In [1]:

```
#importig the libraries

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

#ignore harmless warnings

import warnings
warnings.filterwarnings("ignore")

#set to display all the columns in dataset
pd.set_option("display.max_columns",None)

#to run sql queries on DataFrame
import pandasql as psql
```

In [3]:

```
dp= pd.read_csv(r"C:\Users\Dlc\Downloads\Diamonds Prices2022.csv",header=0)
dp_bk=dp.copy()
dp.head(20)
```

Out[3]:

| | Unnamed: 0 | carat | cut | color | clarity | depth | table | price | x | y | z |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 0.23 | Ideal | E | SI2 | 61.5 | 55.0 | 326 | 3.95 | 3.98 | 2.43 |
| **1** | 2 | 0.21 | Premium | E | SI1 | 59.8 | 61.0 | 326 | 3.89 | 3.84 | 2.31 |
| **2** | 3 | 0.23 | Good | E | VS1 | 56.9 | 65.0 | 327 | 4.05 | 4.07 | 2.31 |
| **3** | 4 | 0.29 | Premium | I | VS2 | 62.4 | 58.0 | 334 | 4.20 | 4.23 | 2.63 |
| **4** | 5 | 0.31 | Good | J | SI2 | 63.3 | 58.0 | 335 | 4.34 | 4.35 | 2.75 |
| **5** | 6 | 0.24 | Very Good | J | VVS2 | 62.8 | 57.0 | 336 | 3.94 | 3.96 | 2.48 |
| **6** | 7 | 0.24 | Very Good | I | VVS1 | 62.3 | 57.0 | 336 | 3.95 | 3.98 | 2.47 |
| **7** | 8 | 0.26 | Very Good | H | SI1 | 61.9 | 55.0 | 337 | 4.07 | 4.11 | 2.53 |
| **8** | 9 | 0.22 | Fair | E | VS2 | 65.1 | 61.0 | 337 | 3.87 | 3.78 | 2.49 |
| **9** | 10 | 0.23 | Very Good | H | VS1 | 59.4 | 61.0 | 338 | 4.00 | 4.05 | 2.39 |
| **10** | 11 | 0.30 | Good | J | SI1 | 64.0 | 55.0 | 339 | 4.25 | 4.28 | 2.73 |
| **11** | 12 | 0.23 | Ideal | J | VS1 | 62.8 | 56.0 | 340 | 3.93 | 3.90 | 2.46 |
| **12** | 13 | 0.22 | Premium | F | SI1 | 60.4 | 61.0 | 342 | 3.88 | 3.84 | 2.33 |
| **13** | 14 | 0.31 | Ideal | J | SI2 | 62.2 | 54.0 | 344 | 4.35 | 4.37 | 2.71 |
| **14** | 15 | 0.20 | Premium | E | SI2 | 60.2 | 62.0 | 345 | 3.79 | 3.75 | 2.27 |
| **15** | 16 | 0.32 | Premium | E | I1 | 60.9 | 58.0 | 345 | 4.38 | 4.42 | 2.68 |
| **16** | 17 | 0.30 | Ideal | I | SI2 | 62.0 | 54.0 | 348 | 4.31 | 4.34 | 2.68 |
| **17** | 18 | 0.30 | Good | J | SI1 | 63.4 | 54.0 | 351 | 4.23 | 4.29 | 2.70 |
| **18** | 19 | 0.30 | Good | J | SI1 | 63.8 | 56.0 | 351 | 4.23 | 4.26 | 2.71 |
| **19** | 20 | 0.30 | Very Good | J | SI1 | 62.7 | 59.0 | 351 | 4.21 | 4.27 | 2.66 |

In [4]:

```
dp.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53943 entries, 0 to 53942
Data columns (total 11 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Unnamed: 0  53943 non-null  int64  
 1   carat       53943 non-null  float64
 2   cut         53943 non-null  object 
 3   color       53943 non-null  object 
 4   clarity     53943 non-null  object 
 5   depth       53943 non-null  float64
 6   table       53943 non-null  float64
 7   price       53943 non-null  int64  
 8   x           53943 non-null  float64
 9   y           53943 non-null  float64
 10  z           53943 non-null  float64
dtypes: float64(6), int64(2), object(3)
memory usage: 4.5+ MB
```

In [5]:

```
dp.columns
```

Out[5]:

```
Index(['Unnamed: 0', 'carat', 'cut', 'color', 'clarity', 'depth', 'table',
       'price', 'x', 'y', 'z'],
      dtype='object')
```

In [6]:

```
dp=dp.drop('Unnamed: 0',axis=1)
dp.columns=['Weight','Cut_Quality','Color','Clarity','Depth','Table','Price
','X_length','Y_width','Z_depth']
```

In [8]:

```
dp['Cut_Quality'].value_counts()
```

Out[8]:

```
Ideal        21551
Premium      13793
Very Good    12083
Good          4906
Fair          1610
Name: Cut_Quality, dtype: int64
```

In [9]:

```
dp['Color'].value_counts()
```

Out[9]:

```
G    11292
E     9799
F     9543
H     8304
D     6775
I     5422
J     2808
Name: Color, dtype: int64
```

In [10]:

```
dp['Clarity'].value_counts()
```

Out[10]:

```
SI1     13067
VS2     12259
SI2      9194
VS1      8171
VVS2     5066
VVS1     3655
IF       1790
I1        741
Name: Clarity, dtype: int64
```

In [11]:

```
dp.describe()
```

Out[11]:

| | Weight | Depth | Table | Price | X_length | Y_width | Z_depth |
|---|---|---|---|---|---|---|---|
| **count** | 53943.000000 | 53943.000000 | 53943.000000 | 53943.000000 | 53943.000000 | 53943.000000 | 53943.000000 |
| **mean** | 0.797935 | 61.749322 | 57.457251 | 3932.734294 | 5.731158 | 5.734526 | 3.538730 |
| **std** | 0.473999 | 1.432626 | 2.234549 | 3989.338447 | 1.121730 | 1.142103 | 0.705679 |
| **min** | 0.200000 | 43.000000 | 43.000000 | 326.000000 | 0.000000 | 0.000000 | 0.000000 |
| **25%** | 0.400000 | 61.000000 | 56.000000 | 950.000000 | 4.710000 | 4.720000 | 2.910000 |
| **50%** | 0.700000 | 61.800000 | 57.000000 | 2401.000000 | 5.700000 | 5.710000 | 3.530000 |
| **75%** | 1.040000 | 62.500000 | 59.000000 | 5324.000000 | 6.540000 | 6.540000 | 4.040000 |
| **max** | 5.010000 | 79.000000 | 95.000000 | 18823.000000 | 10.740000 | 58.900000 | 31.800000 |

DATA CLEANING¶

Remove Duplicates¶

In [12]:

```
dp[dp.duplicated()]
```

Out[12]:

| | Weight | Cut_Quality | Color | Clarity | Depth | Table | Price | X_length | Y_width | Z_depth |
|---|---|---|---|---|---|---|---|---|---|---|
| **1005** | 0.79 | Ideal | G | SI1 | 62.3 | 57.0 | 2898 | 5.90 | 5.85 | 3.66 |
| **1006** | 0.79 | Ideal | G | SI1 | 62.3 | 57.0 | 2898 | 5.90 | 5.85 | 3.66 |
| **1007** | 0.79 | Ideal | G | SI1 | 62.3 | 57.0 | 2898 | 5.90 | 5.85 | 3.66 |
| **1008** | 0.79 | Ideal | G | SI1 | 62.3 | 57.0 | 2898 | 5.90 | 5.85 | 3.66 |
| **2025** | 1.52 | Good | E | I1 | 57.3 | 58.0 | 3105 | 7.53 | 7.42 | 4.28 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **50079** | 0.51 | Ideal | F | VVS2 | 61.2 | 56.0 | 2203 | 5.19 | 5.17 | 3.17 |
| **52861** | 0.50 | Fair | E | VS2 | 79.0 | 73.0 | 2579 | 5.21 | 5.18 | 4.09 |
| **53940** | 0.71 | Premium | E | SI1 | 60.5 | 55.0 | 2756 | 5.79 | 5.74 | 3.49 |
| **53941** | 0.71 | Premium | F | SI1 | 59.8 | 62.0 | 2756 | 5.74 | 5.73 | 3.43 |
| **53942** | 0.70 | Very Good | E | VS2 | 60.5 | 59.0 | 2757 | 5.71 | 5.76 | 3.47 |

149 rows × 10 columns

In [13]:

```
dp.drop_duplicates(inplace=True)
```

In [14]:

```
dp.isnull().sum()
```

Out[14]:

```
Weight         0
Cut_Quality    0
Color          0
Clarity        0
Depth          0
Table          0
Price          0
X_length       0
Y_width        0
Z_depth        0
dtype: int64
```

## [Detect-Remove] Outliers¶

In [15]:

```
numeric_cols = ['Weight', 'Depth', 'Table', 'Price', 'X_length', 'Y_width',
```

```
'Z_depth']
plt.figure(figsize=(15, 15))
for i in range (7) :
    plt.subplot(3, 3, i+1)
    sns.boxplot(x=dp[numeric_cols[i]], color='#6DA59D')
    plt.title(numeric_cols[i])
plt.show()
```

In [16]:

```
def detect_outliers (data, column):
    q1 = dp[column].quantile (.25)
    q3= dp[column].quantile (.75)
    IQR = q3-q1
    lower_bound = q1-(1.5*IQR)
    upper_bound = q3+ (1.5*IQR)
    ls = dp.index [ (dp [column] < lower_bound) | (dp [column] >
upper_bound) ]
    return ls
```

In [17]:

```
index_list = []
for column in numeric_cols:
    index_list.extend( detect_outliers (dp, column))
# remove duplicated indices in the index_list and sort it
index_list = sorted(set(index_list))
```

In [18]:

```
before_remove = dp.shape
dp=dp.drop(index_list)
after_remove =dp.shape
print(f'''Shape of data before removing outliers: {before_remove}
Shape of data after remove : {after_remove}''')
```

```
Shape of data before removing outliers: (53794, 10)
Shape of data after remove : (47416, 10)
```

## Data Visualisation¶

In [19]:

```
cols = ['Weight', 'Depth', 'Table','X_length', 'Y_width', 'Z_depth']
plt.figure(figsize=(18, 12))
for i in range(6) :
    plt.subplot(2, 3, i+1)
    #sns.set()
    plt.scatter(dp[cols[i]], dp['Price'],color='#679C94')
    plt.title(cols[i])
    plt.ylabel('Price', size=13)
plt.show()
```

In [20]:

```
quality= dp.groupby('Cut_Quality').mean().sort_values('Price',
ascending=False)
quality=quality [['Price']].round(2)
quality.reset_index(inplace=True)
quality
```

Out[20]:

| | Cut_Quality | Price |
|---|---|---|
| **0** | Fair | 3701.98 |
| **1** | Premium | 3485.01 |
| **2** | Very Good | 3222.78 |
| **3** | Good | 3215.51 |
| **4** | Ideal | 2801.71 |

In [21]:

```
sns.barplot(x=quality[ 'Cut_Quality'], y= quality['Price'], palette='bone')
plt.show
```

Out[21]:

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

In [22]:

```
color= dp.groupby('Color').mean().sort_values('Price', ascending=False)
color=color[['Price']].round(2)
color.reset_index(inplace=True)
color
```

Out[22]:

| | Color | Price |
|---|---|---|
| **0** | J | 3895.46 |
| **1** | I | 3627.18 |
| **2** | H | 3507.42 |
| **3** | G | 3209.37 |
| **4** | F | 3069.83 |
| **5** | D | 2654.11 |
| **6** | E | 2588.76 |

In [23]:

```
sns.barplot(x= color[ 'Color'], y=color['Price'], palette='bone')
plt.show
```

Out[23]:

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

In [24]:

```
clarity = dp.groupby('Clarity').mean().sort_values('Price',
ascending=False)
clarity= clarity[['Price']].round(2)
clarity.reset_index(inplace=True)
clarity
```

Out[24]:

| | Clarity | Price |
|---|---|---|
| **0** | SI2 | 3760.29 |
| **1** | I1 | 3296.18 |
| **2** | SI1 | 3260.29 |
| **3** | VS1 | 3142.50 |
| **4** | VS2 | 3092.77 |
| **5** | VVS2 | 2782.27 |
| **6** | VVS1 | 2125.65 |
| **7** | IF | 2119.54 |

In [25]:

```
sns.barplot(x= clarity[ 'Clarity'], y=clarity['Price'], palette='bone')
plt.show
```

Out[25]:

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

Importing Libraries Of Machine Learning¶

In [26]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import Ridge
from sklearn. ensemble import RandomForestRegressor
from sklearn.metrics import r2_score
from sklearn.model_selection import cross_val_score
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import GridSearchCV
```

Data Preprocessing¶

Ordinary Encoding¶

In [27]:

```
dp['Cut_Quality'] = dp['Cut_Quality'].map({'Fair': 0, 'Good' :1, 'Very
Good' :2, 'Premium' :3, 'Ideal' :4})
dp['Color'] = dp['Color'].map({'J' :0, 'I':1, 'H':2, 'G' :3, 'F' :4, 'E':5,
'D': 6})
dp['Clarity'] = dp[ 'Clarity'].map({'I1': 0, 'SI2':1, 'SI1':2, 'VS2':3,
'VS1' :4, 'VVS2':5, 'VVS1' :6, 'IF':7})
```

In [28]:

```
dp.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 47416 entries, 0 to 53939
Data columns (total 10 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Weight       47416 non-null  float64
 1   Cut_Quality  47416 non-null  int64
 2   Color        47416 non-null  int64
 3   Clarity      47416 non-null  int64
 4   Depth        47416 non-null  float64
 5   Table        47416 non-null  float64
```

```
 6   Price         47416 non-null  int64
 7   X_length      47416 non-null  float64
 8   Y_width       47416 non-null  float64
 9   Z_depth       47416 non-null  float64
dtypes: float64(6), int64(4)
memory usage: 4.0 MB
```

In [29]:

```
dp.head(20)
```

Out[29]:

| | Weight | Cut_Quality | Color | Clarity | Depth | Table | Price | X_length | Y_width | Z_depth |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.23 | 4 | 5 | 1 | 61.5 | 55.0 | 326 | 3.95 | 3.98 | 2.43 |
| **1** | 0.21 | 3 | 5 | 2 | 59.8 | 61.0 | 326 | 3.89 | 3.84 | 2.31 |
| **3** | 0.29 | 3 | 1 | 3 | 62.4 | 58.0 | 334 | 4.20 | 4.23 | 2.63 |
| **4** | 0.31 | 1 | 0 | 1 | 63.3 | 58.0 | 335 | 4.34 | 4.35 | 2.75 |
| **5** | 0.24 | 2 | 0 | 5 | 62.8 | 57.0 | 336 | 3.94 | 3.96 | 2.48 |
| **6** | 0.24 | 2 | 1 | 6 | 62.3 | 57.0 | 336 | 3.95 | 3.98 | 2.47 |
| **7** | 0.26 | 2 | 2 | 2 | 61.9 | 55.0 | 337 | 4.07 | 4.11 | 2.53 |
| **9** | 0.23 | 2 | 2 | 4 | 59.4 | 61.0 | 338 | 4.00 | 4.05 | 2.39 |
| **10** | 0.30 | 1 | 0 | 2 | 64.0 | 55.0 | 339 | 4.25 | 4.28 | 2.73 |
| **11** | 0.23 | 4 | 0 | 4 | 62.8 | 56.0 | 340 | 3.93 | 3.90 | 2.46 |
| **12** | 0.22 | 3 | 4 | 2 | 60.4 | 61.0 | 342 | 3.88 | 3.84 | 2.33 |

| | Weight | Cut_Quality | Color | Clarity | Depth | Table | Price | X_length | Y_width | Z_depth |
|---|---|---|---|---|---|---|---|---|---|---|
| **13** | 0.31 | 4 | 0 | 1 | 62.2 | 54.0 | 344 | 4.35 | 4.37 | 2.71 |
| **14** | 0.20 | 3 | 5 | 1 | 60.2 | 62.0 | 345 | 3.79 | 3.75 | 2.27 |
| **15** | 0.32 | 3 | 5 | 0 | 60.9 | 58.0 | 345 | 4.38 | 4.42 | 2.68 |
| **16** | 0.30 | 4 | 1 | 1 | 62.0 | 54.0 | 348 | 4.31 | 4.34 | 2.68 |
| **17** | 0.30 | 1 | 0 | 2 | 63.4 | 54.0 | 351 | 4.23 | 4.29 | 2.70 |
| **18** | 0.30 | 1 | 0 | 2 | 63.8 | 56.0 | 351 | 4.23 | 4.26 | 2.71 |
| **19** | 0.30 | 2 | 0 | 2 | 62.7 | 59.0 | 351 | 4.21 | 4.27 | 2.66 |
| **20** | 0.30 | 1 | 1 | 1 | 63.3 | 56.0 | 351 | 4.26 | 4.30 | 2.71 |
| **21** | 0.23 | 2 | 5 | 3 | 63.8 | 55.0 | 352 | 3.85 | 3.92 | 2.48 |

In [30]:

```
IndepVar=[]
for col in dp.columns:
    if col!='Price':
        IndepVar.append(col)
TargetVar='Price'
x=dp[IndepVar]
y=dp[TargetVar]
```

In [31]:

```
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_sta
```

```
te=43)
x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

Out[31]:

```
((33191, 9), (14225, 9), (33191,), (14225,))
```

In [32]:

```
from sklearn.preprocessing import MinMaxScaler
mmscaler=MinMaxScaler(feature_range=(0,1))

x_train=mmscaler.fit_transform(x_train)
x_train=pd.DataFrame(x_train)

x_test=mmscaler.fit_transform(x_test)
x_test=pd.DataFrame(x_test)
```

In [34]:

```
#load the RGR data
RGRResults=pd.read_csv(r"C:\Users\Dlc\Downloads\Diamonds
Prices2022.csv",header = 0)
RGRResults.head()
```

Out[34]:

| | Unnamed: 0 | carat | cut | color | clarity | depth | table | price | x | y | z |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 0.23 | Ideal | E | SI2 | 61.5 | 55.0 | 326 | 3.95 | 3.98 | 2.43 |
| **1** | 2 | 0.21 | Premium | E | SI1 | 59.8 | 61.0 | 326 | 3.89 | 3.84 | 2.31 |
| **2** | 3 | 0.23 | Good | E | VS1 | 56.9 | 65.0 | 327 | 4.05 | 4.07 | 2.31 |
| **3** | 4 | 0.29 | Premium | I | VS2 | 62.4 | 58.0 | 334 | 4.20 | 4.23 | 2.63 |
| **4** | 5 | 0.31 | Good | J | SI2 | 63.3 | 58.0 | 335 | 4.34 | 4.35 | 2.75 |

In [35]:

```
# Build the Regression / Regressor models

from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import ExtraTreesRegressor
#from sklearn.neighbors import KNeighborsRegressor
#from sklearn.linear_model import BayesianRidge
#from sklearn.svm import SVR

# Create objects of Regression / Regressor models with default
hyper-parameters

ModelMLR = LinearRegression()
ModelDCR = DecisionTreeRegressor()
ModelRFR = RandomForestRegressor()
ModelETR = ExtraTreesRegressor()
#ModelKNN = KNeighborsRegressor(n_neighbors=5)
#ModelBRR = BayesianRidge()
#ModelSVR = SVR()

# Evalution matrix for all the algorithms
```

```
#MM = [ModelMLR, ModelDCR, ModelRFR, ModelETR, ModelKNN, ModelBRR,
ModelSVR]
MM = [ModelMLR, ModelDCR, ModelRFR, ModelETR]

for models in MM:

    # Fit the model with train data

    models.fit(x_train, y_train)

    # Predict the model with test data

    y_pred = models.predict(x_test)

    # Print the model name

    print('Model Name: ', models)

    # Evaluation metrics for Regression analysis

    from sklearn import metrics

    print('Mean Absolute Error (MAE):',
round(metrics.mean_absolute_error(y_test, y_pred),3))
    print('Mean Squared Error (MSE):',
round(metrics.mean_squared_error(y_test, y_pred),3))
    print('Root Mean Squared Error (RMSE):',
round(np.sqrt(metrics.mean_squared_error(y_test, y_pred)),3))
    print('R2_score:', round(metrics.r2_score(y_test, y_pred),6))
    print('Root Mean Squared Log Error (RMSLE):',
round(np.log(np.sqrt(metrics.mean_squared_error(y_test, y_pred))),3))

    # Define the function to calculate the MAPE - Mean Absolute Percentage
Error

    def MAPE (y_test, y_pred):
        y_test, y_pred = np.array(y_test), np.array(y_pred)
        return np.mean(np.abs((y_test - y_pred) / y_test)) * 100

    # Evaluation of MAPE

    result = MAPE(y_test, y_pred)
    print('Mean Absolute Percentage Error (MAPE):', round(result, 2), '%')
```

```
    # Calculate Adjusted R squared values

    r_squared = round(metrics.r2_score(y_test, y_pred),6)
    adjusted_r_squared = round(1 -
(1-r_squared)*(len(y)-1)/(len(y)-x.shape[1]-1),6)
    print('Adj R Square: ', adjusted_r_squared)

print('------------------------------------------------------------
----------------------------------------')

#------------------------------------------------------------
-----------------
    new_row = {'Model Name' : models,
               'Mean_Absolute_Error_MAE' :
metrics.mean_absolute_error(y_test, y_pred),
               'Adj_R_Square' : adjusted_r_squared,
               'Root_Mean_Squared_Error_RMSE' :
np.sqrt(metrics.mean_squared_error(y_test, y_pred)),
               'Mean_Absolute_Percentage_Error_MAPE' : result,
               'Mean_Squared_Error_MSE' :
metrics.mean_squared_error(y_test, y_pred),
               'Root_Mean_Squared_Log_Error_RMSLE':
np.log(np.sqrt(metrics.mean_squared_error(y_test, y_pred))),
               'R2_score' : metrics.r2_score(y_test, y_pred)}
    Results = RGRResults.append(new_row, ignore_index=True)

#------------------------------------------------------------
-----------------
```

```
Model Name:  LinearRegression()
Mean Absolute Error (MAE): 1118.887
Mean Squared Error (MSE): 1751705.406
Root Mean Squared Error (RMSE): 1323.52
R2_score: 0.765354
Root Mean Squared Log Error (RMSLE): 7.188
Mean Absolute Percentage Error (MAPE): 89.78 %
Adj R Square:  0.765309
-----------------------------------------------------------------------------
```

```
------------------------------
Model Name:  DecisionTreeRegressor()
Mean Absolute Error (MAE): 330.251
Mean Squared Error (MSE): 359188.121
Root Mean Squared Error (RMSE): 599.323
R2_score: 0.951886
Root Mean Squared Log Error (RMSLE): 6.396
Mean Absolute Percentage Error (MAPE): 9.89 %
Adj R Square:  0.951877
------------------------------------------------------------------------------
------------------------------
Model Name:  RandomForestRegressor()
Mean Absolute Error (MAE): 224.831
Mean Squared Error (MSE): 159739.578
Root Mean Squared Error (RMSE): 399.674
R2_score: 0.978602
Root Mean Squared Log Error (RMSLE): 5.991
Mean Absolute Percentage Error (MAPE): 7.13 %
Adj R Square:  0.978598
------------------------------------------------------------------------------
------------------------------
Model Name:  ExtraTreesRegressor()
Mean Absolute Error (MAE): 293.259
Mean Squared Error (MSE): 194338.139
Root Mean Squared Error (RMSE): 440.838
R2_score: 0.973968
Root Mean Squared Log Error (RMSLE): 6.089
Mean Absolute Percentage Error (MAPE): 12.1 %
Adj R Square:  0.973963
------------------------------------------------------------------------------
------------------------------
```

In [ ]: