

## ▼ Exploratory Data Analysis of Diamond Data Set

Kiranmai Arva, Email: [arvakiranmai@gmail.com](mailto:arvakiranmai@gmail.com), Student Id: 200416487

### ▼ Diamond Data Set

Diamonds are precious and hardest stones among the gemstones. Diamonds are first found in India. Diamonds are beautiful gemstones. They are symbols of love, romance, and commitment, so people use them on their special days like engagement [1]. Diamonds are rare as they are formed under high pressure deep under the earth's crust so diamonds are very costly compared to other gemstones. Even though they are very costly people are very crazy to buy them. In order to know the reason behind why diamonds are costlier and what factors affect the diamond prices led me to choose the Diamond dataset. The diamond data set contains information about more than 50k thousand diamonds. As a beginner in the Data Science field working with such a good number of samples will lead me to come across many challenges and enhance my knowledge.

#### ▼ 1.1) Extract Data Set

- Upload the Data file to goole drive.
- Mount Google Drive at the specified mountain path.
- Change the directory to gdrive.

```
from google.colab import drive
```

```
drive.mount('/content/gdrive')
```

🔗 Go to this URL in a browser: [https://accounts.google.com/o/oauth2/auth?client\\_id=947318989803-6bn6qk8qdgf4n4g3pfee649](https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee649):

Enter your authorization code:

.....

Mounted at /content/gdrive

```
cd gdrive
```

```
↳ /content/gdrive
```

```
cd My\ Drive
```

```
↳ /content/gdrive/My Drive
```

## ▼ 1.2) Importing libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from mpl_toolkits.mplot3d import Axes3D
import sklearn
from sklearn import svm, preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.linear_model import Lasso
%matplotlib inline
```

Reading the CSV file to a Data Frame

```
df=pd.read_csv('diamonds1.csv')
```

```
df.info()
```

```
↳
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53940 entries, 0 to 53939
Data columns (total 11 columns):
Unnamed: 0    53940 non-null int64
carat         53940 non-null float64
cut           53940 non-null object
color         53940 non-null object
clarity       53940 non-null object
depth         53940 non-null object
table         53940 non-null object
```

df.shape

```
(53940, 11)
```

df.dtypes

```
Unnamed: 0    int64
carat         float64
cut           object
color         object
clarity       object
depth         object
table         object
price         object
x             object
y            float64
z            float64
dtype: object
```

df.head(35)

	Unnamed: 0	carat	cut	color	clarity	depth	table	price	x	y	z
0	1	0.23	Ideal	E	SI2	61.5	55	326	3.95	3.98	2.43
1	2	0.21	Premium	E	SI1	59.8	61	326	3.89	3.84	2.31
2	3	0.23	Good	E	VS1	56.9	65	327	4.05	4.07	2.31
3	4	0.29	Premium	I	VS2	62.4	58	334	4.2	4.23	2.63
4	5	0.31	Good	J	SI2	63.3	58	335	4.34	4.35	2.75
5	6	0.24	Very Good	J	VVS2	62.8	57	336	3.94	3.96	2.48
6	7	0.24	Very Good	I	VVS1	62.3	57	336	3.95	3.98	2.47
7	8	0.26	Very Good	H	SI1	61.9	55	337	4.07	4.11	2.53
8	9	0.22	Fair	E	VS2	65.1	61	337	3.87	3.78	2.49
9	10	0.23	Very Good	H	VS1	59.4	61	338	4	4.05	2.39
10	11	0.30	Good	J	SI1	64	55	339	4.25	4.28	2.73
11	12	0.23	Ideal	J	VS1	62.8	56	340	3.93	3.90	2.46
12	13	0.22	Premium	F	SI1	60.4	61	342	3.88	3.84	2.33
13	14	0.31	Ideal	J	SI2	62.2	54	344	4.35	4.37	2.71
14	15	0.20	Premium	E	SI2	60.2	62	345	3.79	3.75	2.27
15	16	0.32	Premium	E	I1	60.9	58	345	4.38	4.42	2.68
16	17	0.30	Ideal	I	SI2	62	54	348	4.31	4.34	2.68
17	18	0.30	Good	J	SI1	63.4	54	351	4.23	4.29	2.70
18	19	0.30	Good	J	SI1	63.8	56	351	4.23	4.26	2.71
19	20	0.30	Very Good	J	SI1	62.7	59	351	4.21	4.27	2.66
20	21	0.30	Good	I	SI2	63.3	56	351	4.26	4.30	2.71
21	22	0.23	Very Good	E	VS2	63.8	55	352	3.85	3.92	2.48
22	23	0.23	Very Good	H	VS1	61	57	353	3.94	3.96	2.41

23	24	0.31	Very Good	J	SI1	59.4	62	353	4.39	4.43	2.62
24	25	0.31	Very Good	J	SI1	58.1	62	353	4.44	4.47	2.59
25	26	0.23	Very Good	G	VVS2	60.4	58	354	3.97	4.01	2.41
26	27	0.24	Premium	I	VS1	62.5	57	355	3.97	3.94	2.47
27	28	0.30	Very Good	J	VS2	62.2	57	357	4.28	4.30	2.67
28	29	0.23	Very Good	D	VS2	60.5	61	357	3.96	3.97	2.40
29	30	0.23	Very Good	F	VS1	60.9	57	357	3.96	3.99	2.42
30	31	0.23	Very Good	F	VS1	Null	57	402	4	4.03	2.41

When executed the above statement there are few Nulls encountered in the data set.

```
29      30      0.23  Very Good      F      VS1      60.9      57      402      3.96      4.01      2.42
```

### ▼ 1.3) Scrubbing and Formatting Numerical and Categorical Data

Data Set contains Null string numpy can not read the Null strings. So replace the Null values in the dataframe with numpy NAN values.

```
df.replace('Null',np.NaN, inplace = True)
```

```
df.isnull().any()
```

```

↳ Unnamed: 0    False
   carat        False
   cut          False
   color        False
   clarity       False
   depth         True
   table         True
   price         True
   x             True
   y            False
   z            False
dtype: bool

```

## Removing additional index

```
df.head()
df = df.drop('Unnamed: 0', axis = 1)
df.head()
```

```
↗
```

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58	334	4.2	4.23	2.63
4	0.31	Good	J	SI2	63.3	58	335	4.34	4.35	2.75

## ▼ Exploring Numerical Columns in the Data Set

### Replacing the missing values in Depth column

```
sum(pd.isnull(df['depth']))
```

```
↗ 4
```

Depth value is NaN for the following Locations

```
a=df[df['depth'].isnull()]
a
```

↗

	carat	cut	color	clarity	depth	table	price	x	y	z
<b>30</b>	0.23	Very Good	F	VS1	NaN	57	402	4	4.03	2.41
<b>50</b>	0.24	Very Good	F	SI1	NaN	61	404	4.02	4.03	2.45
<b>64</b>	0.42	Premium	I	SI2	NaN	59	552	4.78	4.84	2.96
<b>120</b>	0.71	Ideal	D	SI2	NaN	56	2762	5.73	5.69	3.56

Depth percentage is calculated using the formula  $D = 2 * z / (x + y)$  [2].

- To Calculate depth X,Y,Z values should not be NaN
- In this data set X column contains NaN values.
- Total 3 columns of X contains NaN, So dropping these three columns out of 50k records will not affect our final results.

```
a=df[df['x'].isnull()]
a
```

↗

	carat	cut	color	clarity	depth	table	price	x	y	z
<b>1203</b>	0.74	Ideal	F	VS2	60.5	59	2936	NaN	5.86	3.53
<b>1223</b>	0.70	Good	F	VVS2	63.1	57	2940	NaN	5.66	3.55
<b>1486</b>	0.60	Very Good	F	IF	60.1	54	2988	NaN	5.58	3.33

```
df.drop([df.index[1203] , df.index[1223],df.index[1486]],inplace=True)
```

```
df['depth']=pd.to_numeric(df['depth'])
```

```
df['depth'] = df.apply(
```

```
lambda row: float(((row['z']*2)/float(row['x'])+row['y'])*10) if pd.isnull(row['depth']) else row['depth'],
axis=1
)
```

```
a=df[df['depth'].isnull()]
a
```

```
↳ carat cut color clarity depth table price x y z
```

---

### Replacing the missing values in the Table column

```
sum(pd.isnull(df['table']))
```

```
↳ 4
```

```
df["table"].describe()
```

```
↳ count      53933
   unique       127
   top           56
   freq       9881
   Name: table, dtype: object
```

```
df["table"]= pd.to_numeric(df["table"])
```

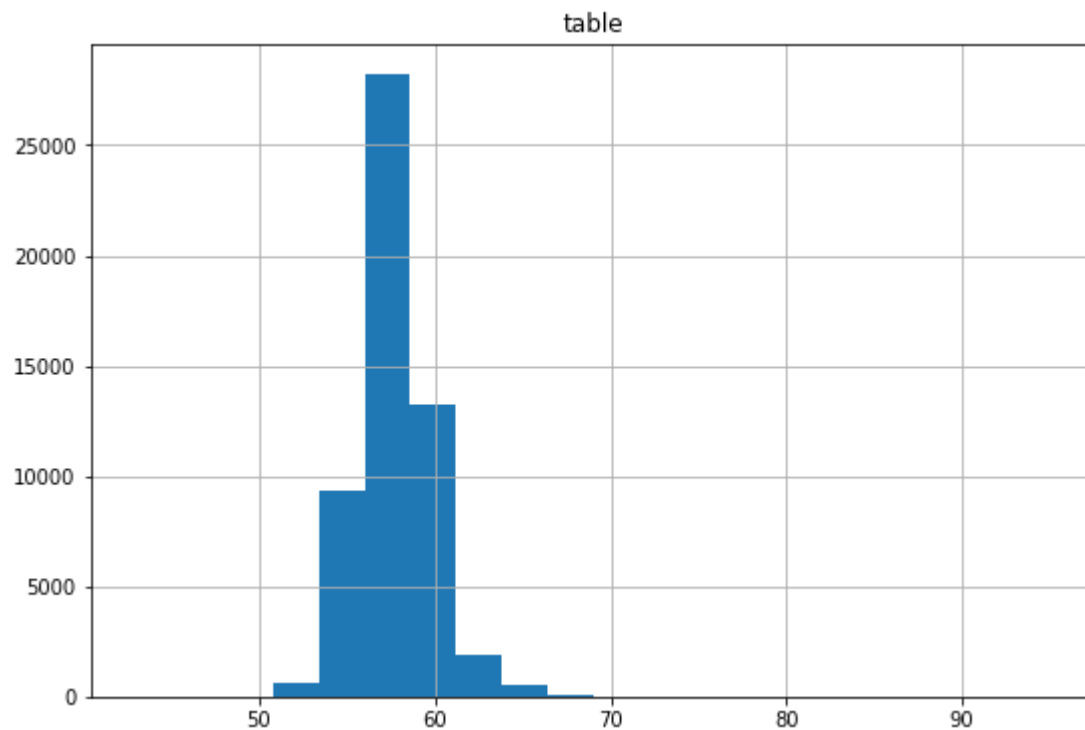
```
df["table"].describe()
```

```
↳
```



```
count    53933.000000
mean      57.457317
std       2.234490
df.hist(column = 'table',
        figsize=(9,6),
        bins = 20)
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7fb49c4e0780>]],
      dtype=object)
```



From the above figure the mean value is 57 so replacing the NaN values with 57 as that makes sense.

```
df['table'] = df['table'].fillna(57)
```

### Replacing the missing values in the Price column

```
sum(pd.isnull(df['price']))
```

```
↳ 6
```

```
df["price"] = pd.to_numeric(df["price"], errors="coerce")
```

```
df["price"].describe()
```

```
↳ count    53931.000000  
   mean      3932.979529  
   std       3989.748194  
   min       326.000000  
   25%       949.500000  
   50%      2401.000000  
   75%      5325.500000  
   max      18823.000000  
   Name: price, dtype: float64
```

In total there are six Nan values in the price column. Here instead of replacing the Nan values with mean, values are replaced according to the standard deviation.

The values are replaced according to the distribution of Standard deviation i.e out of 6 Nan column values 2 are replaced with 25%std, two with 50%std, the rest with 75% std.

```
df[df['price'].isnull()]
```

```
↳
```

	carat	cut	color	clarity	depth	table	price	x	y	z
<b>125</b>	0.70	Premium	F	VS2	61.6	60.0	NaN	5.65	5.59	3.46
<b>144</b>	0.71	Ideal	D	SI2	61.6	55.0	NaN	5.74	5.76	3.54

```
df.at[125,'price']=949
df.at[144,'price']=949
df.at[165,'price']=2401
df.at[449,'price']=2401
df.at[653,'price']=5325
df.at[881,'price']=5325
```

```
df.isnull().any()
```

```
↳ carat      False
   cut        False
   color      False
   clarity    False
   depth      False
   table      False
   price      False
   x          False
   y          False
   z          False
dtype: bool
```

From the above statement there are no Nan values in the data set as all the values are replaced logically.

```
df.info()
```

```
↳
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 53937 entries, 0 to 53939
Data columns (total 10 columns):
carat      53937 non-null float64
cut        53937 non-null object
color      53937 non-null object
clarity     53937 non-null object
depth      53937 non-null float64
table      53937 non-null float64

v          53937 non-null float64

memory usage: 7.0+ MB
```

## ▼ Exploring Categorical values in the Data Set

```
df['cut'].describe()
```

```
count      53937
unique         5
top        Ideal
freq       21550
Name: cut, dtype: object
```

From the above statement "Cut" column contains Five Unique values. 'Ideal' is the top most value with the frequency 21550.

```
df['color'].describe()
```

```
count      53937
unique         7
top         G
freq       11292
Name: color, dtype: object
```

The "Color" column contains seven different values starting from J (worst) to D (best). 'G' has the highest frequency.

```
df['clarity'].describe()
```

```
count    53937
unique         8
top      SI1
freq     13065
Name: clarity, dtype: object
```

The "Clarity" column contains eight unique columns. The eight unique values are (I1 (worst), SI2, SI1, VS2, VS1, VVS2, VVS1, IF (best)).

```
df.head(5)
```

```
↳
```

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326.0	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326.0	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327.0	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334.0	4.2	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335.0	4.34	4.35	2.75

```
df.tail(5)
```

```
↳
```

	carat	cut	color	clarity	depth	table	price	x	y	z
<b>53935</b>	0.72	Ideal	D	SI1	60.8	57.0	2757.0	5.75	5.76	3.50
<b>53936</b>	0.72	Good	D	SI1	63.1	55.0	2757.0	5.69	5.75	3.61
<b>53937</b>	0.70	Very Good	D	SI1	62.8	60.0	2757.0	5.66	5.68	3.56
<b>53938</b>	0.86	Premium	H	SI2	61.0	58.0	2757.0	6.15	6.12	3.74
<b>53939</b>	0.75	Ideal	D	SI2	62.2	55.0	2757.0	5.83	5.87	3.64

```
df.dtypes
```

```
↳ carat    float64
   cut      object
   color    object
   clarity  object
   depth    float64
   table    float64
   price    float64
   x        object
   y        float64
   z        float64
   dtype: object
```


```
categorical = df.dtypes[df.dtypes == "object"].index
df[categorical].describe()
```

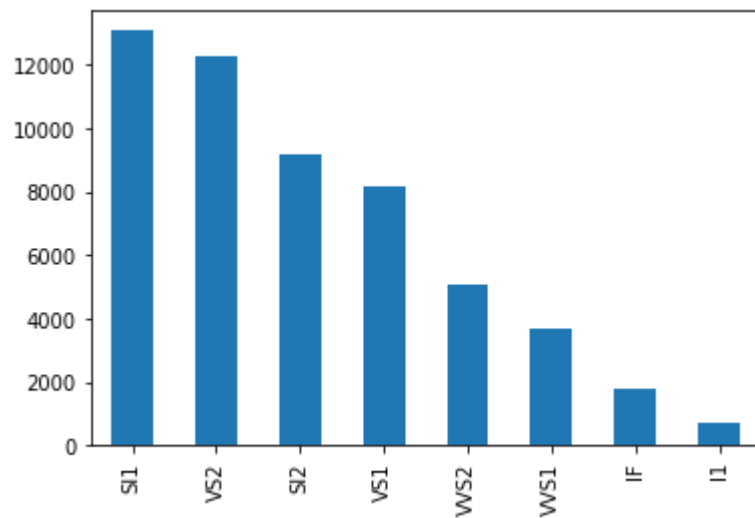
```
↳
```

	cut	color	clarity	x
count	53937	53937	53937	53937
unique	5	7	8	554

Distribution of Categorical Data

```
df['clarity'].value_counts().plot(kind='bar')
```

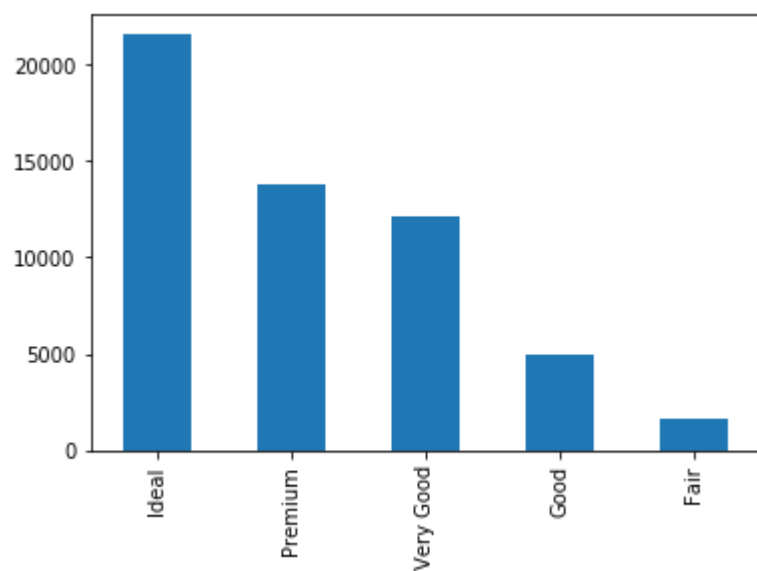
 <matplotlib.axes.\_subplots.AxesSubplot at 0x7fb49c4c1080>



**Observation:** The eight unique values are (I1 (worst), SI2, SI1, VS2, VS1, VVS2, VVS1, IF (best)). From the above observation there are only very few diamonds with Ideal clarity. Most of the diamonds falls under the category SI1 and SI2 which are below average.

```
df['cut'].value_counts().plot(kind='bar')
```

↗ <matplotlib.axes.\_subplots.AxesSubplot at 0x7fb49c447898>

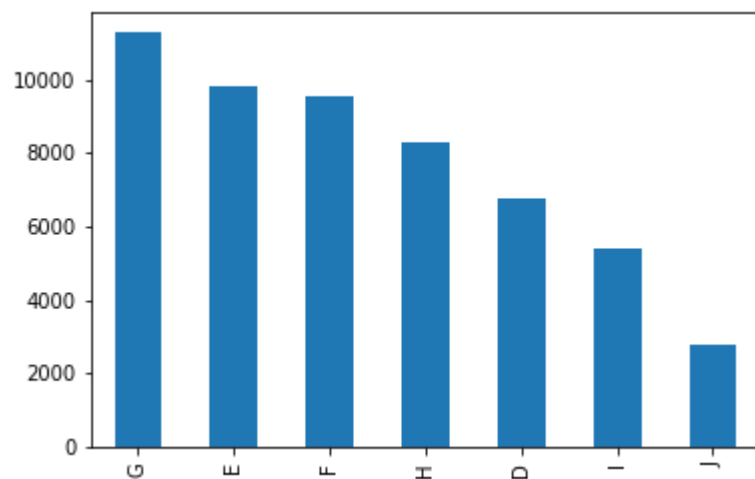


**Observation:** Ideal cut category is the best cut among diamonds. Most of the diamonds fall under this category.



```
df['color'].value_counts().plot(kind='bar')
```

↳ <matplotlib.axes.\_subplots.AxesSubplot at 0x7fb49c450ba8>



**Observation:** J (worst) to D (best). very few diamonds have the worst color that is of 'J' category. Out of seven categories in color the highest falls under the G category which is in between worst and best.

For Training Models the best approach is converting the categorical values to numeric.i.e assigning each attribute of categorical value with a numeric value.

```
d_cut = {'Fair' : 5, 'Good' : 4, 'Very Good' : 3, 'Premium' : 2, 'Ideal' : 1}
d_clarity ={'I1' : 8, 'SI2' : 7, 'SI1' : 6, 'VS2' : 5, 'VS1' : 4, 'VVS2' : 3, 'VVS1' : 2 , 'IF' : 1}
d_color = {'D':1, 'E':2, 'F':3, 'G':4, 'H':5, 'I':6, 'J':7}
```

```
df['cut'] = df['cut'].map(d_cut)
df['clarity'] = df['clarity'].map(d_clarity)
df['color'] = df['color'].map(d_color)
```

```
df.head()
```

```
↗
```

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	1	2	7	61.5	55.0	326.0	3.95	3.98	2.43
1	0.21	2	2	6	59.8	61.0	326.0	3.89	3.84	2.31
2	0.23	4	2	4	56.9	65.0	327.0	4.05	4.07	2.31
3	0.29	2	6	5	62.4	58.0	334.0	4.2	4.23	2.63
4	0.31	4	7	7	63.3	58.0	335.0	4.34	4.35	2.75

#### ▼ 1.4) Analysis:

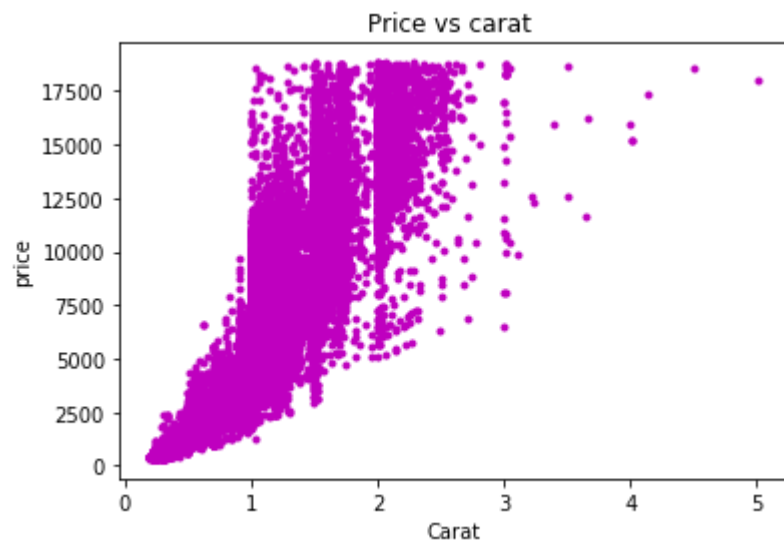
Predicting the Diamond Prices based on the available features. Below are the Features of Diamonds [2].

1. carat: weight of the diamond
2. price: price in US dollars

3. cut quality of the cut
4. color diamond colour
5. clarity a measurement of how clear the diamond
6. x length in mm
7. y width in mm
8. z depth in mm
9. table width of top of diamond
10. depth total depth percentage

```
plt.scatter(df["carat"], df["price"], marker='.', c='m')  
plt.title("Price vs carat ")  
plt.ylabel('price')  
plt.xlabel('Carat')
```

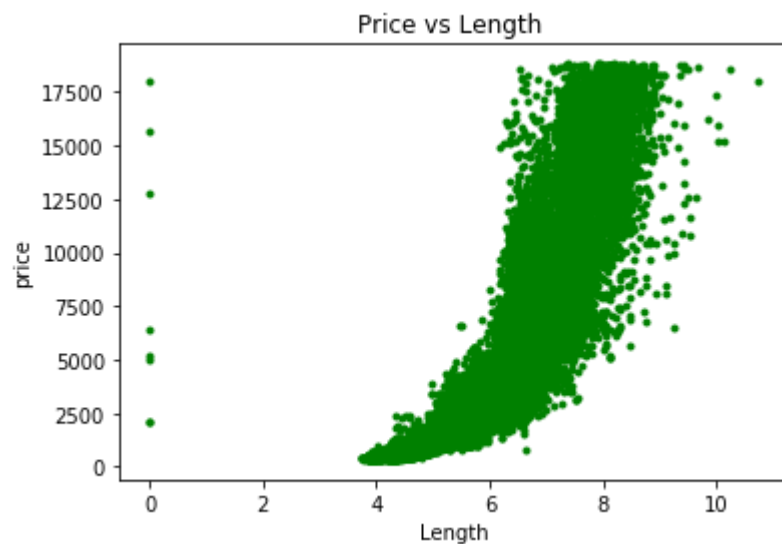
☞ `Text(0.5, 0, 'Carat')`



**Observation:** From the above scatter graph as the carat increases the price of the diamond also increases in most of the cases.

```
df["x"] = pd.to_numeric(df["x"])
plt.scatter(df["x"], df["price"], marker='.', c='g')
plt.title("Price vs Length ")
plt.ylabel('price')
plt.xlabel('Length')
```

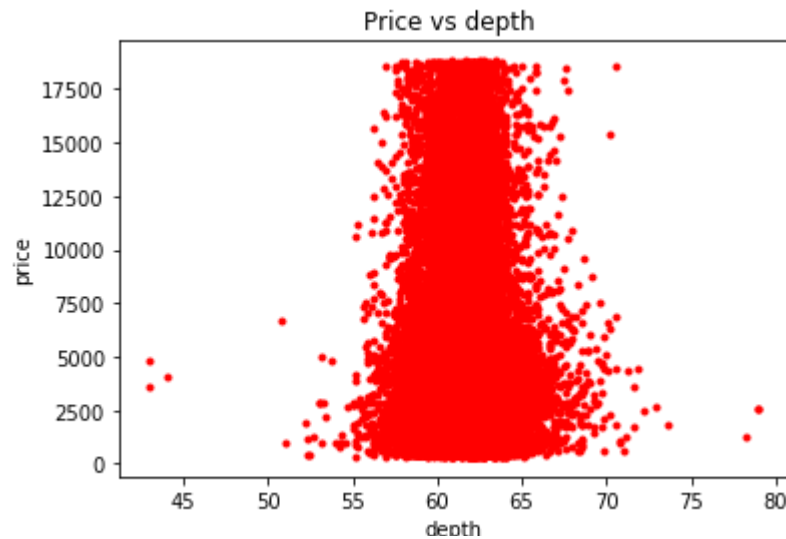
☞ Text(0.5, 0, 'Length')



**Observation:** As the length of the diamond incerases the price also increases so both are directly proportional.

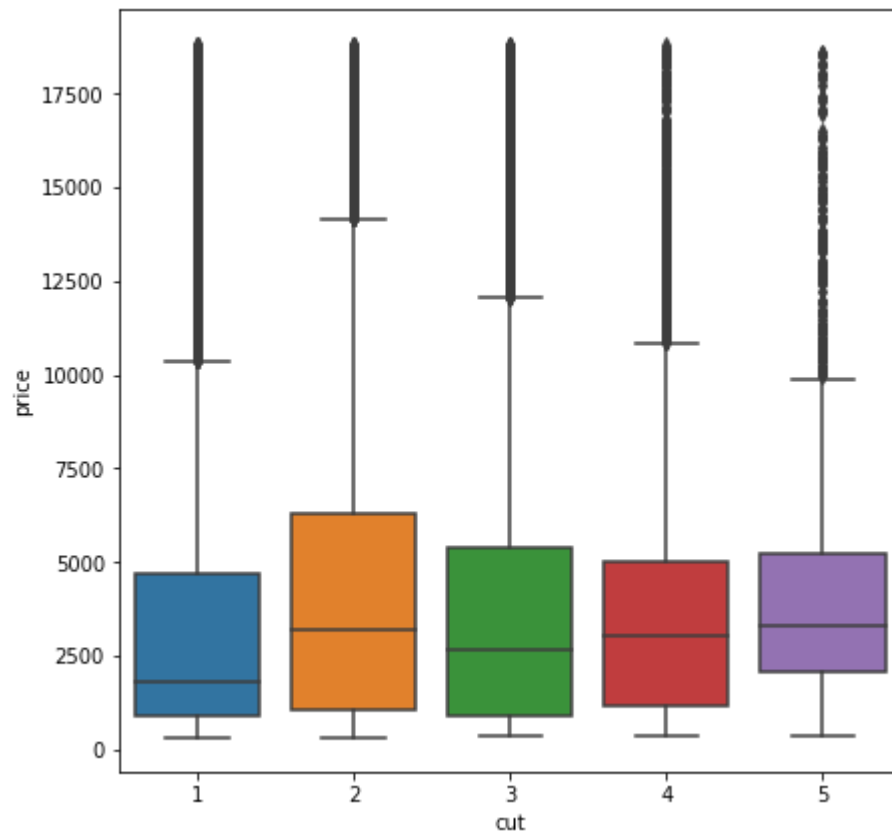
```
plt.scatter(df["depth"], df["price"], marker='.', c='R')
plt.title("Price vs depth ")
plt.ylabel('price')
plt.xlabel('depth')
```

```
↳ Text(0.5, 0, 'depth')
```



**Observation:** All most all the diamonds depth fall under the range 55 to 70. So the clarity of the diamond depend on the depth as the depth increases the reflection and refraction of the light decreases due to this factor the Clarity of diamond may decrease. So only on the basis of the Depth the Price can not be calculated. But from the graph very few scatter points are out of the range in that case the price is very low.

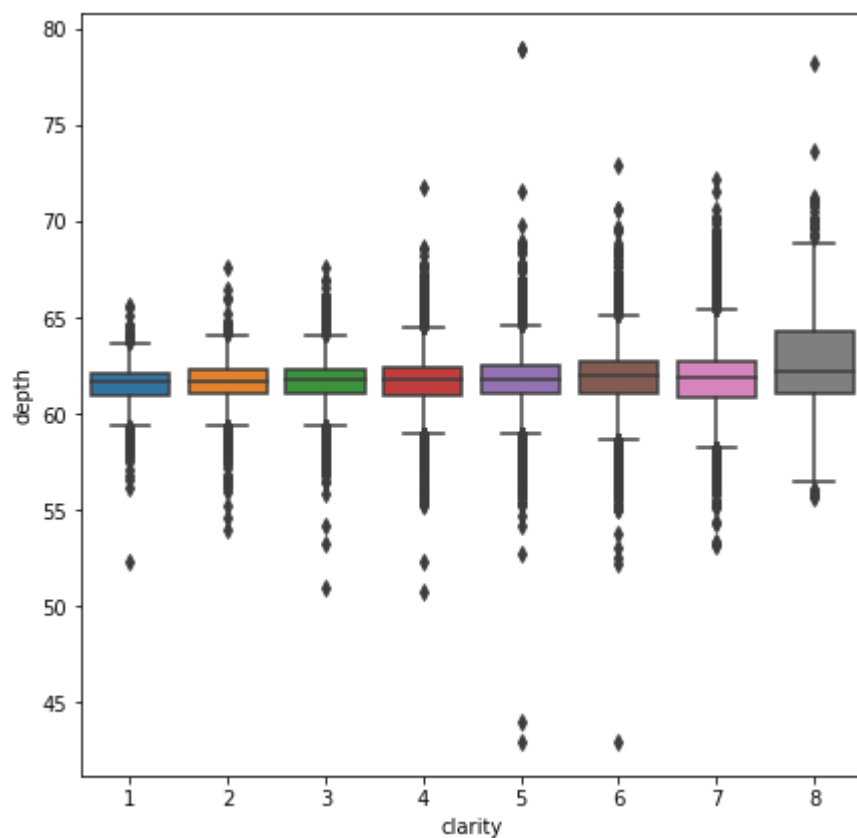
```
x='cut'
y='price'
plt.figure(figsize=(7,7))
ax=sns.boxplot(x=x,y=y,data=df)
plt.show()
```



**Observation:** According to the data set Ideal is the top cut but the premium cut has the highest price compared to ideal cut. Most of the diamonds fall under the premium category followed by very good cut that is 3.

```
x='clarity'
y='depth'
plt.figure(figsize=(7,7))
ax=sns.boxplot(x=x,y=y,data=df)
```

```
plt.show()
```

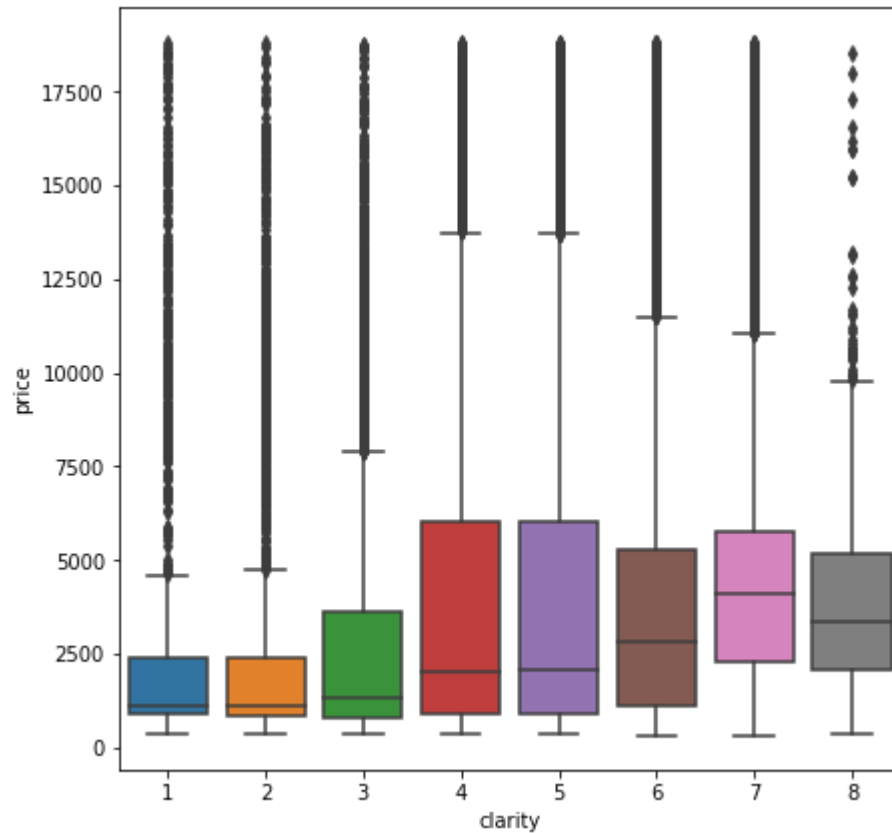


**Observation:** 'I1' : 8, 'SI2' : 7, 'SI1' : 6, 'VS2' : 5, 'VS1' : 4, 'VVS2' : 3, 'VVS1' : 2, 'IF' : 1. As the depth increases the clarity decreases. It is clearly shown from the figure. I1 stands for worst clarity which is 8 in the plot has the highest depth value. IF which is the best clarity, number 1 in the plot has the lowest depth.

```
x='clarity'
y='price'
plt.figure(figsize=(7,7))
```



```
ax=sns.boxplot(x=x,y=y,data=df)  
plt.show()
```



**Observation:** The above graph leads to an interesting observation even though the level 1,2,3 are the best at clarity, their prices are very low. The prices are high for the level 4,5 this is due to the diamonds in this range are eye clear and maybe preferable by people and so the prices are very high.

### Finding the highest diamond price in the data set:

```
df[(df.carat <= 1) & (df.price > 12000)]
```

	carat	cut	color	clarity	depth	table	price	x	y	z
<b>25625</b>	1.0	1	1	2	60.7	56.0	14498.0	6.47	6.54	3.95
<b>26407</b>	1.0	4	1	1	63.3	59.0	15928.0	6.33	6.37	4.02
<b>26483</b>	1.0	3	1	1	63.3	59.0	16073.0	6.37	6.33	4.02
<b>26660</b>	1.0	1	1	1	60.7	57.0	16469.0	6.44	6.48	3.92

From the above the diamonds with less than or equal to one carat and price less than 12 k, four rows are retrieved of which 3 diamonds have highest prices because they have high grades of color and clarity.

## 1.5) Report Initial Findings

After Exploring the data set the problem statement which I came up with is

**Problem Statement** Based on the available features in the data set what factors play an important role to predict the price of diamond

After the initial level of exploring the data few observations which are to be noticed are described below.

All the features are tested against the price, below are the few observations.

1. Carat Weight: A positive correlation is observed, As the Carat value increases the price also increases.
2. Length(X): This feature also has a positive correlation with the price.
3. Depth: At first it is assumed that it has a positive relationship with price but after plotting the graph all most all diamonds fall under the range 57 to 69. so depth doesn't have a positive relationship with price.
4. Table: Similar to depth, table also has range from 43-95. One can observe this range from the description of diamond data set so no graph is plotted against this feature.
5. Cut, Color and Clarity: For the diamonds with 'IF' clarity grade, which is highest grade have the lowest prices and the diamonds with clarity grade 'VS2' and 'VS1' have the highest prices. Ideal cut is the highest grade but they have low prices compared to the cut grades 'Premium' and 'Very

Good'. All seems to have negative corelationship with the mean price from the above box plots.

**Modeling Techniques:** According to the Scikitlearn algorithm cheat sheet [3]. Following are observed from the sheet while choosing an appropriate algorithm.

1. Diamond data set contains more than 50 samples.
2. Predicting a quantity which is 'Price'.
3. Samples are less than 100k.
4. Few features are important. Considering above senarios, I suspect "Lasso Regression","Linear Regression" and "Elastic Net" are the best Modeling techiques for the Diamond Data set beacuse predicting a quantity is the target. All the algorithms which i mentioned above are supervised learning algorithms techniques.

## References

1. Diamond, R. (2019). Why Diamonds Are Important & Significant for Love & Commitment. [online] Real Is A Diamond. Available at: <https://realisadiamond.com/diamonds-symbols-commitment/>
2. Kaggle.com. (2019). Diamonds. [online] Available at: <https://www.kaggle.com/shivam2503/diamonds>
3. Scikit-learn.org. (2019). Choosing the right estimator — scikit-learn 0.21.3 documentation. [online] Available at: [https://scikit-learn.org/stable/tutorial/machine\\_learning\\_map/index.html](https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html) .

