PROJECT: FLIGHT PRICE PREDICTION

In [1]:
```
#Importing the libreris like pandas, numpy for selecing the data and convering the data to
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
#Creating the variable train_df and loading the dataset  to the variable train_df
train_df=pd.read_excel('flightprice.xlsx')
train_df
```

Out[2]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	A
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	
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	19h	2 stops	
3	IndiGo	12/05/2019	Kolkata	Banglore	CCU → NAG → BLR	18:05	23:30	5h 25m	1 stop	
4	IndiGo	01/03/2019	Banglore	New Delhi	BLR → NAG → DEL	16:50	21:35	4h 45m	1 stop	
...	...	...	...	...	...	...	...	...	...	...
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	
10680	Jet Airways	27/04/2019	Banglore	Delhi	BLR → DEL	08:20	11:20	3h	non-stop	
10681	Vistara	01/03/2019	Banglore	New Delhi	BLR → DEL	11:30	14:10	2h 40m	non-stop	

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	A
10682	Air India	9/05/2019	Delhi	Cochin	DEL → GOI → BOM → COK	10:55	19:15	8h 20m	2 stops	

10683 rows × 11 columns

In [3]:
```
train_df.info()
```

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

In [4]:
```
#checking the shape of the train_df dataset shape
train_df.shape
```

Out[4]:
```
(10683, 11)
```

train_df having the shape 10683 rows 11 columns

In [5]:
```
# checking for null values
train_df.isnull().sum()
```

Out[5]:
```
Airline            0
Date_of_Journey    0
Source             0
Destination        0
Route              1
Dep_Time           0
Arrival_Time       0
Duration           0
Total_Stops        1
Additional_Info    0
Price              0
dtype: int64
```

In [6]:
```
# only 2 null value is present so we can drop that rows
train_df.dropna(inplace=True)
```

In [7]:
```
ng the nullrows once again checking the null values
```

```
train_df.isnull().sum()
```

Out[7]:

```
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
```

From the above table we can see that there is no null values in the train_df dataset

Exploratory Data Analysis

From description we can see that Date_of_Journey,Arrival_hour is a object data type, Therefore, we have to convert this datatype into timestamp to use this column properly for prediction For this we require pandas to_datetime to convert object data type to datetime dtype

In [9]:

```
# new columns Journey_date, journey_month addeded to the train_df dataset for which the da
# new columns Arrival_hour, Arrival_minute addeded to the train_df dataset for which the d
train_df["Journey_date"] =pd.to_datetime(train_df["Date_of_Journey"],format="%d/%m/%Y").dt
train_df["Journey_month"] =pd.to_datetime(train_df["Date_of_Journey"],format="%d/%m/%Y").d
train_df["Arrival_hour"] = pd.to_datetime(train_df["Arrival_Time"]).dt.hour
train_df["Arrival_minute"] = pd.to_datetime(train_df["Arrival_Time"]).dt.minute
```

In [10]:

```
# since we have converted Date_of_Journey and Arrival_Time column into integers, Now we ca
# droping the Date_of_Journey and Arrival_Time
train_df.drop(['Date_of_Journey','Arrival_Time'],axis=1,inplace=True)
```

In [11]:

```
# Time taken by plane to reach destination is called Duration
# It is the differnce between Departure Time and Arrival time
# Assigning and converting Duration column into list
duration = list(train_df["Duration"])

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

duration_hours = []
duration_mins = []
for i in range(len(duration)):
    duration_hours.append(int(duration[i].split(sep = "h")[0]))    # Extract hours from du
    duration_mins.append(int(duration[i].split(sep = "m")[0].split()[-1]))   # Extracts or
# Adding duration_hours and duration_mins list to train_data dataframe

train_df["Duration_hours"] = duration_hours
train_df["Duration_mins"] =duration_mins
# now we can drop the duration column
train_df.drop(["Duration"], axis =1, inplace=True)
```

In [12]:
```
train_df
```

Out[12]:

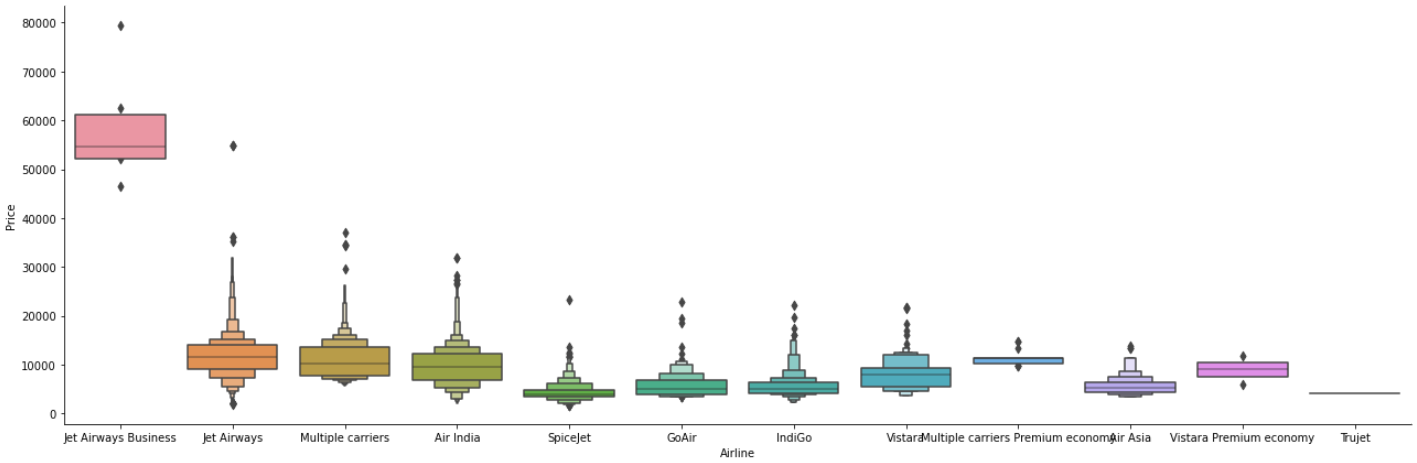
	Airline	Source	Destination	Route	Dep_Time	Total_Stops	Additional_Info	Price	Journey_date	Journ
0	IndiGo	Banglore	New Delhi	BLR → DEL	22:20	non-stop	No info	3897	24	
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	2 stops	No info	7662	1	
2	Jet Airways	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	2 stops	No info	13882	9	
3	IndiGo	Kolkata	Banglore	CCU → NAG → BLR	18:05	1 stop	No info	6218	12	
4	IndiGo	Banglore	New Delhi	BLR → NAG → DEL	16:50	1 stop	No info	13302	1	
...	...	...	...	...	...	...	...	...	...	...
10678	Air Asia	Kolkata	Banglore	CCU → BLR	19:55	non-stop	No info	4107	9	
10679	Air India	Kolkata	Banglore	CCU → BLR	20:45	non-stop	No info	4145	27	
10680	Jet Airways	Banglore	Delhi	BLR → DEL	08:20	non-stop	No info	7229	27	
10681	Vistara	Banglore	New Delhi	BLR → DEL	11:30	non-stop	No info	12648	1	
10682	Air India	Delhi	Cochin	DEL → GOI → BOM → COK	10:55	2 stops	No info	11753	9	

10682 rows × 14 columns

Handling Categorical Data and Numerical Data

In [13]:
```
import matplotlib.pyplot as plt
import seaborn as sns
# Airline vs price
```

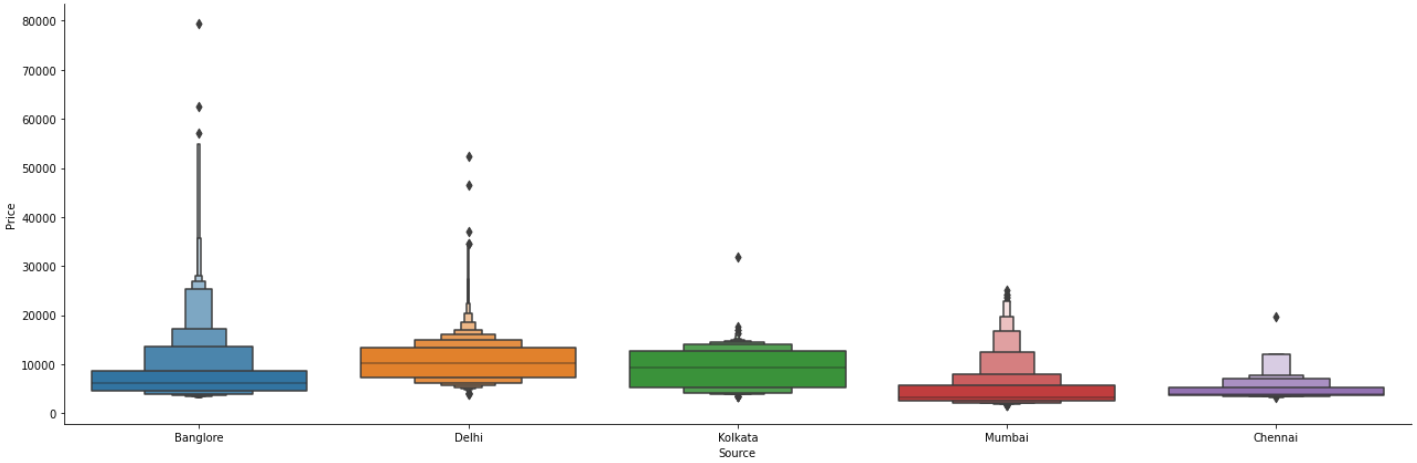
```
sns.catplot(y= "Price", x = "Airline", data = train_df.sort_values("Price", ascending = Fa
plt.show()
```



From the grap we can say that jet Airways Business as the maximum price

In [14]:
```
# Compare Source and Price
sns.catplot(y = "Price", x= "Source", data = train_df.sort_values("Price", ascending = Fal
```

Out[14]:
```
<seaborn.axisgrid.FacetGrid at 0x8cc3c553d0>
```



From the graph we can say that banglore as the maximum price

In [15]:
```
# changing the categorical value to numerical value by labelencoder tecnique
from sklearn import preprocessing
le=preprocessing.LabelEncoder()
train_df['Airline']=le.fit_transform(train_df['Airline'])
train_df['Source']=le.fit_transform(train_df['Source'])
train_df['Destination']=le.fit_transform(train_df['Destination'])
train_df['Dep_Time']=le.fit_transform(train_df['Dep_Time'])
train_df['Total_Stops']=le.fit_transform(train_df['Total_Stops'])
train_df['Route']=le.fit_transform(train_df['Route'])
train_df['Additional_Info']=le.fit_transform(train_df['Additional_Info'])
train_df
```

Out[15]:

	Airline	Source	Destination	Route	Dep_Time	Total_Stops	Additional_Info	Price	Journey_date	Journey
0	3	0	5	18	211	4	8	3897	24	
1	1	3	0	84	31	1	8	7662	1	
2	4	2	1	118	70	1	8	13882	9	
3	3	3	0	91	164	0	8	6218	12	

	Airline	Source	Destination	Route	Dep_Time	Total_Stops	Additional_Info	Price	Journey_date	Journey
4	3	0	5	29	149	0	8	13302	1	
...	...	...	...	...	...	...	...	...	...	
10678	0	3	0	64	183	4	8	4107	9	
10679	1	3	0	64	193	4	8	4145	27	
10680	4	0	2	18	58	4	8	7229	27	
10681	10	0	5	18	92	4	8	12648	1	
10682	1	2	1	108	85	1	8	11753	9	

10682 rows × 14 columns

In [16]:
```
# checking for skewness in the train_df dataset
train_df.skew()
```

Out[16]:
```
Airline            0.731057
Source            -0.424023
Destination        1.244046
Route             -0.501911
Dep_Time           0.194914
Total_Stops        0.631532
Additional_Info   -1.779689
Price              1.812405
Journey_date       0.118174
Journey_month     -0.387409
Arrival_hour      -0.370146
Arrival_minute     0.110945
Duration_hours     0.851197
Duration_mins     -0.090680
dtype: float64
```

In [17]:
```
# checking for co relation
train_df.corr()
```

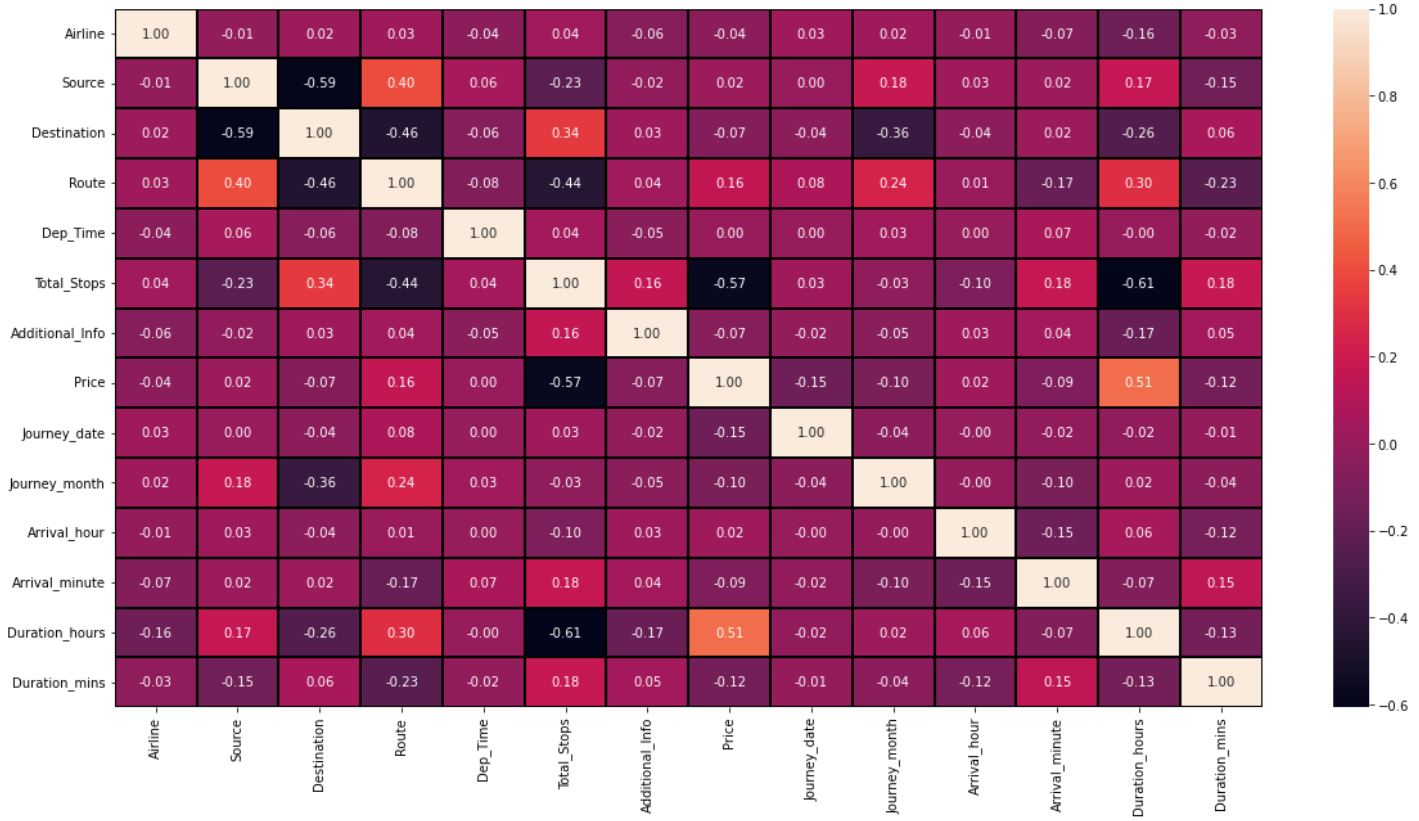
Out[17]:

	Airline	Source	Destination	Route	Dep_Time	Total_Stops	Additional_Info	Price	J
Airline	1.000000	-0.013397	0.018446	0.025214	-0.039508	0.035973	-0.060748	-0.039565	
Source	-0.013397	1.000000	-0.592576	0.403412	0.055194	-0.225605	-0.022109	0.015999	
Destination	0.018446	-0.592576	1.000000	-0.461176	-0.063625	0.337872	0.026821	-0.071122	
Route	0.025214	0.403412	-0.461176	1.000000	-0.082013	-0.437749	0.035152	0.164149	
Dep_Time	-0.039508	0.055194	-0.063625	-0.082013	1.000000	0.044647	-0.052828	0.002931	
Total_Stops	0.035973	-0.225605	0.337872	-0.437749	0.044647	1.000000	0.164054	-0.571221	
Additional_Info	-0.060748	-0.022109	0.026821	0.035152	-0.052828	0.164054	1.000000	-0.065463	
Price	-0.039565	0.015999	-0.071122	0.164149	0.002931	-0.571221	-0.065463	1.000000	
Journey_date	0.026137	0.004902	-0.041025	0.081632	0.001450	0.029225	-0.016296	-0.153774	
Journey_month	0.024674	0.183268	-0.364682	0.244186	0.034610	-0.026328	-0.051491	-0.103643	
Arrival_hour	-0.007567	0.025635	-0.039729	0.013898	0.000015	-0.095650	0.026204	0.024244	
Arrival_minute	-0.071092	0.021040	0.017196	-0.173352	0.066656	0.175980	0.041310	-0.086155	
Duration_hours	-0.158136	0.166121	-0.258446	0.295444	-0.001247	-0.606137	-0.168815	0.508778	
Duration_mins	-0.028009	-0.145193	0.061235	-0.232427	-0.016261	0.182223	0.046910	-0.124855	

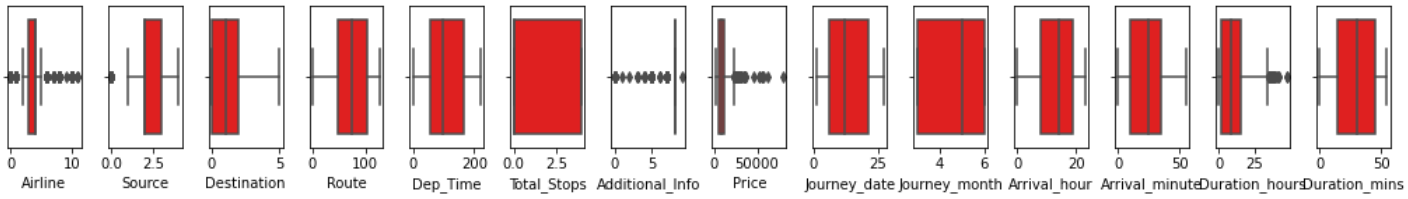
From the table we can say that the Duration_hours is highly co_related with the target column (price column) and followed by other columns as shown in the co relation columns

In [18]:
```
#Reprasantation of co relation values by using the heatmap
import matplotlib.pyplot as plt
plt.figure(figsize=(20,10))
sns.heatmap(train_df.corr(),annot=True,linewidth=0.1,linecolor='black',fmt='0.2f')
```

Out[18]:
```
<AxesSubplot:>
```



In [19]:
```
#check for outliers with the help of box plot
columns_list=train_df.columns.values
ncol=100
nrows=50
plt.figure(figsize=(ncol,ncol))
for i in range (0,len(columns_list)):
    plt.subplot(nrows,ncol,i+1)
    sns.boxplot(train_df[columns_list[i]],color='red',orient='h')
    plt.tight_layout()
```



by graph we come to know that outliers present in the train_df dataset

In [20]:
```
# checking how much number of outliers are present in the dataframe
from scipy.stats import zscore
z=np.abs(zscore(train_df))
print(train_df.shape)
print(z.shape)
```

```
print(np.where(z>3))
len(np.where(z>3)[0])
```

```
(10682, 14)
(10682, 14)
(array([  123,   226,   396,   486,   510,   553,   597,   628,   657,
         785,   785,   826,   936,   946,   959,   966,   969,   975,
        1043,  1067,  1113,  1196,  1246,  1341,  1346,  1424,  1442,
        1466,  1478,  1496,  1551,  1629,  1654,  1657,  1785,  1791,
        1804,  1918,  2055,  2079,  2099,  2108,  2172,  2405,  2481,
        2495,  2495,  2553,  2556,  2556,  2618,  2635,  2641,  2654,
        2693,  2718,  2924,  2924,  3032,  3032,  3111,  3111,  3257,
        3400,  3457,  3535,  3700,  3700,  3788,  3790,  3815,  3966,
        4012,  4047,  4118,  4463,  4521,  4521,  4655,  4676,  4829,
        5013,  5013,  5136,  5372,  5372,  5439,  5516,  5628,  5662,
        5701,  5701,  5710,  5711,  5719,  5738,  5745,  5782,  5856,
        5933,  5953,  5986,  6024,  6059,  6085,  6314,  6321,  6336,
        6407,  6476,  6576,  6576,  6588,  6605,  6884,  6991,  7189,
        7232,  7306,  7329,  7351,  7356,  7377,  7537,  7553,  7554,
        7611,  7612,  7617,  7713,  7724,  7724,  7756,  7876,  7902,
        7906,  8020,  8080,  8127,  8153,  8360,  8409,  8412,  8451,
        8470,  8536,  8598,  8601,  8621,  8698,  8729,  8856,  8940,
        8957,  8957,  8990,  9019,  9019,  9192,  9238,  9246,  9395,
        9483,  9550,  9631,  9656,  9672,  9702,  9714,  9847,  9973,
       10019, 10051, 10112, 10159, 10181, 10188, 10325, 10352, 10363,
       10363, 10382, 10438, 10455, 10510, 10510, 10638], dtype=int64), array([ 7, 12,  7,
7,  7, 12,  7,  7,  7,  6,  7,  7,  7,  7,  7, 12, 12,
        7,  6, 12, 12,  7,  7,  7, 12,  7, 12,  7,  7, 12, 12,  7,  7, 12,
        7, 12, 12,  7,  7,  6,  7,  7, 12,  6,  6,  6,  7, 12,  6,  7,  7,
        7, 12, 12,  7, 12,  6,  7,  6,  7,  6,  7,  7,  7, 12,  7,  6,  7,
        6, 12, 12, 12,  7, 12, 12, 12,  6,  7, 12,  7,  7,  6,  7,  7,  6,
        7,  7, 12,  6,  7,  7, 12,  7, 12,  7,  7,  7, 12,  7, 12, 12,  7,
        6, 12, 12,  7, 12,  6,  7, 12,  6,  7,  7,  7, 12,  7, 12, 12, 12,
       12,  7,  7,  7,  7, 12,  7, 12, 12,  7,  7,  6,  7,  7, 12,  7, 12,
        7,  7, 12, 12, 12, 12, 12,  7,  7,  7,  7, 12, 12, 12, 12,  7,  7,
        6,  7,  7,  6,  7,  7,  7, 12, 12, 12, 12,  7,  6, 12, 12,  7, 12,
        7, 12,  7,  7,  7,  7,  7, 12,  7,  6,  7,  7,  7, 12,  6,  7, 12],
      dtype=int64))
187
```

Out[20]:

In [21]:
```
# 187 outliers are present in the df data frame
df_new=train_df[(z<3).all(axis=1)]
print(df_new.shape)
```

```
(10512, 14)
```

after removing the outliers the shape of the df is reduced to 10512*12

In [22]:
```
#df_new is the data frame after removing the outliers
df_new
```

Out[22]:

	Airline	Source	Destination	Route	Dep_Time	Total_Stops	Additional_Info	Price	Journey_date	Journey
0	3	0	5	18	211	4	8	3897	24	
1	1	3	0	84	31	1	8	7662	1	
2	4	2	1	118	70	1	8	13882	9	
3	3	3	0	91	164	0	8	6218	12	
4	3	0	5	29	149	0	8	13302	1	
	...	...	...	...	...	...	...	...	...	

	Airline	Source	Destination	Route	Dep_Time	Total_Stops	Additional_Info	Price	Journey_date	Journey
10678	0	3	0	64	183	4	8	4107	9	
10679	1	3	0	64	193	4	8	4145	27	
10680	4	0	2	18	58	4	8	7229	27	
10681	10	0	5	18	92	4	8	12648	1	
10682	1	2	1	108	85	1	8	11753	9	

10512 rows × 14 columns

In [23]:
```
df_new.columns
```

Out[23]:
```
Index(['Airline', 'Source', 'Destination', 'Route', 'Dep_Time', 'Total_Stops',
       'Additional_Info', 'Price', 'Journey_date', 'Journey_month',
       'Arrival_hour', 'Arrival_minute', 'Duration_hours', 'Duration_mins'],
      dtype='object')
```

In [24]:
```
# assigning the x and y values
x=df_new.loc[:,['Airline', 'Source', 'Destination', 'Route', 'Dep_Time', 'Total_Stops',
       'Additional_Info','Journey_date', 'Journey_month',
       'Duration_hours', 'Duration_mins']]
y=df_new.loc[:,['Price']]
print(x)
print(y)
```

```
       Airline  Source  Destination  Route  Dep_Time  Total_Stops  \
0            3       0            5     18       211            4
1            1       3            0     84        31            1
2            4       2            1    118        70            1
3            3       3            0     91       164            0
4            3       0            5     29       149            0
...        ...     ...          ...    ...       ...          ...
10678        0       3            0     64       183            4
10679        1       3            0     64       193            4
10680        4       0            2     18        58            4
10681       10       0            5     18        92            4
10682        1       2            1    108        85            1

       Additional_Info  Journey_date  Journey_month  Duration_hours  \
0                    8            24              3               2
1                    8             1              5               7
2                    8             9              6              19
3                    8            12              5               5
4                    8             1              3               4
...                ...           ...            ...             ...
10678                8             9              4               2
10679                8            27              4               2
10680                8            27              4               3
10681                8             1              3               2
10682                8             9              5               8

       Duration_mins
0                 50
1                 25
2                  0
3                 25
4                 45
...              ...
10678             30
10679             35
```

```
10680             0
10681            40
10682            20

[10512 rows x 11 columns]
       Price
0       3897
1       7662
2      13882
3       6218
4      13302
...      ...
10678   4107
10679   4145
10680   7229
10681  12648
10682  11753

[10512 rows x 1 columns]
```

In [25]:

```
#removing the skewness by yeo johnson method
from sklearn.preprocessing import power_transform
x=power_transform(x,method='yeo-johnson')
x
```

Out[25]:

```
array([[-0.30220487, -1.61243172,  1.77586442, ..., -1.42209562,
        -1.16677731,  1.19568332],
       [-1.42486321,  0.89471597, -1.36324041, ...,  0.13442623,
        -0.04032576, -0.10676208],
       [ 0.14066537, -0.02743332,  0.02103253, ...,  1.1712599 ,
         1.09945835, -1.87944638],
       ...,
       [ 0.14066537, -1.61243172,  0.7257265 , ..., -0.72902126,
        -0.84596872, -1.87944638],
       [ 2.1166493 , -1.61243172,  1.77586442, ..., -1.42209562,
        -1.16677731,  0.69843526],
       [-1.42486321, -0.02743332,  0.02103253, ...,  0.13442623,
         0.10130796, -0.3979608 ]])
```

In [26]:

```
# checking the skweness after removing the skewness by yeo-johnson method
pd.DataFrame(x).skew()
```

Out[26]:

```
0    -0.014479
1    -0.236472
2     0.041378
3    -0.434610
4    -0.152045
5     0.322537
6    -1.359457
7    -0.203290
8    -0.223729
9    -0.031337
10   -0.378064
dtype: float64
```

In [27]:

```
from sklearn.preprocessing import MinMaxScaler
mms=MinMaxScaler()
x1=mms.fit_transform(x)
```

In [28]:

```
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
```

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
from sklearn import metrics
```

In [30]:

```
model=[LinearRegression(),RandomForestRegressor()]
max_r2_score=0
for i_state in range(0,50):
    x_train,x_test,y_train,y_test=train_test_split(x1,y,random_state=i_state,test_size=0.3
    for a in model:
        a.fit(x_train,y_train)
        pred=a.predict(x_test)
        score=r2_score(y_test,pred)
        print('score for random_state',i_state,'is',score)
        if score>max_r2_score:
            max_r2_score=score
            Final_state=i_state
            Final_model= a
print('R2_score ',max_r2_score,'for random state ',Final_state, 'and model is ',Final_mode
```

```
score for random_state 0 is 0.49906133534672137
score for random_state 0 is 0.9240034447902414
score for random_state 1 is 0.4938943305471144
score for random_state 1 is 0.9189486503953768
score for random_state 2 is 0.4930029586369241
score for random_state 2 is 0.9234229589355837
score for random_state 3 is 0.49206399646152477
score for random_state 3 is 0.9203472380899715
score for random_state 4 is 0.47929349730141446
score for random_state 4 is 0.9197230410956991
score for random_state 5 is 0.49021295334905113
score for random_state 5 is 0.917999529100253
score for random_state 6 is 0.4949578761128225
score for random_state 6 is 0.9188802410797634
score for random_state 7 is 0.4954718045619626
score for random_state 7 is 0.9248091129453248
score for random_state 8 is 0.4972609592485323
score for random_state 8 is 0.9214501618077494
score for random_state 9 is 0.4957579300182152
score for random_state 9 is 0.9219570629306163
score for random_state 10 is 0.4859266705126203
score for random_state 10 is 0.9262781299232707
score for random_state 11 is 0.4745038684243108
score for random_state 11 is 0.9148417689866523
score for random_state 12 is 0.48473815800148645
score for random_state 12 is 0.9071499571209515
score for random_state 13 is 0.4720229722394904
score for random_state 13 is 0.9135641715290712
score for random_state 14 is 0.48856556878028234
score for random_state 14 is 0.9145217947457204
score for random_state 15 is 0.4933484806576166
score for random_state 15 is 0.9247272018302536
score for random_state 16 is 0.5013975353066706
score for random_state 16 is 0.9173569229022943
score for random_state 17 is 0.49154162427854564
score for random_state 17 is 0.923071990606004
score for random_state 18 is 0.4866153398602313
score for random_state 18 is 0.9174287016286744
score for random_state 19 is 0.48787511711654286
score for random_state 19 is 0.9165137884485328
score for random_state 20 is 0.5070803180794119
score for random_state 20 is 0.9228115848700219
score for random_state 21 is 0.4976538755920542
score for random_state 21 is 0.9248234946820624
```

```
score for random_state 22 is 0.48788111829230874
score for random_state 22 is 0.9234584455841109
score for random_state 23 is 0.5069692850041776
score for random_state 23 is 0.9164959970897364
score for random_state 24 is 0.4919261016273525
score for random_state 24 is 0.9146994571805984
score for random_state 25 is 0.49454749241594476
score for random_state 25 is 0.923780670175068
score for random_state 26 is 0.4906405245000083
score for random_state 26 is 0.919012941737946
score for random_state 27 is 0.4995919539597664
score for random_state 27 is 0.9189743872885999
score for random_state 28 is 0.5011177831788514
score for random_state 28 is 0.9188531027791196
score for random_state 29 is 0.4723751390539763
score for random_state 29 is 0.9221495602010508
score for random_state 30 is 0.4746711757423464
score for random_state 30 is 0.9128226057504317
score for random_state 31 is 0.49365716052877495
score for random_state 31 is 0.9091731006125463
score for random_state 32 is 0.4947182559163694
score for random_state 32 is 0.9108689911880709
score for random_state 33 is 0.4910142109833773
score for random_state 33 is 0.9187846291464673
score for random_state 34 is 0.5053141276403857
score for random_state 34 is 0.9237654481061437
score for random_state 35 is 0.4712381135100079
score for random_state 35 is 0.9139582559766167
score for random_state 36 is 0.4943751683170504
score for random_state 36 is 0.9139672936946605
score for random_state 37 is 0.4859229230155493
score for random_state 37 is 0.9199542544759272
score for random_state 38 is 0.47836403455939724
score for random_state 38 is 0.9141732449984653
score for random_state 39 is 0.46834826878910707
score for random_state 39 is 0.9165223598653599
score for random_state 40 is 0.5060321419405039
score for random_state 40 is 0.9254308492688753
score for random_state 41 is 0.4895671483483386
score for random_state 41 is 0.9163633741237269
score for random_state 42 is 0.5046558570246225
score for random_state 42 is 0.9268317811273927
score for random_state 43 is 0.4891802393930883
score for random_state 43 is 0.9138109254668876
score for random_state 44 is 0.4924532995957185
score for random_state 44 is 0.9125764812042424
score for random_state 45 is 0.4942076843750949
score for random_state 45 is 0.9143016442077225
score for random_state 46 is 0.49033099224279153
score for random_state 46 is 0.9198849470415653
score for random_state 47 is 0.49464245345537217
score for random_state 47 is 0.9191982731740098
score for random_state 48 is 0.5027151220333719
score for random_state 48 is 0.9208002325777033
score for random_state 49 is 0.495607387355814
score for random_state 49 is 0.9276432497616025
R2_score  0.9276432497616025 for random state  49 and model is  RandomForestRegressor()
```

In [31]:

```
# RandomForestRegressor() gives the good r2 score at randomstate 10
rf=RandomForestRegressor()
x_train,x_test,y_train,y_test=train_test_split(x1,y,random_state=49,test_size=0.33)
rf.fit(x_train,y_train)
rf.score(x_train,y_train)
=rf.predict(x_test)
```

```
rfs=r2_score(y_test,pred_decision)
print('r2_score =',rfs*100)
from sklearn.model_selection import cross_val_score
cv_score=cross_val_score(rf,x,y,cv=5)
cv_mean=cv_score.mean()
print("cross_val_score=",cv_mean*100)
```

```
r2_score = 92.80488798177338
cross_val_score= 92.30786716924693
```

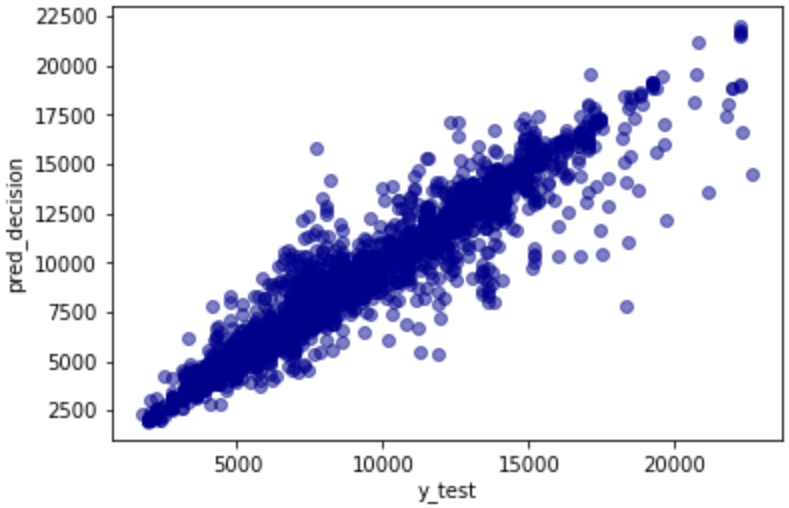
In [32]:

```
print("MAE:" , metrics.mean_absolute_error(y_test,pred_decision))
print("MSE:" , metrics.mean_squared_error(y_test,pred_decision))
print("RMSE:" , np.sqrt(metrics.mean_squared_error(y_test,pred_decision)))
```

```
MAE: 595.0883108679141
MSE: 1193117.675336339
RMSE: 1092.2992608879395
```

In [33]:

```
plt.scatter(y_test,pred_decision,alpha =0.5,color="DarkBlue")
plt.xlabel("y_test")
plt.ylabel("pred_decision")
plt.show()
```



For improving the model performance we will go for hyper parameter tuning

In [34]:

```
# importing the gridsearchcv for hyper parameter tuning
from sklearn.model_selection import GridSearchCV
# for selecting the best parameters
parameters={'criterion':['mse','mae'],'max_features':['auto','sqrt','log2']}
rf=RandomForestRegressor()
clf=GridSearchCV(rf,parameters)
clf.fit(x_train,y_train)
print(clf.best_params_)
```

```
{'criterion': 'mse', 'max_features': 'auto'}
```

In [35]:

```
# After selecting the best parameter we need to implement them on the algorithms for check
rf=RandomForestRegressor(criterion='mse',max_features='auto')
rf.fit(x_train,y_train)
rf.score(x_train,y_train)
pred_rf=rf.predict(x_test)
rfs=r2_score(y_test,pred_rf)
print('R2_score =',rfs*100)
cv_score=cross_val_score(rf,x1,y,cv=5)
```

```
cv_mean=cv_score.mean()
print('mean_cv value = ',cv_mean*100)
```

```
R2_score = 92.84479361471114
mean_cv value =  92.30078128293216
```

conclusion:

- we are getting the good results for RandomForestRegressor, accuracy score is 92.844 and cross validation score is 92.300, so we are considering the RandomForestRegressor for model performance
- saving the model by using pickel