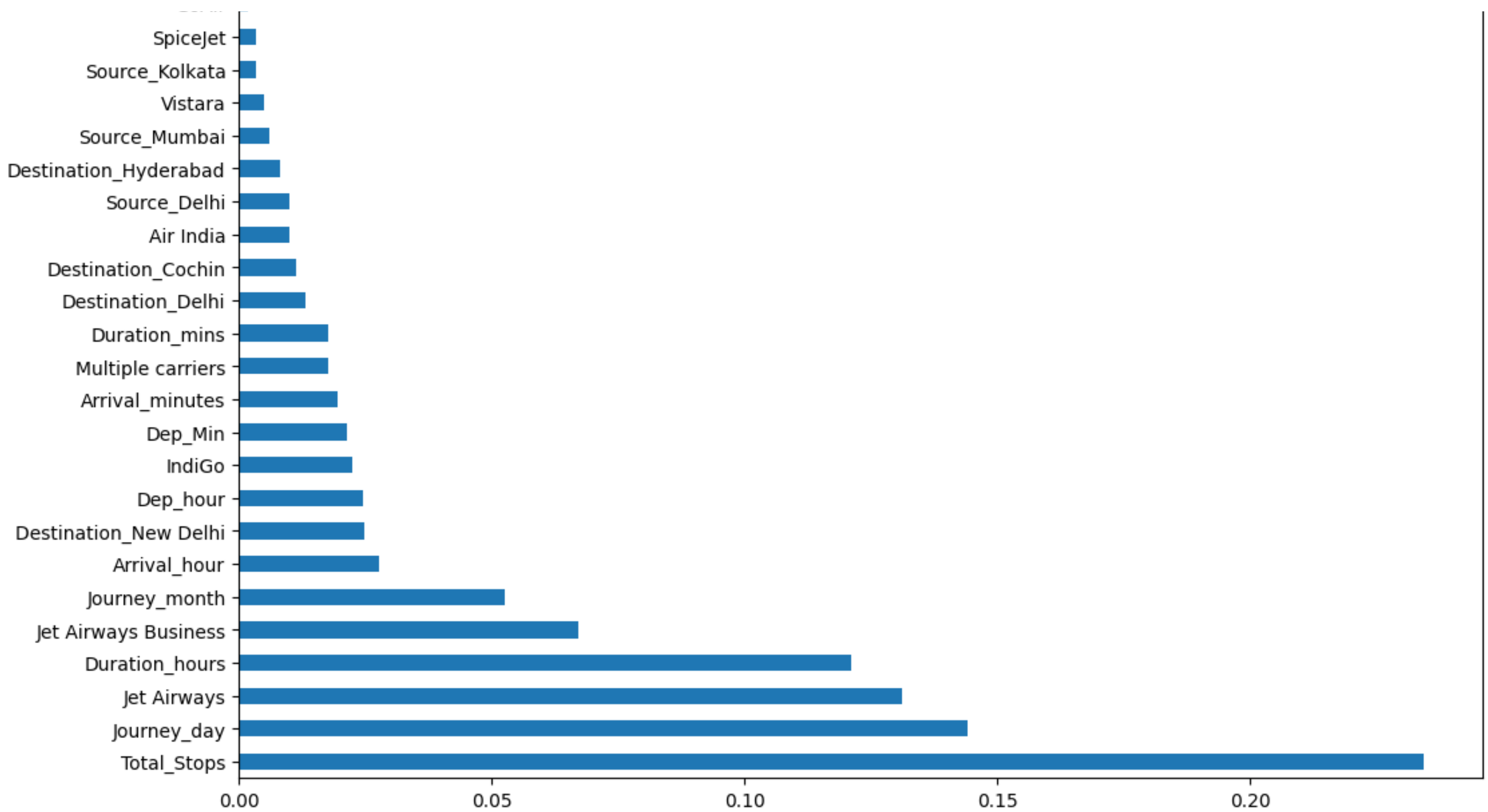
```
#import features using Extratreesregressor

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

```
▾ ExtraTreesRegressor
ExtraTreesRegressor()
```

```
selection.feature_importances_
```

```
array([2.34256105e-01, 1.44027931e-01, 5.25673591e-02, 2.44769226e-02,
       2.14357286e-02, 2.76833134e-02, 1.95973628e-02, 1.21065870e-01,
       1.76248680e-02, 1.00725758e-02, 1.82376142e-03, 2.23972409e-02,
       1.31200015e-01, 6.70664674e-02, 1.77486765e-02, 8.60863498e-04,
       3.26127771e-03, 1.18198221e-04, 4.89402976e-03, 8.57002560e-05,
       4.89584164e-04, 9.99408356e-03, 3.45830016e-03, 6.00342864e-03,
       1.14038095e-02, 1.32131387e-02, 8.01200763e-03, 4.37937626e-04,
       2.47234438e-02])
```

```
#pot graph of feature importance for better visualization
plt.figure(figsize = (12,8))
feat_importances = pd.Series(selection.feature_importances_, index = x.columns)
feat_importances.nlargest(25).plot(kind = 'barh')
plt.show()
```



we can see that Total_Stops is the feature with the highest feature importance in deciding tree

Model Fitting

Splitting the data into train and test data. We taken 70% data for training and remaining 30% for testing.

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=42)
```

Random Forest

```
from sklearn.ensemble import RandomForestRegressor
model=RandomForestRegressor()
model.fit(x_train,y_train)
y_pred = model.predict(x_test)
y_pred
```

```
array([16720.79,  5308.59,  8918.64, ...,  5881.42,  3287.9 ,  7083.2 ])
```

```
model.score(x_train,y_train) #training data score 95
```

```
0.955435814854203
```

```
model.score(x_test,y_test) #testing r2 score is 80
```

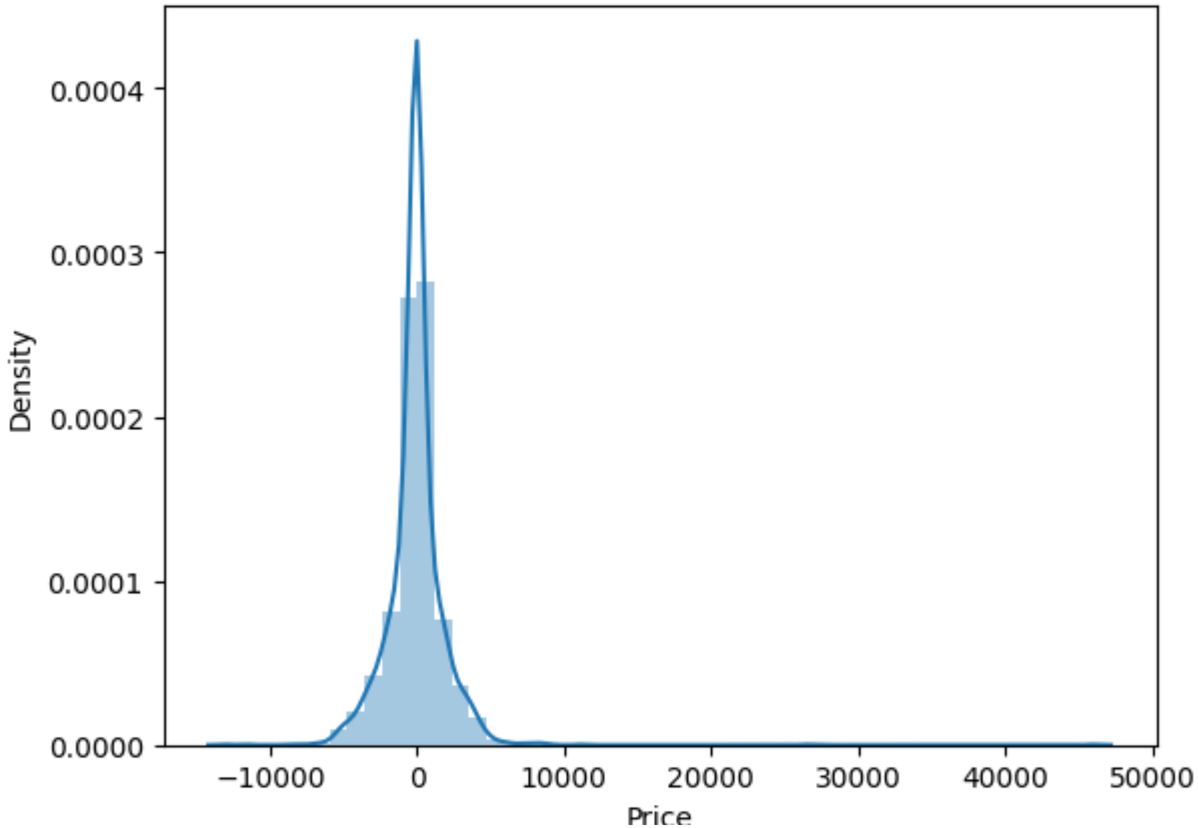
```
0.8041893963441062
```

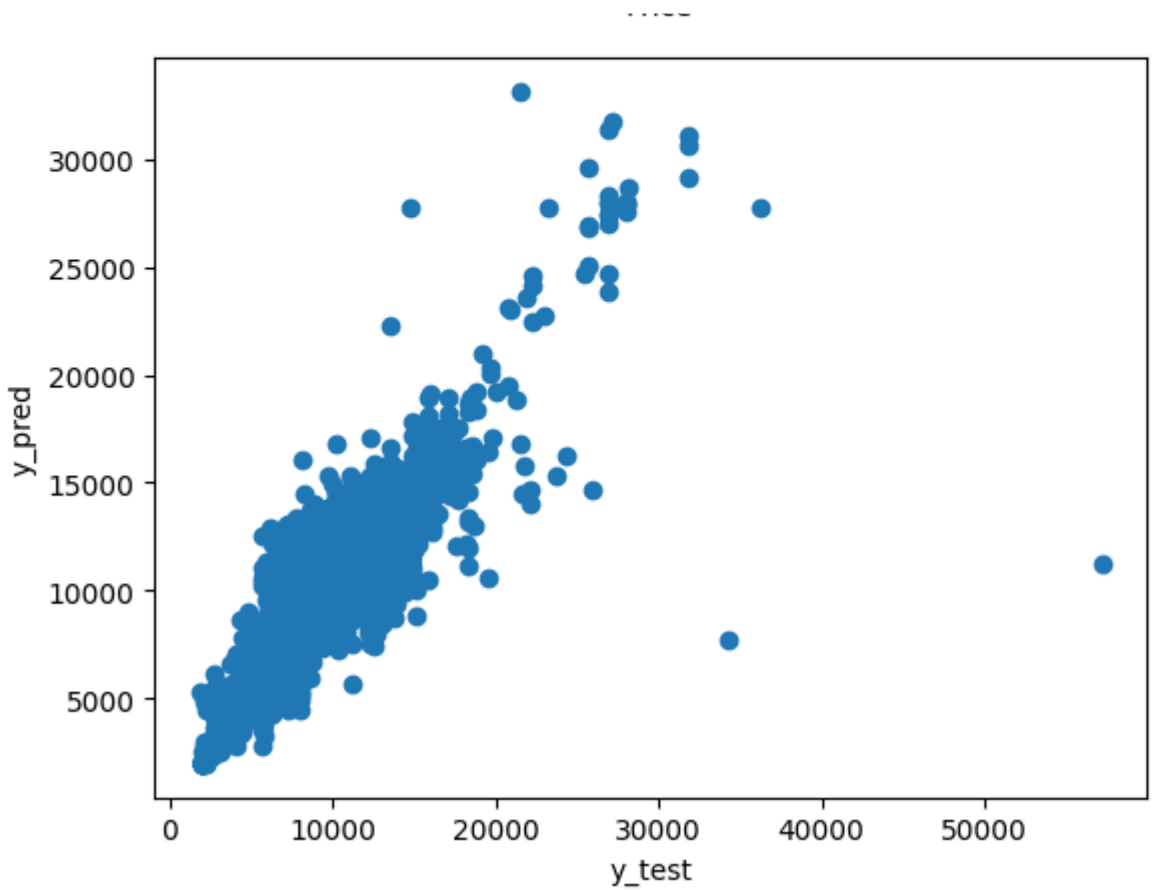
Fitting parametric distribution

Fitting parametric distribution

we use displot() to fit a parametric distribution to a dataset and visually evaluated how closely it corresponds to the observed data.it is difference between 'y_test' and 'prediction' should be minimal.Here most of the residuals are 0,which means our model is well.

```
sns.distplot(y_test-y_pred)
plt.show() #Gaussian Disrtibution
plt.scatter(y_test,y_pred)
plt.xlabel("y_test")
plt.ylabel("y_pred")
plt.show()
   #scatterplot we can see good linear relationship between our actualtest set price and the
                                     #predicted prices
```






```
#create a data frame with actual value and predicted value
df1=pd.DataFrame({'Actual value':y_test,'Predicted Value':y_pred})
df1
```

	Actual value	Predicted Value
6075	16655	16720.790000
3544	4959	5308.590000
9291	9187	8918.640000
5032	3858	3738.590000

[illegible]	[illegible]	[illegible]
2483	12898	14844.154333
...	...	...
7917	16263	14617.050000
5858	10844	13291.708667
2689	5000	5881.420000
4486	3100	3287.900000
7877	6734	7083.200000

3205 rows × 2 columns

Performace Evaluation

```
from sklearn import metrics
print("Mean Absolute Error: ", metrics.mean_absolute_error(y_test,y_pred))
print("Mean Absolute Percentage Error: ", metrics.mean_absolute_percentage_error(y_test,y_pred))
print("MSE",metrics.mean_squared_error(y_test,y_pred))
```

```
Mean Absolute Error:  1164.966200935137
Mean Absolute Percentage Error:  0.1298250944716494
MSE 3994717.7486367226
```

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

```
0.8041893963441062
```

Hyperparameter Tuning

method for hyperparameter tunning

1.RandomizedSearchCV 2.GridSearchCV

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

```
#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_sample_leaf = [1, 2, 5, 10]
```

```
# creating 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_sample_leaf}
```

```
# Random search of parameters, using 5 fold cross validation,
# search across 100 different combinations
model_random = RandomizedSearchCV(estimator = model,param_distributions = random_grid)
```

```
model_random.fit(x_train,y_train)
```

▸ **RandomizedSearchCV**
▸ **estimator: RandomForestRegressor**
▸ RandomForestRegressor

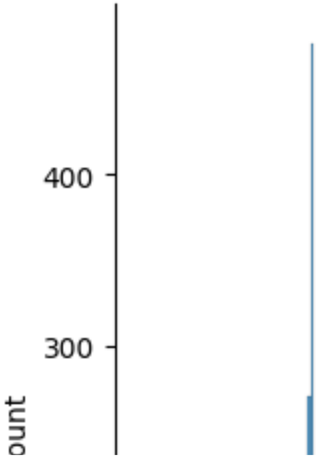
```
model_random.best_params_
```

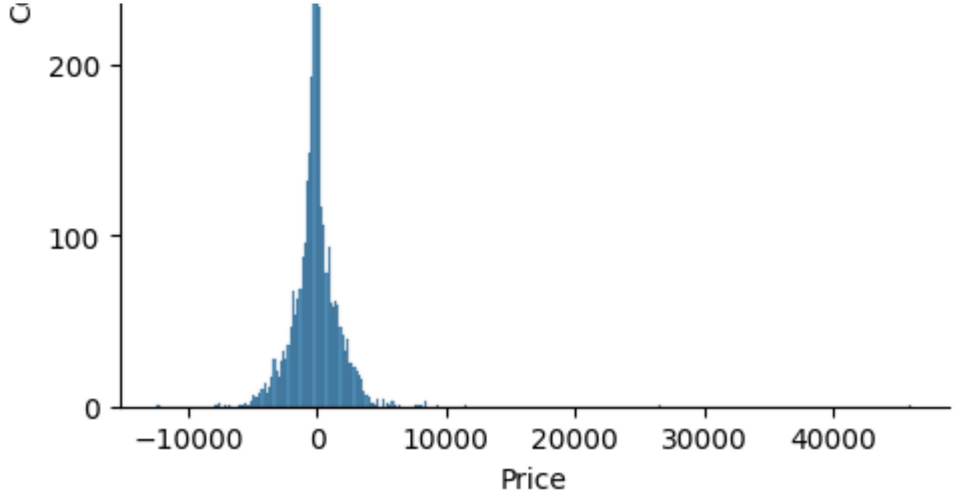
```
{'n_estimators': 900,
 'min_samples_split': 5,
 'min_samples_leaf': 1,
 'max_features': 'auto',
 'max_depth': 15}
```

```
prediction=model_random.predict(x_test)
```

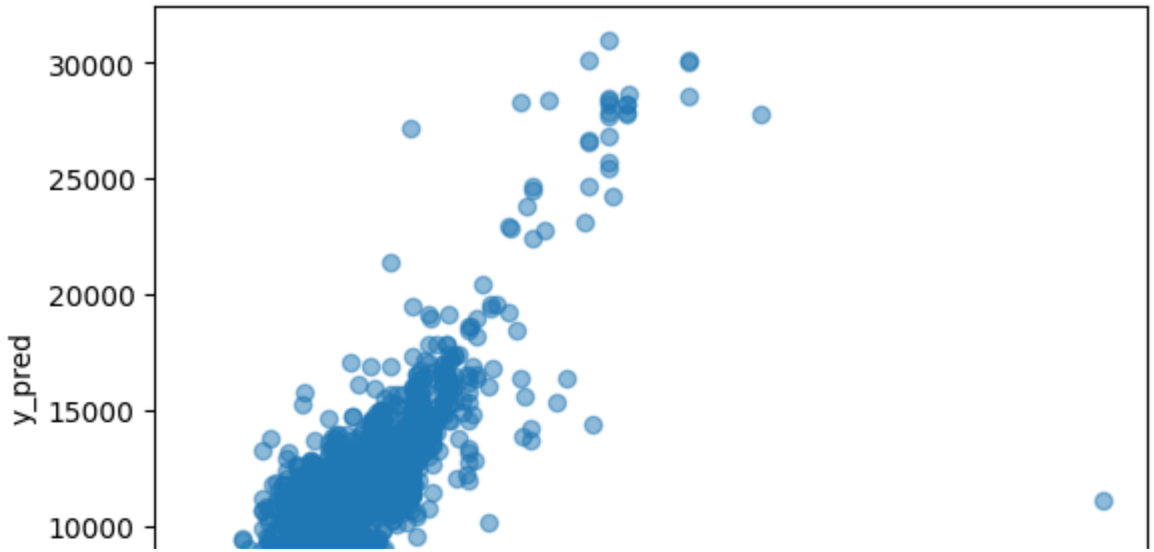
```
sns.displot(y_test-prediction)   #plotting the residuals
plt.show
```

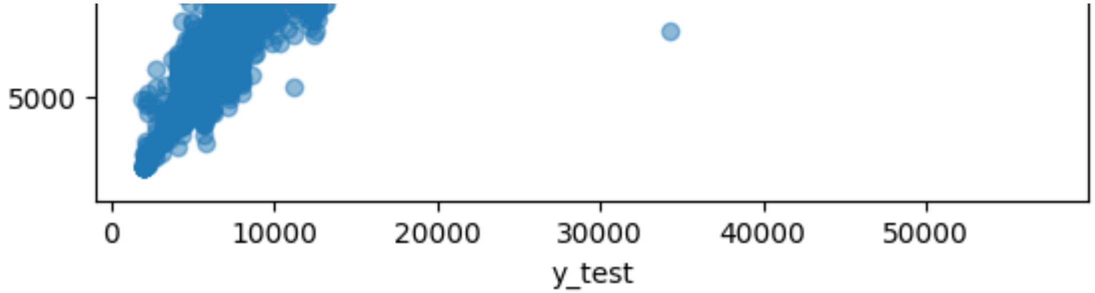
<function matplotlib.pyplot.show(close=None, block=None)>






```
#as we see that most residual is 0 that is mean that our model generalizing well.

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






```
#creating a data frame with actual value and predicted value
df2=pd.DataFrame({'Actual value':y_test,'Predicted value':prediction})
df2
```

	Actual value	Predicted value
6075	16655	16548.803552
3544	4959	5636.082775
9291	9187	8713.459497
5032	3858	3675.326612
2483	12898	14617.117384
...	...	...
7917	16263	14587.361963
5858	10844	12880.355263
2689	5000	5779.954897
4486	3100	3298.662389
7877	6734	7031.615469

3205 rows × 2 columns

```
from sklearn.metrics import r2_score


print("Mean Absolute Error: ", metrics.mean_absolute_error(y_test,prediction))
print("Mean Absolute Percentage Error: ", metrics.mean_absolute_percentage_error(y_test,prediction))
print("MSE",metrics.mean_squared_error(y_test,prediction))
```

```
Mean Absolute Error:  1130.1063347630484
Mean Absolute Percentage Error:  0.1272866127851626
MSE 3614570.533632817
```

```
rf_score=r2_score(y_test,prediction)
rf_score
```

```
0.8228232173877643
```

DECISION TREE

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.30,random_state=42)
```

```
from sklearn.tree import DecisionTreeRegressor
dt=DecisionTreeRegressor()
dt.fit(x_train,y_train) #training a model
y_decision=dt.predict(x_test)
```

```
from sklearn.metrics import r2_score
print("mean Absolute error",metrics.mean_absolute_error(y_test,y_decision))
print("mean absolute percentage Error",metrics.mean_absolute_percentage_error(y_test,y_decision))
print("mean Squred error",metrics.mean_squared_error(y_test,y_decision))
```

```
mean Absolute error 1356.279958398336
mean absolute percentage Error 0.1495652097506327
mean Squred error 6147595.558179754
```

```
de_score=r2_score(y_test,y_decision)
de_score
```

```
0.6986609635461034
```

Linear Regression

```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
lr_pred=lr.predict(x_test)
```

```
print("Mean Absolute Error: ", metrics.mean_absolute_error(y_test,lr_pred))
print("Mean Absolute Percentage Error: ", metrics.mean_absolute_percentage_error(y_test,lr_pred))
print("MSE",metrics.mean_squared_error(y_test,lr_pred))
```

```
Mean Absolute Error:  1937.8989872881186
Mean Absolute Percentage Error:  0.23768723337133388
MSE 7621946.516730598
```

```
lr_score=r2_score(y_test,lr_pred)
lr_score
```

```
0.626392140224854
```

KNN

```
from sklearn.neighbors import KNeighborsRegressor
knn=KNeighborsRegressor(n_neighbors=4)
knn.fit(x_train,y_train)
knn_pred=knn.predict(x_test)
```

```
print("Mean Absolute Error: ", metrics.mean_absolute_error(y_test,knn_pred))
print("Mean Absolute Percentage Error: ", metrics.mean_absolute_percentage_error(y_test,knn_pred))
print("MSE",metrics.mean_squared_error(y_test,knn_pred))
```

```
Mean Absolute Error:  1888.5921216848674
Mean Absolute Percentage Error:  0.21237390568357523
MSE 8995061.713319033
```

```
knn_score=r2_score(y_test,knn_pred)
knn_score
```

```
0.5590856288637396
```

RESULT

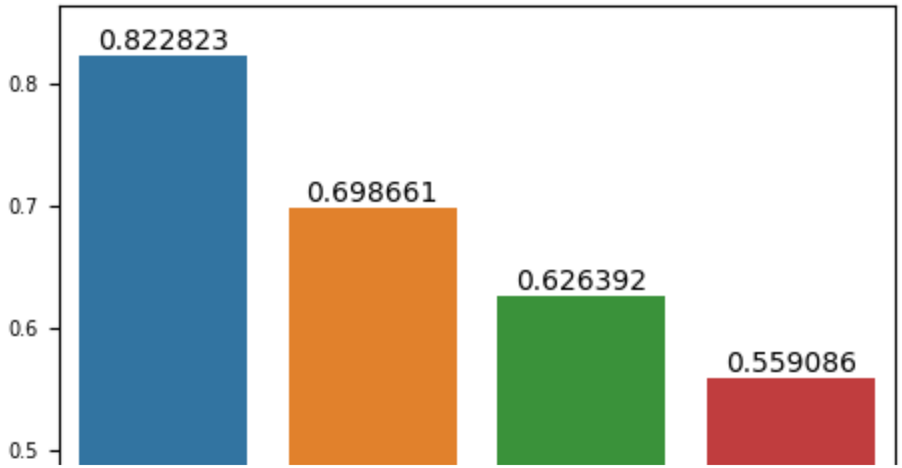
Comparison of the best model evaluated by cross validation

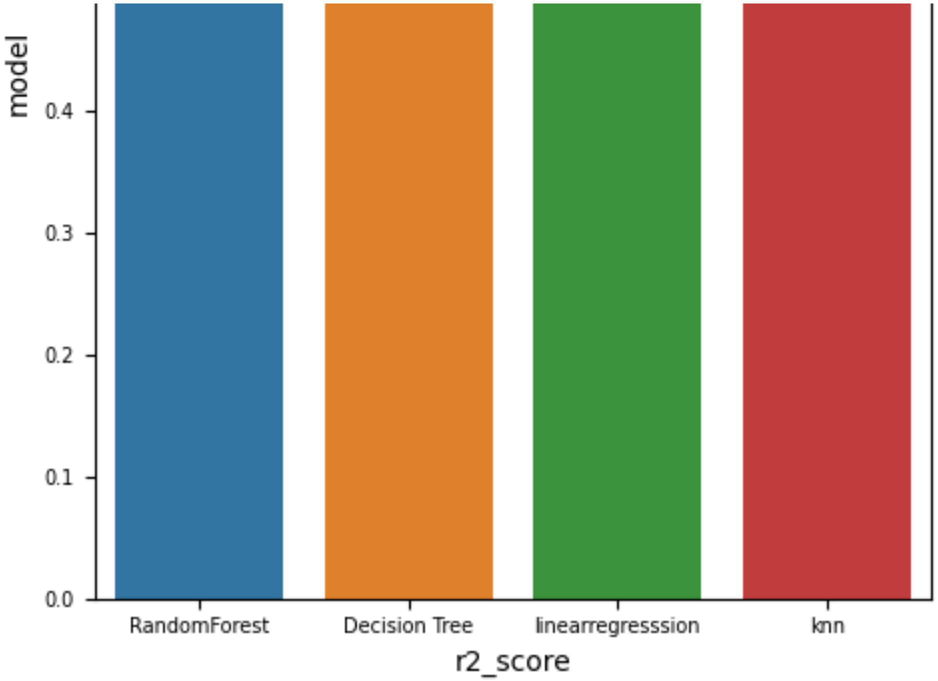
- RandomForestRegressor-CV : 0.81
- Desicion Tree-CV :0.69
- Linear Regression :0.62
- KNN :0.55

```
lst=[rf_score,de_score,lr_score,knn_score]
lst1=["RandomForest","Decision Tree","linearregresssion","knn"]
for i in range(len(lst1)):
  print("The r2 score usin "+lst1[i]+ " is:\t"+str(lst[i])+"%")
```

```
The r2 score usin RandomForest is:      0.8228232173877643%
The r2 score usin Decision Tree is:     0.6986609635461034%
The r2 score usin linearregresssion is: 0.626392140224854%
The r2 score usin knn is:       0.5590856288637396%
```

```
plt.figure(figsize=(5,6))
plt.xlabel("r2_score")
plt.ylabel("model")
sn=sns.barplot(x=lst1,y=lst,)
for label in sn.containers:
  sn.bar_label(label)
plt.tight_layout()
plt.tick_params(labelsize=7)
```





Conclusion

By using machine learning Algorithms on the dataset,one can predict the dynamic fare of flight,thereby obtaining the predicted flight fare values to obtain a flight ticket at the lowest cost.The accuracy of the model is determined by the R-squred values obtained from the algorithm.The random forest algorithm was utilized,resulting in an accuracy of 82%.