Diamond Price Prediction Model

A. Data Preprocessing

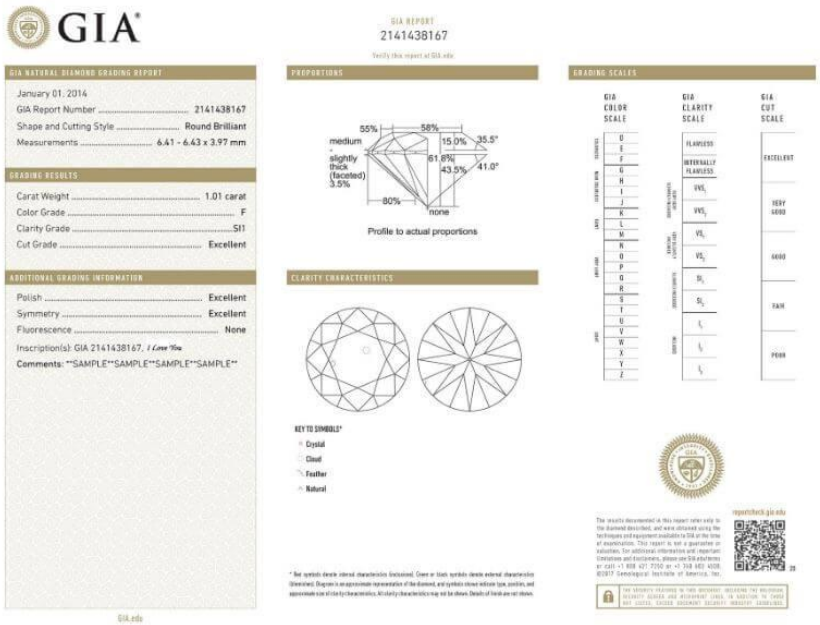
a. Converting text data to numeric data
b. Handling categorical variables
c. Handling ordinal variables
d. Handling outliers
e. Filling Missing Values
f. Extracting information from the text data

B. Machine Learning Modeling

a. Feature Engineering
b. Feature Scaling
a. Feature Selection
b. Experimenting and training the data with various algorithms
c. Evaluating performance Metrics

The research for Diamond Pricing has been done from these websites:

- www.gia.edu
- www.diamonds.pro
- Online Diamond Calculators



A. Data Preprocessing

You can go through the entire code for the preprocessing of the data in the 'Data Preprocessing Diamond Price Prediction.ipynb' file

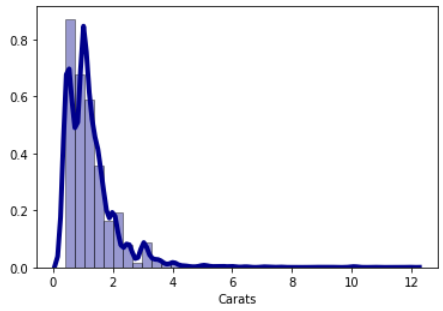
- The training data consists of various types of variables:

- **Continuous numeric variables**: Carats, Depth, Table
- **Categorical variables**: Regions, Vendor
- **Ordinal variables**: Cert, Color, Cut, Clarity, Known_Conflict_Diamond, Polish, Symmetry, Fluorescence

- The training data was divided at the very beginning into training and test data in order to avoid data leakage while performing the preprocessing and feature engineering on the data.
- The independent variables are of various data types like int64, float64 and object(text) data type.
- The variables needed to be converted into numeric data types in order to be used for training of the statistical models.
- The Price and Retail variables were converted from object data type to numeric data types by removing the '**$**' characters.
- The variable 'id' was dropped from the data as there is no significance of this variable for predicting the diamond price.

Carats:

The variable 'Carats' doesn't have any missing values. The distribution plot below shows that there are outliers in the variable but while checking upon the outliers in this variable, the outliers don't need to be treated because the data isn't incorrect as these are the diamonds having high carat weight and high price.



Cert:

- Certification also affects the diamond price.The diamonds which are not certified cost less than the diamonds that are certified.
- The 'Cert' variable consists of missing values. While going through the data on the excel sheet I filtered the data where the value was missing for the 'Cert' variable.
- I observed that diamonds having the same values in the rest of the variables but which are certified have higher price than the diamonds who have the same values but no certification.

176	0.4	NA	SI2	E		62.2	Medium	FALSE	4.72x4.74x		Russia	Round		57	1	$630	$1,080	6.44572	6.984716
177	0.4	AGSL	SI2	E		63.4	None	FALSE	4.62x4.64x		Russia	Round		56	1	$800	$1,320	6.684612	7.185387
178	0.4	NA	SI2	E		63.2	None	FALSE	4.65x4.69x		Canada	Cushion		57	1	$720	$1,220	6.579251	7.106606
179	0.4	GemEx	SI2	E		NA	NA	FALSE	4.66x4.7x2		Russia	Emerald		NA	2	$800	$1,225	6.684612	7.110696

- This can be observed by looking at the image below. These diamonds have identical features but are priced differently based on the fact that whether they are certified or not.

The missing values have been filled with 'No Cert' as these are the diamonds that do not have any certification.

```
X_train['Cert'].fillna('No Cert', inplace = True)
```

```
X_train['Cert'].value_counts()
```

```
AGSL       5261
No Cert     395
GemEx       381
Name: Cert, dtype: int64
```

These text values are then converted into numerical values. The certified diamonds have the value 1 and non certified diamonds have value 0 for the 'Cert' variable.

The 'Cert' variable is being treated as an ordinal variable having the order,

Cert - 1

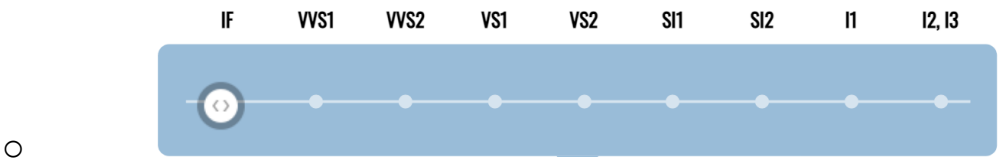
No Cert - 0

As diamonds that are certified have higher price than the diamonds that do not have any certification.

The biggest factor in diamond pricing are the Four Cs — cut quality, clarity, color and carat weight. The better a diamond's four Cs (or, in the case of carat weight, the higher) the more expensive it will be. In short, the better a diamond's quality, the more you'll need to pay to purchase it.

Clarity:

Diamond Clarity Grading Scale by GIA



The most expensive diamonds are on the extreme left side of this scale and as we go futher on the right side of the scale the diamond price decreases.

Ordinal Encoding - Clarity

'Clarity' is an ordinal variable and the labels are ordered from 0 to 10, 10 being the highest.

The label 10 is given for the FL (Flawless) diamonds and the label 0 is given for the Inclusions 3 (I3) diamonds. Here the text values of the 'Clarity' variable have been converted into numerical values.

As these labels have an order,the 'Clarity' variable is an Ordinal Variable.

This is the order for the labels

'I3' - 0, 'I2' - 1, 'I1' - 2,
'SI2' - 3, 'SI1' - 4,

'VS2' - 5, 'VS1' - 6, 'VVS2' - 7,

'VVS1' - 8, 'IF' - 9, 'FL' - 10

```
X_train['Clarity'].replace({'I3':0,
                            'I2':1,
                            'I1':2,
                            'SI2':3,
                            'SI1':4,
                            'VS2':5,
                            'VS1':6,
                            'VVS2':7,
                            'VVS1':8,
                            'IF':9,
                            'FL':10
                           }, inplace=True)
```

Color:

- This is how the diamonds are labeled based on their color. The diamonds that are colorless are the most expensive and thediamonds that are yellowish in color are less expensive.
- This is the grading scale for the diamonds.The 'Color' variable is an ordinal variable as the labels are ordered. Label 'D' being at the top of the grading scale and label 'Z' being at the bottom of the grading scale.
- There are many labels which were part of the 'F' label like 'Ffcdbrown', 'Fvyellow' etc. These labels had to be converted to 'F' label. Similarly for other labels.

This variable was then converted into a numeric variable using the order given below. This order for the labels have taken from the GIA Diamond Color Scale.

Label 'D' having the highest order 17 and lbel 'W' having the least order 1.

```
# Replacing the ordinal values
X_train['Color'].replace({'W':1,
                          'U':2,
                          'S':3,
                          'Q':4,
                          'P':5,
                          'O':6,
                          'N':7,
                          'M':8,
                          'L':9,
                          'K':10,
                          'J':11,
                          'I':12,
                          'H':13,
                          'G':14,
                          'F':15,
                          'E':16,
                          'D':17
                         }, inplace=True)
```

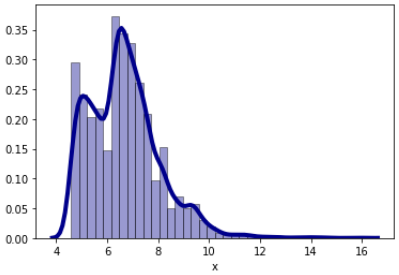
Measurements:

- The length, width and depth values of the diamond had to be extracted from the 'Measurements' variable using the code given below.

These are the new variables x,y,z added to the dataset and extracted from the 'Measurements' variable.

```
X_train[['x','y','z']].head()
```

	x	y	z
7612	9.54	9.58	5.93
2789	6.24	6.28	3.74
3381	7.14	7.17	4.45
7724	7.31	7.34	4.49
6361	6.94	6.89	4.32



- You can see below that these values (for variables x,y,z) are at the very end of the graph above are not outliers.This is based on the Carats weight for the diamonds.
- Hence I have decided to not treat these values as they are not outliers.This is the data of the bigger diamonds (large carat weight).

Known_conflict_diamond:

- This is a categorical variable which gives the information about whether the diamond was acquired through a conflict or not. As this is a categorical variable it can be taken care of using Label encoding or dummy variables.
- The new dummy variable created is 'Known_Conflict_Diamond_True' which consists of values 0 and 1
  True - 1
  False – 0

Shape:

Diamond Price by Shape

The shape is an important consideration when buying a diamond as it directly influences price. Round diamonds, the most popular diamond shape, tend to be priced higher than other shapes (referred to as fancy shapes) because of market demand, increased manufacturing costs, and their incredible brilliance.

After doing some research on how the diamond shape affects the diamond price I have categorized the diamonds in this manner.Round Shaped diamonds are the most popular and hence the most expensive. The uncut diamonds are relatively cheaper. The fancy diamonds are expensive but less expensive than the diamonds haveing 'Round' shape.

The 'Shape' variable is being treated as an ordinal variable with the 'Round' shape diamonds been given the highest order and the uncut diamonds given the least order.

Uncut - 0

Other Fancy Shapes - 1

Round - 2

17]:
```
X_train['Shape'].replace({'Round':2,
                          'Princess':1,
                         'Marquise':1,
                         'Oval':1,
                          'Emerald':1,
                          'Radiant':1,
                          'Pear':1,
                          'Asscher':1,
                          'Cushion':1,
                          'Uncut':0
                        }, inplace=True)
```

Regions:

- The variable 'Regions' is a categorical variable. I used pd.get_dummies to create dummy variables.

There are 9 unique labels in the 'Regions' variable. 8 new variables have been created and the 'Regions' variables has been dropped. The 8 new variables are:-

Region_Australia

Region_Botswana

Region_Canada

Region_DR Congo

Region_Other/Unknown

Region_Russia

Region_South Africa

Region_Zimbabwe

```
X_test.head(3)
```

onflict_Diamond_True	Region_Australia	Region_Botswana	Region_Canada	Region_DR Congo	Region_Other/Unknown	Region_Russia	Region_South Africa	Region_Zimbabwe
0	1	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0

Vendors

The variable 'Vendors' is a categorical variable. I used pd.get_dummies to apply dummy variables.

There are 4 unique labels in the 'Vendors' variable. 3 new variables have been created and the 'Vendors' variables has been dropped.

The 3 new variables are:

Vendor_2

Vendor_3

Vendor_4

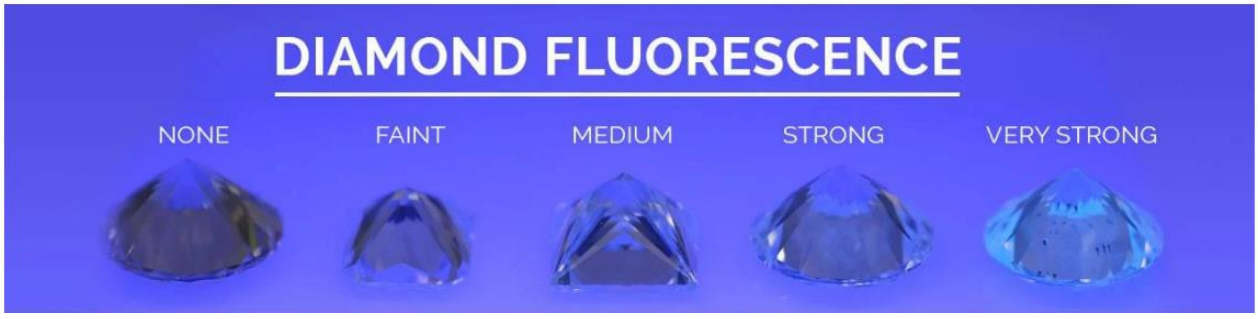
```
X_train[['Vendor_2','Vendor_3','Vendor_4']].head(3)
```

	Vendor_2	Vendor_3	Vendor_4
7612	1	0	0
2789	1	0	0
3381	1	0	0

Fluorescence:

Diamond Fluorescence

Fluorescence is the tendency of a diamond to emit a (soft) glow when exposed to ultraviolet light (UV light). The *fluorescence effect* is present in over 30% of diamonds and is an important consideration when buying a loose diamond.



- By doing some research about how the fluorescence affects diamond price it can be concluded that fluorescence is an ordinal variable as the labels have an order.
- **None > Faint > Medium > Strong > Very Strong.**
- Diamonds having no fluorescence are expensive in comparison with diamonds that do have fluorescence. The diamond price decreases as fluorescence increases.
- The missing values in the Fluorescence variable are 871.
- These missing values are filled by the label 'None' as it the most frequent label and the number of diamonds having 'None' label for the fluorescence variable are significantly more than the other labels.
- Hence these missing values were filled with the 'None' label.

As Fluroescence is an ordinal variable the labels are given an order

None - 4

Faint - 3

Medium - 2

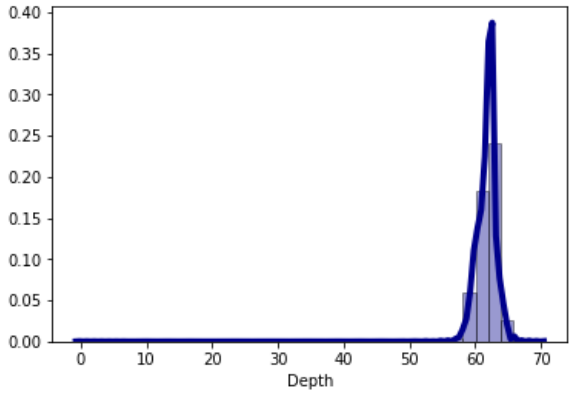
Strong - 1

Very Strong - 0

```
X_test['Fluroescence'].replace({'None':4,
                                'Faint':3,
                                'Medium':2,
                                'Strong':1,
                                 'Very Strong':0
                           }, inplace=True)
```

Depth:

- The Depth variable is a continuous variable. It has 1061 missing values.



The distribution plot of the Depth variable shows that it has some outliers. These missing values and outliers will be handled in the next steps.

Here I will be filling the values (nan and zero) of the Depth variable from the Depth1 variable that was custom made by using the formula and information given above.

I came across this formula after doing research on the depth percentage of diamond

Depth : total depth percentage = z / mean(x,y) = 2 * z / (x + y)

```
X_train['Depth1'] = round(((X_train['z']/((X_train['x']+ X_train['y'])/2))*100), 1)
```

```
X_test['Depth1'] = round(((X_test['z']/((X_test['x']+ X_test['y'])/2))*100), 1)
```

A new variable 'Depth1' was created where the values were filled by the x,y,z values extracted from the 'Measurements' variable.The values from the 'Depth1' variable were then used to fill the missing values in the 'Depth' variable

```
X_train['Depth'].fillna(X_train['Depth1'], inplace=True)
```

```
X_test['Depth'].fillna(X_test['Depth1'], inplace=True)
```

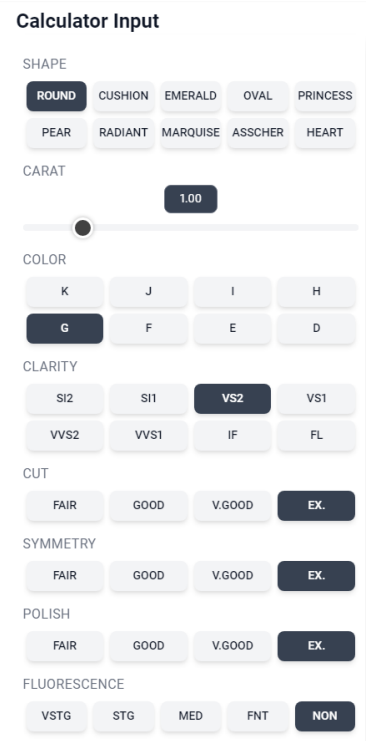
```
X_train['Depth'].isnull().sum()
```

```
0
```

Here it can be observed that the missing values have been handled

- Another method for filling the missing values is by predicting the missing values. Here the missing values of the Depth variable can be predicted using other variables that are correlated with the Depth variable. The correlation between continuous variables and categorical variables can be found out using the Anova Test.
- Here the method of filling the missing values using the Depth1 variable using x,y,z values proved to be extremely accurate, which led me to use this method for filling the missing values, rather than predicting the missing values.

Diamond Price Calculator



Polish:

The Polish variable doesn't have any null values. According to my research and after going through many diamond calculators, the Polish variable has 4 labels.

Excellent > Very Good > Good > Fair

The diamonds having Excellent Polish are the most expensive and the price of the diamond goes down as the Polish level moves to the right side of the scale.

```
X_train['Polish'].value_counts()
```

```
Excellent    4045
Very good    1186
              670
Good          136
Name: Polish, dtype: int64
```

The blank spaces in the polish variable are 670

Symmetry:

- The variable 'Symmetry' is an ordinal variable as the labels have an order.
- Excellent > Very Good > Good > Fair
- It can be observed that the rows where the data is blank for the 'Polish' are also the rows where the data is blank for the 'Symmetry' variable and the 'Cut' variable.
- Hence I have decided to delete these rows as the values from the 'Polish', 'Cut' and 'Symmetry' variable are missing entirely.

Now it can be observed that the blank values are not present and the ordinal variables can be converted by giving order to the labels of the Polish variable.

Excellent - 3

Very Good - 2

Good - 1

Fair - 0

```
X_train['Polish'].unique()
```

```
array(['Excellent', 'Very good', 'Good'], dtype=object)
```

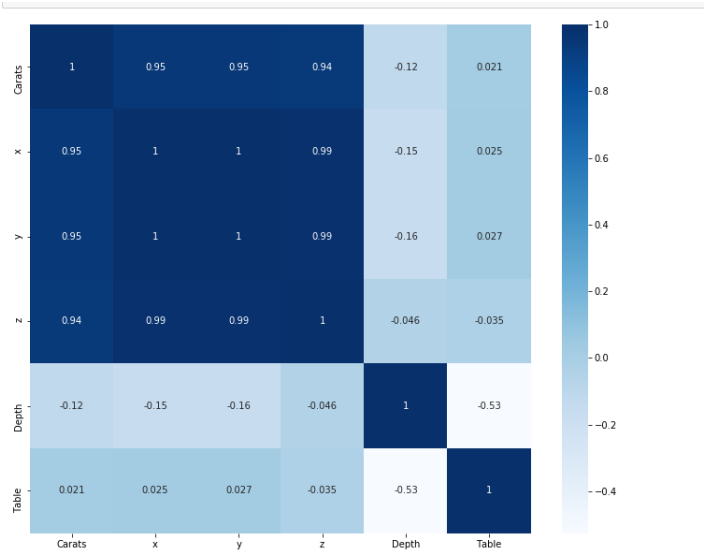
```
X_train['Polish'].replace({'Excellent':3,
                           'Very good':2,
                           'Good':1,
                           'Fair':0
                          }, inplace=True)
```

```
X_train['Polish'].value_counts()
```

```
3    4045
2    1186
1     136
Name: Polish, dtype: int64
```

Table:

- Table is a continuous variable with 1674 null values.
- The correlation matrix was plotted to check the correlation between Table and other continuous variables.
- The correlation matrix shows that Table doesn't have any correlation with other continuous variables.



- The Anova Test was performed to check whether there is any correlation between the Table variable and the other categorical variable.

```
##### ANOVA Results #####

Cut is NOT correlated with Table | P-Value: nan
Color is NOT correlated with Table | P-Value: nan
Clarity is NOT correlated with Table | P-Value: nan
Cert is NOT correlated with Table | P-Value: nan
Fluroescence is NOT correlated with Table | P-Value: nan
Shape is NOT correlated with Table | P-Value: nan
Polish is NOT correlated with Table | P-Value: nan
Symmetry is NOT correlated with Table | P-Value: nan

[]
```

I wanted to fill the missing values of the Table variable using the prediction method, but it can be seen by the correaltion matrix and the Anova test that none of the variables are correlated with the Table variable.

Hence none of the variable can be used for the prediction method for filling the missing values of the Table variable.

The option of filling the missing values using prediction method is unavailable.

- As the number of missing values are huge and they can't be filled by prediction method or any other formula like the way the variable Depth was filled.
- I decided to drop the variable while building the statistical models.

Cut:

- The Cut variable doesn't have any null values, but it does a huge number of have blank spaces.
- There are 2274 values in the 'Cut' variable of the training dataset. Hence I decided to drop this variable, as the number of missing values are significantly high.

B. Machine Learning Modeling:

Feature Engineering

I performed Log transformation on the continuous variables to transform the data to normality.

Log Normal Distribution

```
#numerical features

num_features = ['Carats', 'Depth', 'x','y','z']

for feature in num_features:
    X_train[feature]=np.log(X_train[feature])
```

```
num_features = ['Carats', 'Depth', 'x','y','z']

for feature in num_features:
    X_test[feature]=np.log(X_test[feature])
```

```
num_features = ['Carats', 'Depth', 'x','y','z']

for feature in num_features:
    offers[feature]=np.log(offers[feature])
```

```
X_train[['Carats', 'Depth', 'x','y','z']].head(3)
```

	Carats	Depth	x	y	z
7612	1.208960	4.127134	2.255493	2.259678	1.780024
7724	0.405465	4.115780	1.989243	1.993339	1.501853
6361	0.246860	4.133565	1.937302	1.930071	1.463255

Feature Scaling

As the features have different magnitudes, scaling the features ensured that the features were on the same scale and none of the variables had more impact than the other.

```
from sklearn.preprocessing import MinMaxScaler

scaler=MinMaxScaler()
```

```
X_train = scaler.fit_transform(X_train)
```

```
X_train = pd.DataFrame(X_train, columns=['Carats', 'Cert', 'Clarity', 'Color', 'Depth', 'Fluroescence', 'Polish','Shape', 'Symmet
```

```
X_test = scaler.fit_transform(X_test)
```

```
X_test = pd.DataFrame(X_test, columns=['Carats', 'Cert', 'Clarity', 'Color', 'Depth', 'Fluroescence', 'Polish','Shape', 'Symmetry
```

```
offers = scaler.fit_transform(offers)
```

```
offers = pd.DataFrame(offers, columns=['Carats', 'Cert', 'Clarity', 'Color', 'Depth', 'Fluroescence', 'Polish','Shape', 'Symmetry
```

Feature Selection:

- I implemented 2 methods for Feature Selection
    - Lasso Regression and SelectFromModel
    - Correlation Matrix and the Anova Test.

- Implementing the Machine Learning Algorithms after using both these methods, Method 1 proved to be better.
- Using Method 1 for feature selection, all the variables were used for model building.

Method 1

```
from sklearn.linear_model import Lasso
from sklearn.feature_selection import SelectFromModel
```

```
feature_sel_model = SelectFromModel(Lasso(alpha=0.005, random_state=0)) # remember to set the seed, the random state in this func
feature_sel_model.fit(X_train, y_train)
```

```
D:\ANA\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:531: ConvergenceWarning: Objective did not converge. You m
ight want to increase the number of iterations. Duality gap: 1853777087423.45, tolerance: 670574746.8942734
  positive)
```

```
SelectFromModel(estimator=Lasso(alpha=0.005, random_state=0))
```

```
feature_sel_model.get_support()
```

```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True])
```

```
selected_feat = X_train.columns[(feature_sel_model.get_support())]
```

```
selected_feat
```

```
Index(['Carats', 'Cert', 'Clarity', 'Color', 'Depth', 'Fluroescence', 'Polish',
       'Shape', 'Symmetry', 'x', 'y', 'z', 'Known_Conflict_Diamond_True',
       'Region_Australia', 'Region_Botswana', 'Region_Canada',
       'Region_DR Congo', 'Region_Other/Unknown', 'Region_Russia',
       'Region_South Africa', 'Region_Zimbabwe', 'Vendor_2', 'Vendor_3',
       'Vendor_4'],
      dtype='object')
```

Method 2

```
##### ANOVA Results #####

Carats is correlated with Price | P-Value: 0.0
Cert is NOT correlated with Price | P-Value: 0.5481011887765381
Clarity is correlated with Price | P-Value: 1.3023350656985665e-61
Color is correlated with Price | P-Value: 1.1598263335742668e-07
Depth is correlated with Price | P-Value: 2.7926578136664357e-14
Fluroescence is NOT correlated with Price | P-Value: 0.549346716712695
Polish is correlated with Price | P-Value: 1.1511569309474373e-18
Shape is NOT correlated with Price | P-Value: 0.5092367834228739
Symmetry is correlated with Price | P-Value: 2.8798870867374225e-24
x is correlated with Price | P-Value: 0.0
y is correlated with Price | P-Value: 0.0
z is correlated with Price | P-Value: 0.0
Known_Conflict_Diamond_True is NOT correlated with Price | P-Value: 0.31576482414233636
Region_Australia is NOT correlated with Price | P-Value: 0.5783814827420218
Region_Botswana is NOT correlated with Price | P-Value: 0.6442707219215673
Region_Canada is NOT correlated with Price | P-Value: 0.6403376843881223
Region_DR Congo is NOT correlated with Price | P-Value: 0.8283573702027429
Region_Other/Unknown is NOT correlated with Price | P-Value: 0.7819802825014366
Region_Russia is NOT correlated with Price | P-Value: 0.3715525210839806
Region_South Africa is NOT correlated with Price | P-Value: 0.7802828204524481
Region_Zimbabwe is NOT correlated with Price | P-Value: 0.38356592904541065
Vendor_2 is correlated with Price | P-Value: 2.818962195237454e-104
Vendor_3 is correlated with Price | P-Value: 3.083282891773386e-05
Vendor_4 is NOT correlated with Price | P-Value: 0.8695663618730887

['Carats',
 'Clarity',
 'Color',
 'Depth',
 'Polish',
 'Symmetry',
 'x',
 'y',
 'z',
 'Vendor_2',
 'Vendor_3']
```

Machine Learning Algorithms:

The Machine Learning Modeling code is present in the 'Machine Learning Model – Diamond Price Prediction.ipynb'. As the file is 100 mb please open it in google colab.

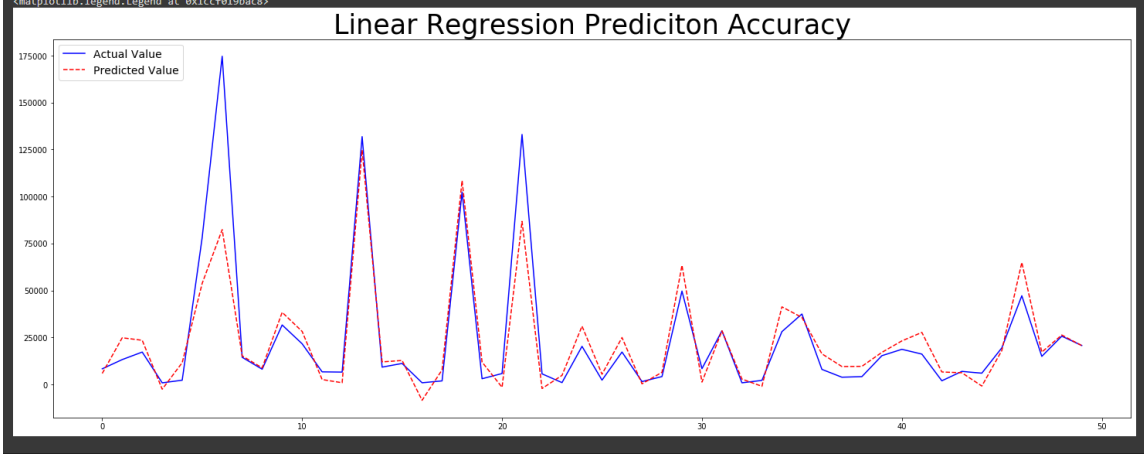
I implemented 4 Machine Learning Algorithms to predict the diamond prices and populate the offers dataset.

- Linear Regression
- Decision Tree
- Random Forest
- K-Nearest Neighbors

Linear Regression

```
x_axis= list(range(0, 50, 1))
plt.figure(figsize=(25,9))
plt.title("Linear Regression Prediciton Accuracy", fontsize=35)
plt.plot(x_axis, y_test[:50], 'b-', label="Actual Value")
plt.plot(x_axis, prediction[:50], 'r--', label="Predicted Value")
plt.legend(loc="upper left", fontsize=14)
```

```
<matplotlib.legend.Legend at 0x1ccf019bac8>
```




```
print("MAE: ",mae)
print("Mean squared error: ", mse)
print("Root Mean squared error: ", rmse)
print("Root Mean squared Log error: ", rmsle)

print("R^2 Score:", r2_score)
print("Adjusted R^2 Score:", adj_r2_score)
```

```
MAE:  8407.577479851654
Mean squared error:  726356610.6908556
Root Mean squared error:  26951.00389022375
Root Mean squared Log error:  10.201775825744443
R^2 Score: 0.6926264261761343
Adjusted R^2 Score: 0.6884325854872357
```

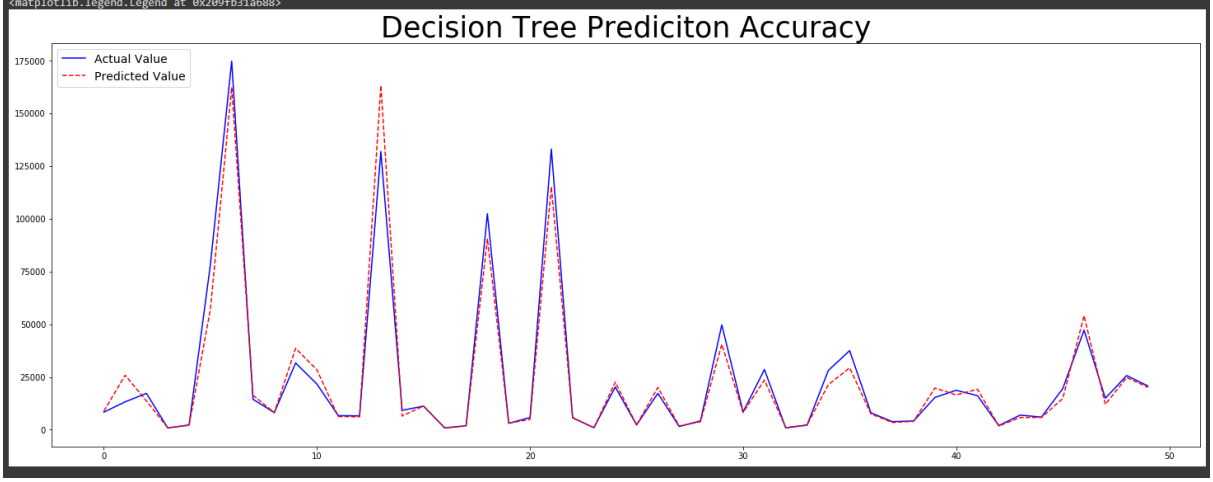
These algorithms were compared using the metrics given above.

- Mean Absolute Error
- Mean Squared Error
- Root Mean Squared Error
- Root Mean Squared Log Error
- R Squared Value
- Adjusted R Squared Value

Decision Tree

```
x_axis= list(range(0, 50, 1))
plt.figure(figsize=(25,9))
plt.title("Decision Tree Prediciton Accuracy", fontsize=35)
plt.plot(x_axis, y_test[:50], 'b-', label="Actual Value")
plt.plot(x_axis, prediction2[:50], 'r--', label="Predicted Value")
plt.legend(loc="upper left", fontsize=14)
```

```
<matplotlib.legend.Legend at 0x209fb31a688>
```



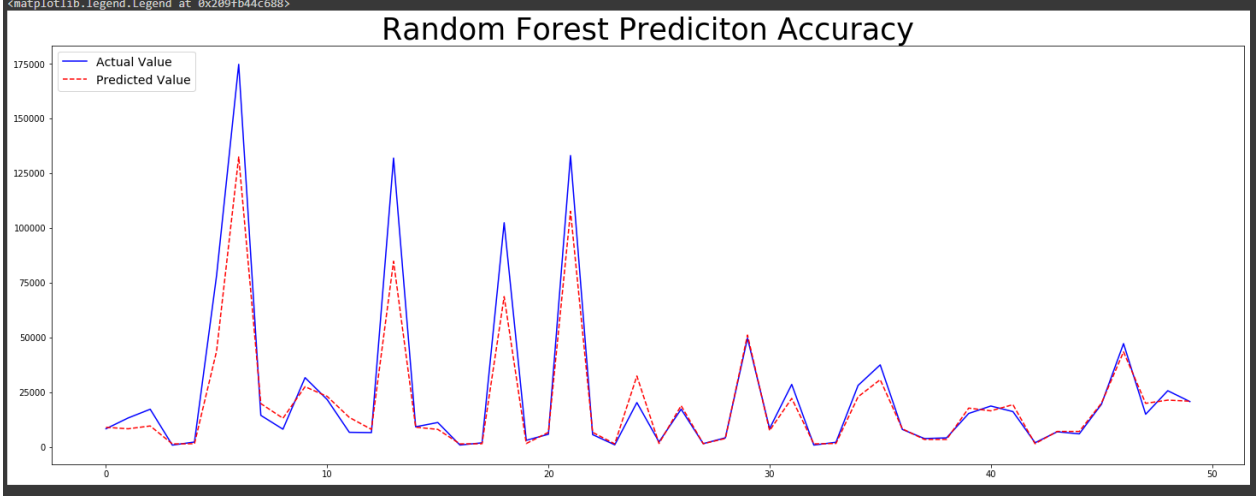

```
print("Mean Absolute Error: ",mae2)
print("Mean squared error: ", mse2)
print("Root Mean squared error: ", rmse2)
print("Root Mean squared Log error: ", rmsle2)
print("R^2 Score:", r2_score2)
print("Adjusted R^2 Score:", adj_r2_score2)
```

```
Mean Absolute Error:  3664.5982042513942
Mean squared error:  475400954.19419986
Root Mean squared error:  21803.69129744319
Root Mean squared Log error:  9.989834560014012
R^2 Score: 0.9960511026421581
Adjusted R^2 Score: 0.9959972234286344
```

Random Forest

```
x_axis= list(range(0, 50, 1))
plt.figure(figsize=(25,9))
plt.title("Random Forest Prediciton Accuracy", fontsize=35)
plt.plot(x_axis, y_test[:50], 'b-', label="Actual Value")
plt.plot(x_axis, prediction3[:50], 'r--', label="Predicted Value")
plt.legend(loc="upper left", fontsize=14)
```

```
<matplotlib.legend.Legend at 0x209fb44c688>
```



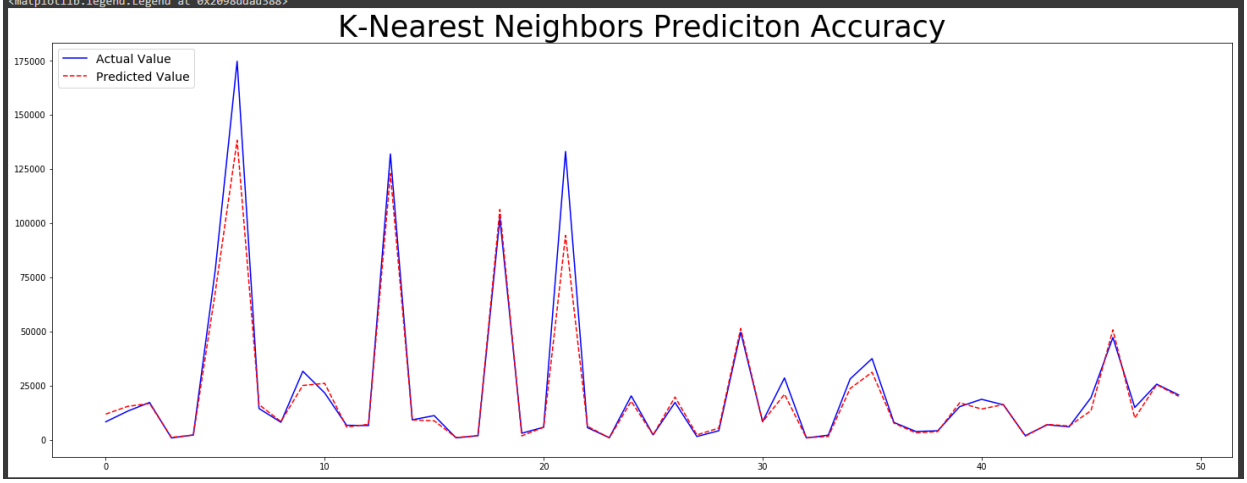
```
print("MAE: ",mae3)
print("Mean squared error: ", mse3)
print("Root Mean squared error: ", rmse3)
print("Root Mean squared Log error: ", rmsle3)
print("R^2 Score:", r2_score3)
print("Adjusted R^2 Score:", adj_r2_score3)
```

```
MAE:  4274.050664830203
Mean squared error:  372687988.71411276
Root Mean squared error:  19305.128559896013
Root Mean squared Log error:  9.868126068085436
R^2 Score: 0.9482451423768304
Adjusted R^2 Score: 0.9475389930971511
```

K-Nearest Neighbors

```
x_axis= list(range(0, 50, 1))
plt.figure(figsize=(25,9))
plt.title("K-Nearest Neighbors Prediciton Accuracy", fontsize=35)
plt.plot(x_axis, y_test[:50], 'b-', label="Actual Value")
plt.plot(x_axis, prediction6[:50], 'r--', label="Predicted Value")
plt.legend(loc="upper left", fontsize=14)
```

```
<matplotlib.legend.Legend at 0x2098ddad388>
```




```
print("MAE: ",mae6)
print("Mean squared error: ", mse6)
print("Root Mean squared error: ", rmse6)
print("Root Mean squared Log error: ", rmsle6)
print("R^2 Score:", r2_score6)
print("Adjusted R^2 Score:", adj_r2_score6)
```

```
MAE:  3354.61182735426
Mean squared error:  307108712.345852
Root Mean squared error:  17524.51746399461
Root Mean squared Log error:  9.7713561773722
R^2 Score: 0.9208067503937163
Adjusted R^2 Score: 0.9197262285116522
```

Among all the Machine Learning Algorithms the performance metrics for Decision Tree show that this algorithm performs better than the other Algorithms. The table below shows the comparison between the Actual Price and Predicted Price and also the Profit which is the scoring for the assessment.

```
Result2 = pd.DataFrame({'Actual Price': y_test, 'Predicted Price by Decision Tree': prediction2})
```

```
Result2[1770:]
```

	Actual Price	Predicted Price by Decision Tree	Retail	Profit
7161	175060	95605.0	204820	109215.0
4351	2475	2100.0	3130	1030.0
7234	10365	10740.0	17540	6800.0
4993	9770	10275.0	14545	4270.0
3816	36670	26095.0	5625	-20470.0
2555	8535	6830.0	8285	1455.0
6333	2450	1625.0	3920	2295.0
6976	11500	10925.0	17690	6765.0
1690	3895	4990.0	6420	1430.0
1328	2355	1960.0	3155	1195.0
3955	57715	38320.0	90530	52210.0
1872	4870	5480.0	7840	2360.0
4397	12675	13300.0	17215	3915.0
689	1110	1110.0	135	-975.0

Hyperparameter Tuning

```
tuning_model.best_params_
```

```
{'max_depth': 7,
 'max_features': 'log2',
 'max_leaf_nodes': None,
 'min_samples_leaf': 3,
 'min_weight_fraction_leaf': 0.1,
 'splitter': 'best'}
```

After choosing the Decision Tree Algorithm for predicting the diamond prices from the offers dataset, hyperparameter tuning was performed and these were the best parameters that were calculated.

These parameters were set while predicting the diamond prices.

- **After predicting the prices for the offers data. I populated the offers.csv file with 893 diamonds where the cost of all the diamonds was 4986956.**
- **The cost of all the diamonds is under the budget.**