Diamond Dataset Analysis

This project attempts to understand various trends present in the `Diamonds Dataset` and also tries to make an attempt to make a Linear Regression Model which tries to predict the price of any diamond based on its attributes like `carat` , `cut` , `color` , `clarity` , etc.

For this project, we shall be using the pre-existing dataset already present in the `Seaborn` library which provides us with a datset for prices and all other attributes of **53,840 diamonds** in total. Some sampling may also be performed in order to derive insights from the entire dataset and to understand various trends exhibited by the data. This is because performing operations on 53,840 tuples of data may increase the noise and the duration of execution of the programs as well.

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [3]:
```
diamonds = sns.load_dataset("diamonds")
```

In [3]:
```
diamonds
```

Out[3]:

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75
...	...	...	...	...	...	...	...	...	...	...
53935	0.72	Ideal	D	SI1	60.8	57.0	2757	5.75	5.76	3.50
53936	0.72	Good	D	SI1	63.1	55.0	2757	5.69	5.75	3.61
53937	0.70	Very Good	D	SI1	62.8	60.0	2757	5.66	5.68	3.56
53938	0.86	Premium	H	SI2	61.0	58.0	2757	6.15	6.12	3.74
53939	0.75	Ideal	D	SI2	62.2	55.0	2757	5.83	5.87	3.64

53940 rows × 10 columns

In [4]:
```
1 diamonds.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53940 entries, 0 to 53939
Data columns (total 10 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   carat    53940 non-null  float64
 1   cut      53940 non-null  category
 2   color    53940 non-null  category
 3   clarity  53940 non-null  category
 4   depth    53940 non-null  float64
 5   table    53940 non-null  float64
 6   price    53940 non-null  int64
 7   x        53940 non-null  float64
 8   y        53940 non-null  float64
 9   z        53940 non-null  float64
dtypes: category(3), float64(6), int64(1)
memory usage: 3.0 MB
```

In [5]:
```
1 diamonds[diamonds.isnull().any(axis=1)]
```

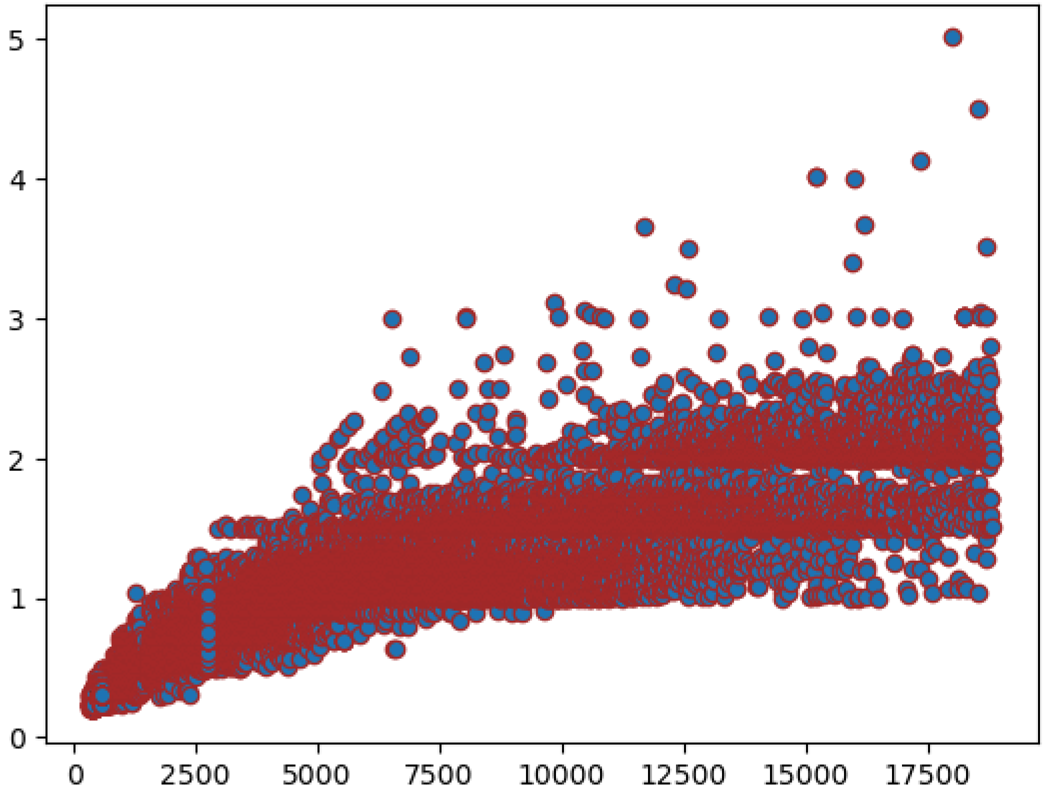
Out[5]:

carat	cut	color	clarity	depth	table	price	x	y	z

diamonds.describe()

In [52]:
```
1 plt.scatter(diamonds['price'], diamonds['carat'], edgecolor='brown')
```

Out[52]: <matplotlib.collections.PathCollection at 0x228ea6ad3d0>



Trying to take samples from the dataset:

In [53]:

```
help(diamonds.sample)
```

```
spider    8
falcon    2
Name: num_legs, dtype: int64

A random 50% sample of the ``DataFrame`` with replacement:

>>> df.sample(frac=0.5, replace=True, random_state=1)
      num_legs  num_wings  num_specimen_seen
dog          4          0                  2
fish         0          0                  8

An upsample sample of the ``DataFrame`` with replacement:
Note that `replace` parameter has to be `True` for `frac` parameter
> 1.

>>> df.sample(frac=2, replace=True, random_state=1)
        num_legs  num_wings  num_specimen_seen
dog            4          0                  2
fish           0          0                  8
```

In [4]:

```
new_diamonds = diamonds.sample(frac=0.25)
new_diamonds
```

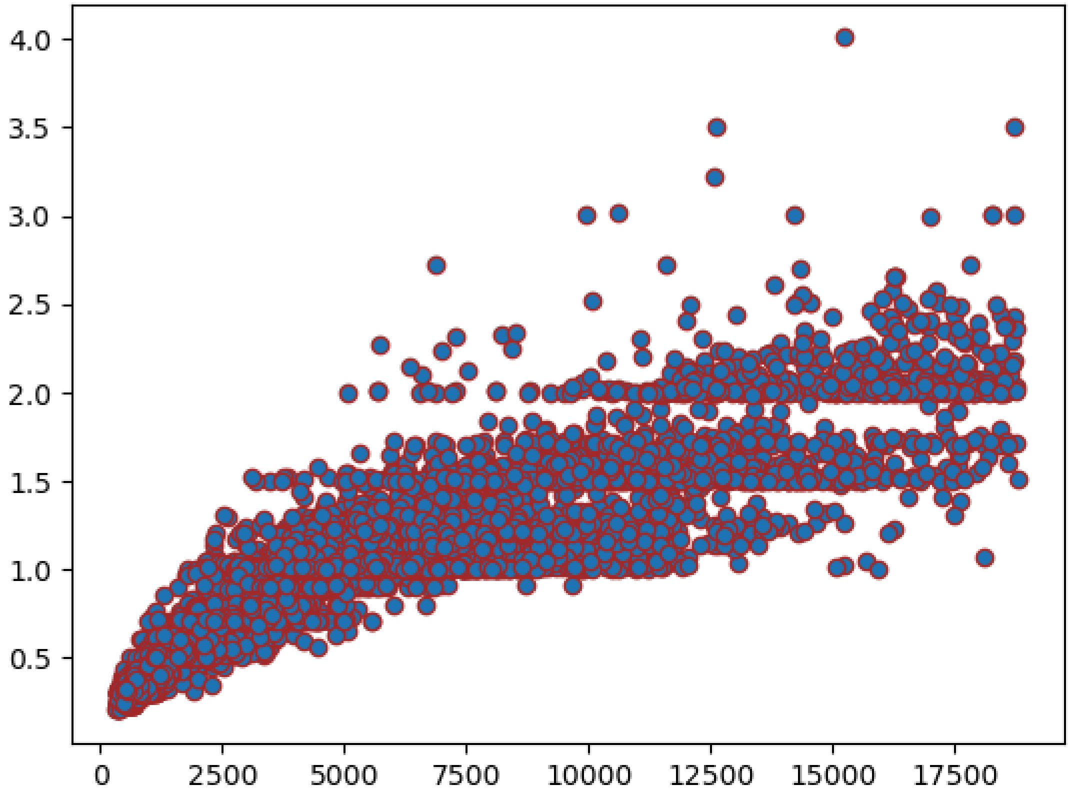
Out[4]:

	carat	cut	color	clarity	depth	table	price	x	y	z
15164	1.31	Premium	H	VS2	59.6	58.0	6095	7.14	7.08	4.24
45273	0.54	Very Good	F	VS2	59.4	57.0	1662	5.29	5.35	3.16
28326	0.33	Premium	G	VS1	61.9	58.0	666	4.43	4.46	2.75
16964	1.10	Very Good	E	VS2	61.9	55.0	6776	6.61	6.67	4.11
34871	0.30	Very Good	G	VVS2	63.2	57.0	878	4.28	4.23	2.69
...	...	...	...	...	...	...	...	...	...	...
4412	1.01	Very Good	I	SI1	58.8	58.0	3610	6.52	6.57	3.85
8318	1.00	Very Good	E	SI2	63.3	55.0	4390	6.29	6.25	3.97
26588	2.40	Premium	J	SI1	59.7	58.0	16304	8.75	8.71	5.21
18211	1.00	Premium	E	VS1	58.7	62.0	7392	6.60	6.55	3.86
7146	0.93	Ideal	E	SI1	62.0	56.0	4179	6.25	6.29	3.89

13485 rows × 10 columns

In [8]:
```
1 plt.scatter(new_diamonds['price'], new_diamonds['carat'], edgecolor='br
```

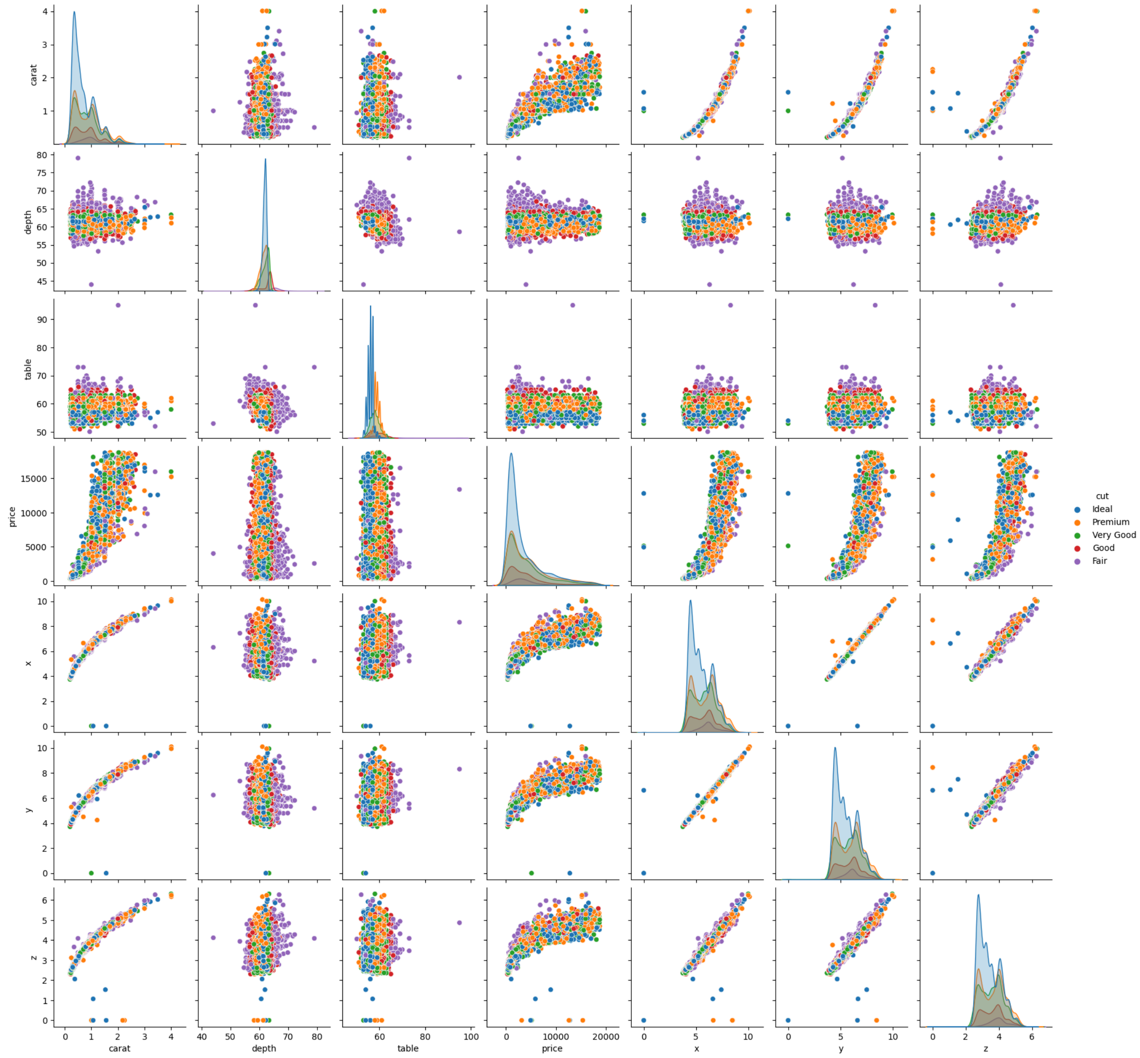
Out[8]: <matplotlib.collections.PathCollection at 0x278999156d0>



In [56]:
```
1 sns.pairplot(new_diamonds, hue = 'cut')
```

```
c:\Users\Admin\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```

Out[56]:
```
<seaborn.axisgrid.PairGrid at 0x228ec0fd290>
```



In this given dataset, if we consider 'price' to be the dependent variable, and other values as the independent variable, then we do have a linear graph with respect to all the other attributes available to us.

Here, from the given dataset:

```
Categorical Variables : {'cut','clarity', 'color'}
Numerical Variavles : {'carat', 'depth', 'table', 'price', 'x', 'y', 'z'}
```

In [6]:
```
1 list1 = list(new_diamonds.columns)
2 list1
```

Out[6]:
```
['carat', 'cut', 'color', 'clarity', 'depth', 'table', 'price', 'x', 'y', 'z']
```

In [7]:
```
categorical = ['cut', 'clarity', 'color']
numerical = list(set(list1)-set(categorical))
numerical
```

Out[7]:
```
['x', 'z', 'depth', 'y', 'carat', 'table', 'price']
```

In [8]:
```
new_diamonds.groupby('carat')[['carat']].count()
```

Out[8]:

	carat
carat	
0.20	4
0.23	80
0.24	62
0.25	47
0.26	62
...	...
2.80	1
3.00	1
3.01	6
3.05	1
3.40	1

238 rows × 1 columns

In [9]:

```
x = pd.DataFrame(new_diamonds.groupby('carat')[['price']].mean())
print(x)
print('Mean price = ',x[['price']].mean())
print('Median price = ',x[['price']].median())
print('Max price = ',x[['price']].max())
print('Min price = ',x[['price']].min())
print("Total Types of Carats of Diamonds available : ", x['price'].cour
```

```
              price
carat
0.20     367.000000
0.23     478.100000
0.24     505.612903
0.25     539.936170
0.26     545.258065
...             ...
2.80   15030.000000
3.00   16970.000000
3.01   17933.000000
3.05   10453.000000
3.40   15964.000000

[238 rows x 1 columns]
Mean price =  price    8706.209329
dtype: float64
Median price =  price    8238.8
dtype: float64
Max price =  price    18756.0
dtype: float64
Min price =  price    367.0
dtype: float64
Total Types of Carats of Diamonds available :  238
```

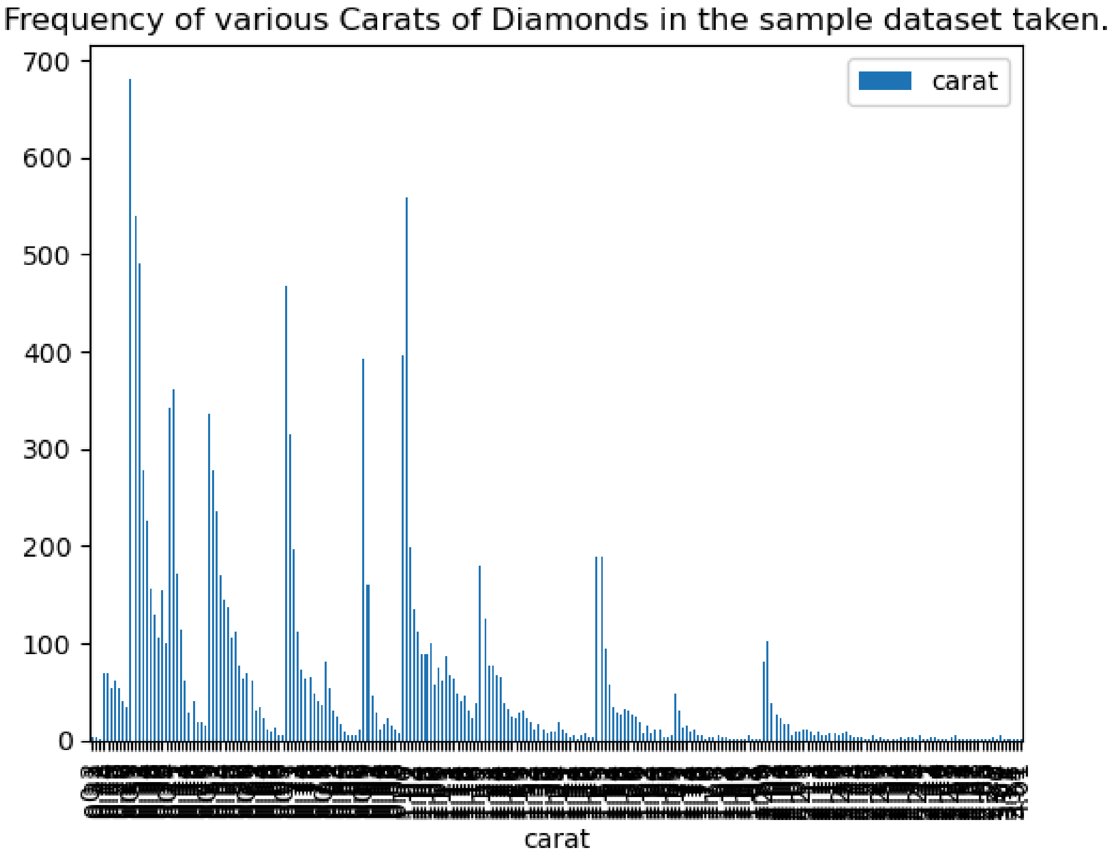
In [10]:

```
unique_carat = list(x.index)
print(unique_carat)
```

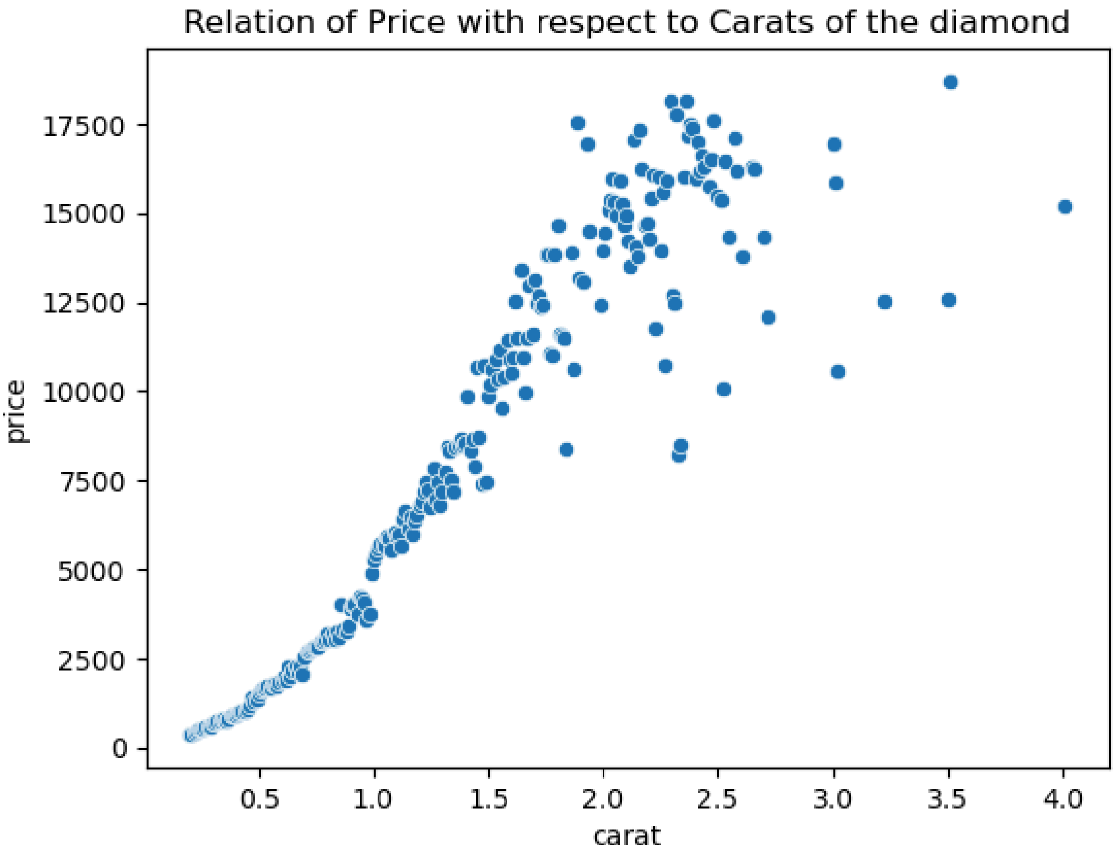
```
[0.2, 0.23, 0.24, 0.25, 0.26, 0.27, 0.28, 0.29, 0.3, 0.31, 0.32, 0.33, 0.3
4, 0.35, 0.36, 0.37, 0.38, 0.39, 0.4, 0.41, 0.42, 0.43, 0.44, 0.45, 0.46,
0.47, 0.48, 0.49, 0.5, 0.51, 0.52, 0.53, 0.54, 0.55, 0.56, 0.57, 0.58, 0.5
9, 0.6, 0.61, 0.62, 0.63, 0.64, 0.65, 0.66, 0.67, 0.68, 0.69, 0.7, 0.71,
0.72, 0.73, 0.74, 0.75, 0.76, 0.77, 0.78, 0.79, 0.8, 0.81, 0.82, 0.83, 0.8
4, 0.85, 0.86, 0.87, 0.88, 0.89, 0.9, 0.91, 0.92, 0.93, 0.94, 0.95, 0.96,
0.97, 0.98, 0.99, 1.0, 1.01, 1.02, 1.03, 1.04, 1.05, 1.06, 1.07, 1.08, 1.0
9, 1.1, 1.11, 1.12, 1.13, 1.14, 1.15, 1.16, 1.17, 1.18, 1.19, 1.2, 1.21,
1.22, 1.23, 1.24, 1.25, 1.26, 1.27, 1.28, 1.29, 1.3, 1.31, 1.32, 1.33, 1.3
4, 1.35, 1.36, 1.37, 1.38, 1.39, 1.4, 1.41, 1.42, 1.43, 1.44, 1.45, 1.46,
1.47, 1.48, 1.5, 1.51, 1.52, 1.53, 1.54, 1.55, 1.56, 1.57, 1.58, 1.59, 1.
6, 1.61, 1.62, 1.63, 1.64, 1.65, 1.66, 1.67, 1.68, 1.69, 1.7, 1.71, 1.72,
1.73, 1.74, 1.75, 1.76, 1.77, 1.78, 1.79, 1.8, 1.81, 1.82, 1.83, 1.85, 1.8
6, 1.87, 1.89, 1.9, 1.91, 1.92, 1.93, 1.95, 1.96, 1.97, 1.99, 2.0, 2.01,
2.02, 2.03, 2.04, 2.05, 2.06, 2.07, 2.08, 2.09, 2.1, 2.11, 2.12, 2.13, 2.1
4, 2.15, 2.16, 2.17, 2.18, 2.19, 2.2, 2.21, 2.22, 2.23, 2.24, 2.25, 2.26,
2.27, 2.28, 2.29, 2.3, 2.31, 2.32, 2.33, 2.34, 2.35, 2.36, 2.37, 2.38, 2.
4, 2.41, 2.42, 2.43, 2.45, 2.47, 2.48, 2.49, 2.5, 2.51, 2.52, 2.53, 2.54,
2.57, 2.58, 2.6, 2.61, 2.66, 2.67, 2.74, 2.75, 2.8, 3.0, 3.01, 3.05, 3.4]
```

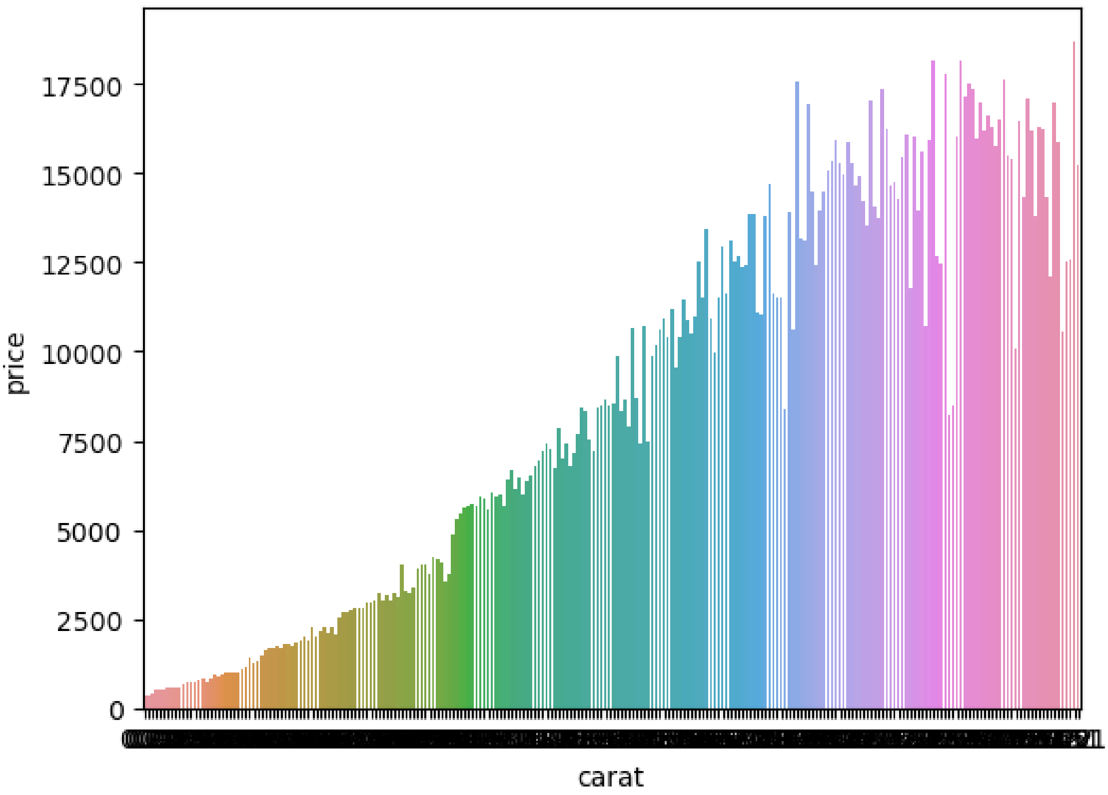
In [14]:

```
new_diamonds.groupby('carat')[['carat']].count().plot(kind='bar')
plt.title("Frequency of various Carats of Diamonds in the sample datase
plt.show()
```



In [15]:
```
sns.scatterplot(data=x, x=x.index, y=x['price'])
plt.title("Relation of Price with respect to Carats of the diamond")
plt.show()
sns.barplot(data=x, x=x.index, y=x['price'])
plt.show()
```





To find trends with respect to each type of `Cut` existing in the given dataset:

In [16]:

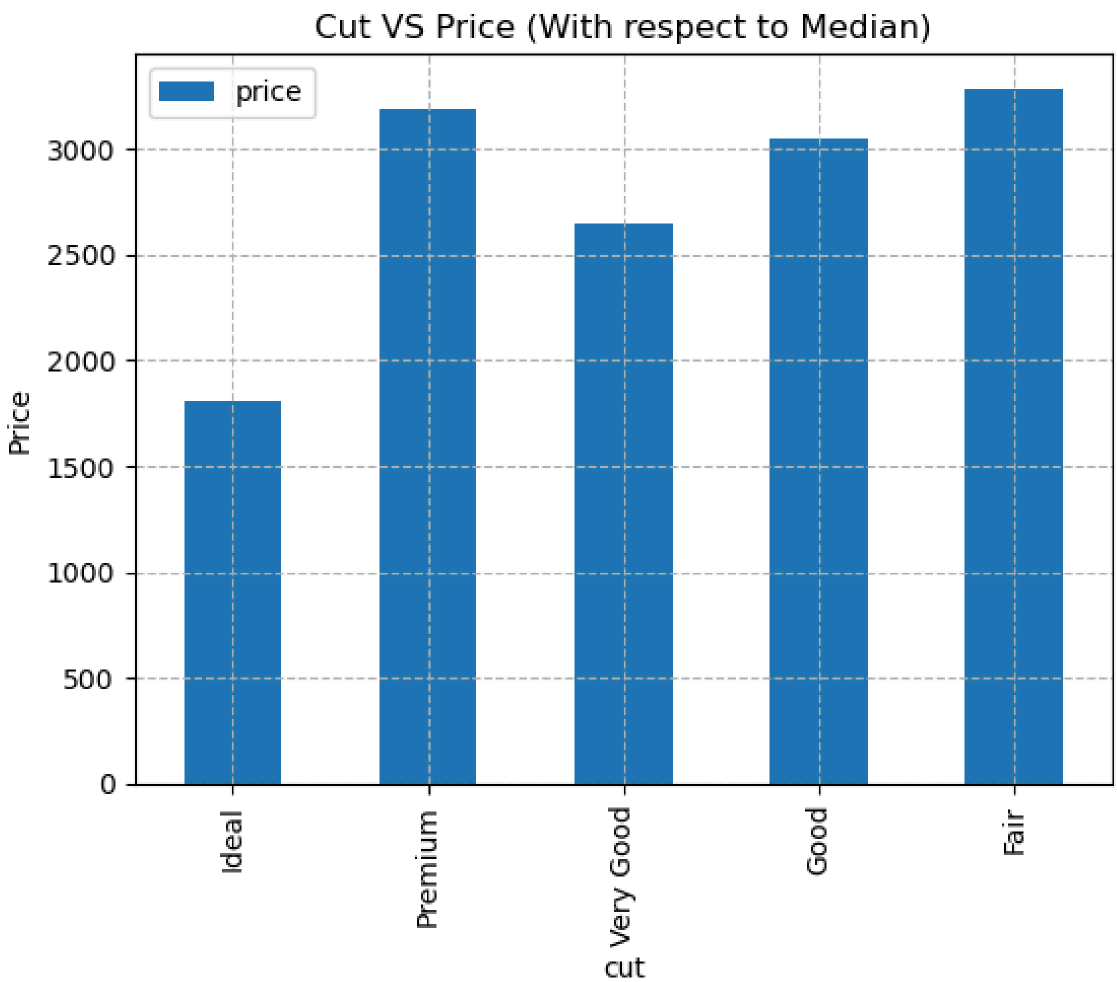
```
y = new_diamonds.groupby('cut')[["cut", 'price']].agg({
    "cut":"count",
    "price":"mean"
})
new_names = {'cut': 'Grouped Cut count', 'price':'Median of Group of Pr

y.rename(columns=new_names)
```

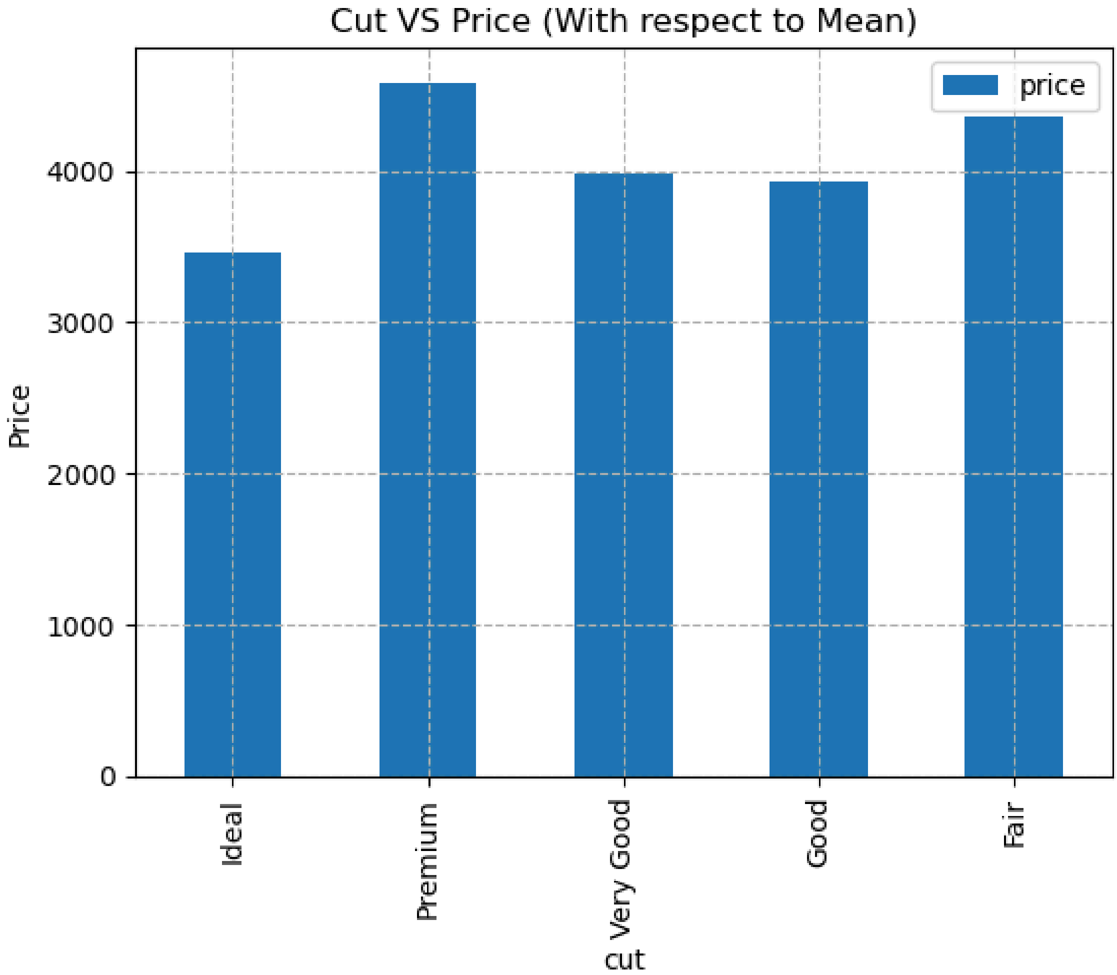
Out[16]:

	Grouped Cut count	Median of Group of Prices
cut		
Ideal	5412	3411.394863
Premium	3373	4550.726949
Very Good	3091	4019.195406
Good	1206	3932.620232
Fair	403	4491.404467

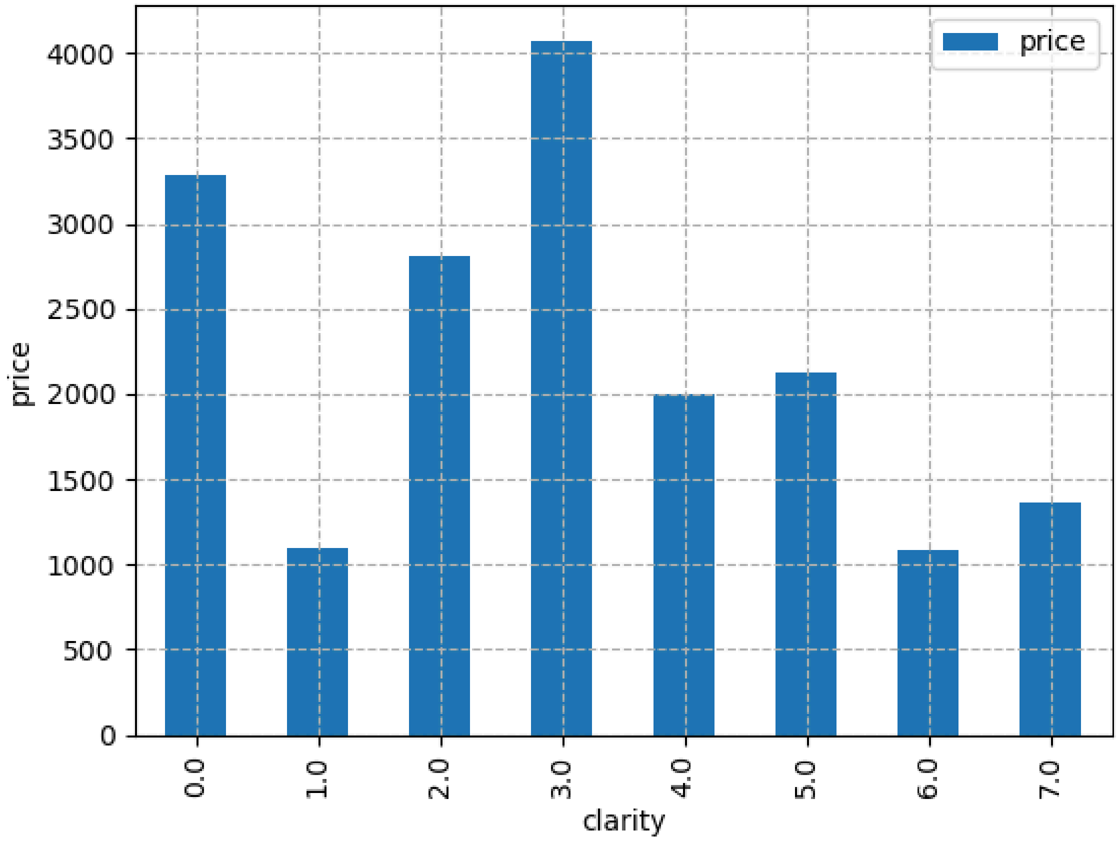
In [17]:

```
diamonds.groupby('cut')[['price']].median().plot(kind="bar")
plt.grid(linestyle="--")
plt.ylabel("Price")
plt.title("Cut VS Price (With respect to Median)")
plt.show()
diamonds.groupby('cut')[['price']].mean().plot(kind="bar")
plt.grid(linestyle="--")
plt.ylabel("Price")
plt.title("Cut VS Price (With respect to Mean)")
plt.show()
```

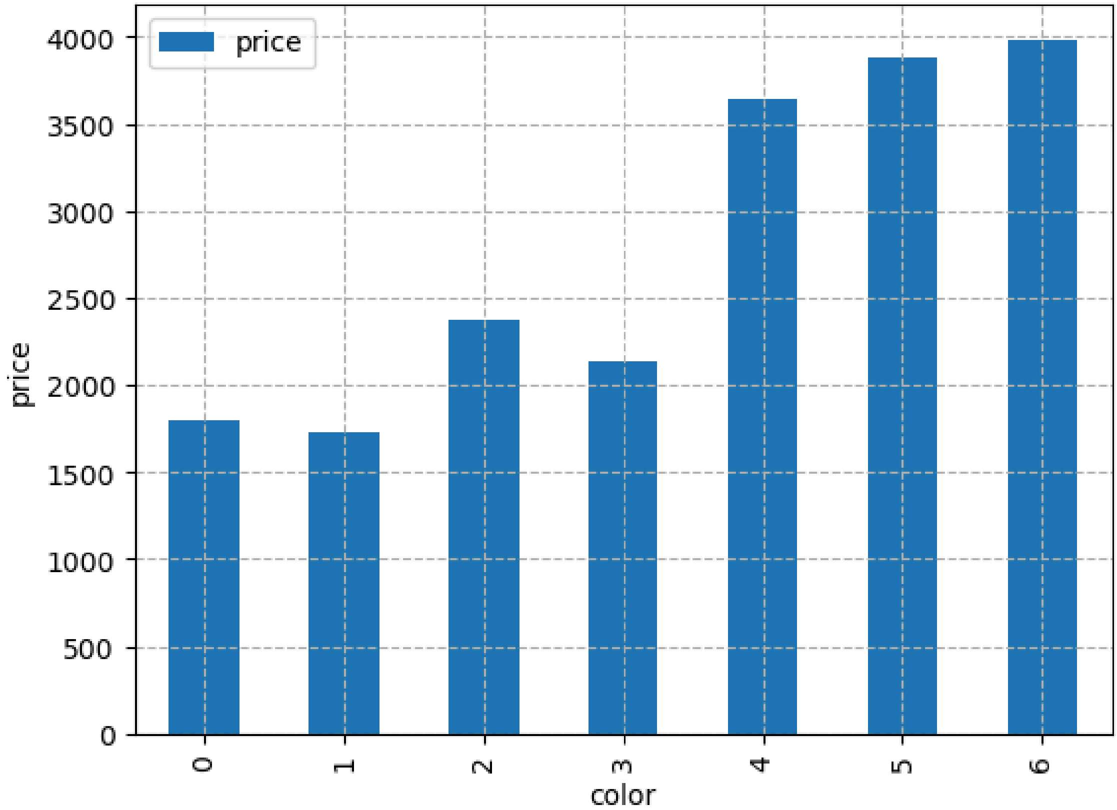



Cut VS Price (With respect to Mean)
price
4000
3000
2000
1000
0
Price
Ideal
Premium
Very Good
Good
Fair
cut

In [91]:
```
new_diamonds.groupby("clarity")[["price"]].median().plot(kind="bar")
plt.ylabel("price")
plt.grid(linestyle='--')
plt.show()
```



In [92]:
```
new_diamonds.groupby("color")[["price"]].median().plot(kind="bar")
plt.ylabel("price")
plt.grid(linestyle='--')
plt.show()
```



In [5]:
```
y = new_diamonds['price']
y
```

Out[5]:
```
15164     6095
45273     1662
28326      666
16964     6776
34871      878
         ...
4412      3610
8318      4390
26588    16304
18211     7392
7146      4179
Name: price, Length: 13485, dtype: int64
```

In [6]:

```
new_diamonds = new_diamonds.drop('price', axis=1)
new_diamonds
```

Out[6]:

	carat	cut	color	clarity	depth	table	x	y	z
15164	1.31	Premium	H	VS2	59.6	58.0	7.14	7.08	4.24
45273	0.54	Very Good	F	VS2	59.4	57.0	5.29	5.35	3.16
28326	0.33	Premium	G	VS1	61.9	58.0	4.43	4.46	2.75
16964	1.10	Very Good	E	VS2	61.9	55.0	6.61	6.67	4.11
34871	0.30	Very Good	G	VVS2	63.2	57.0	4.28	4.23	2.69
...	...	...	...	...	...	...	...	...	...
4412	1.01	Very Good	I	SI1	58.8	58.0	6.52	6.57	3.85
8318	1.00	Very Good	E	SI2	63.3	55.0	6.29	6.25	3.97
26588	2.40	Premium	J	SI1	59.7	58.0	8.75	8.71	5.21
18211	1.00	Premium	E	VS1	58.7	62.0	6.60	6.55	3.86
7146	0.93	Ideal	E	SI1	62.0	56.0	6.25	6.29	3.89

13485 rows × 9 columns

In [7]:

```
new_diamonds.insert(9,'price',y,True)
new_diamonds
```

Out[7]:

	carat	cut	color	clarity	depth	table	x	y	z	price
15164	1.31	Premium	H	VS2	59.6	58.0	7.14	7.08	4.24	6095
45273	0.54	Very Good	F	VS2	59.4	57.0	5.29	5.35	3.16	1662
28326	0.33	Premium	G	VS1	61.9	58.0	4.43	4.46	2.75	666
16964	1.10	Very Good	E	VS2	61.9	55.0	6.61	6.67	4.11	6776
34871	0.30	Very Good	G	VVS2	63.2	57.0	4.28	4.23	2.69	878
...	...	...	...	...	...	...	...	...	...	...
4412	1.01	Very Good	I	SI1	58.8	58.0	6.52	6.57	3.85	3610
8318	1.00	Very Good	E	SI2	63.3	55.0	6.29	6.25	3.97	4390
26588	2.40	Premium	J	SI1	59.7	58.0	8.75	8.71	5.21	16304
18211	1.00	Premium	E	VS1	58.7	62.0	6.60	6.55	3.86	7392
7146	0.93	Ideal	E	SI1	62.0	56.0	6.25	6.29	3.89	4179

13485 rows × 10 columns

In [22]:

```
from sklearn.preprocessing import OrdinalEncoder
encoder = OrdinalEncoder()
encoder.fit(np.asarray(new_diamonds['cut']).reshape(-1,1))
new_diamonds['cut'] = encoder.transform(np.asarray(new_diamonds['cut'])
```

In [23]:
```
encoder_clarity = OrdinalEncoder()
encoder_clarity.fit(np.asarray(new_diamonds['clarity']).reshape(-1,1))
new_diamonds['clarity'] = encoder_clarity.transform(np.asarray(new_diam
```

In [24]:
```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
new_diamonds['color']=encoder.fit_transform(new_diamonds['color'])
new_diamonds
```

Out[24]:

	carat	cut	color	clarity	depth	table	x	y	z	price
15164	1.31	3.0	4	5.0	59.6	58.0	7.14	7.08	4.24	6095
45273	0.54	4.0	2	5.0	59.4	57.0	5.29	5.35	3.16	1662
28326	0.33	3.0	3	4.0	61.9	58.0	4.43	4.46	2.75	666
16964	1.10	4.0	1	5.0	61.9	55.0	6.61	6.67	4.11	6776
34871	0.30	4.0	3	7.0	63.2	57.0	4.28	4.23	2.69	878
...	...	...	...	...	...	...	...	...	...	...
4412	1.01	4.0	5	2.0	58.8	58.0	6.52	6.57	3.85	3610
8318	1.00	4.0	1	3.0	63.3	55.0	6.29	6.25	3.97	4390
26588	2.40	3.0	6	2.0	59.7	58.0	8.75	8.71	5.21	16304
18211	1.00	3.0	1	4.0	58.7	62.0	6.60	6.55	3.86	7392
7146	0.93	2.0	1	2.0	62.0	56.0	6.25	6.29	3.89	4179

13485 rows × 10 columns

In [25]:
```
X = new_diamonds.iloc[:,0:9]
Y = new_diamonds.iloc[:,-1]
X
```

Out[25]:

	carat	cut	color	clarity	depth	table	x	y	z
15164	1.31	3.0	4	5.0	59.6	58.0	7.14	7.08	4.24
45273	0.54	4.0	2	5.0	59.4	57.0	5.29	5.35	3.16
28326	0.33	3.0	3	4.0	61.9	58.0	4.43	4.46	2.75
16964	1.10	4.0	1	5.0	61.9	55.0	6.61	6.67	4.11
34871	0.30	4.0	3	7.0	63.2	57.0	4.28	4.23	2.69
...	...	...	...	...	...	...	...	...	...
4412	1.01	4.0	5	2.0	58.8	58.0	6.52	6.57	3.85
8318	1.00	4.0	1	3.0	63.3	55.0	6.29	6.25	3.97
26588	2.40	3.0	6	2.0	59.7	58.0	8.75	8.71	5.21
18211	1.00	3.0	1	4.0	58.7	62.0	6.60	6.55	3.86
7146	0.93	2.0	1	2.0	62.0	56.0	6.25	6.29	3.89

13485 rows × 9 columns

In [26]:
```
Y
```

Out[26]:
```
15164     6095
45273     1662
28326      666
16964     6776
34871      878
         ...
4412      3610
8318      4390
26588    16304
18211     7392
7146      4179
Name: price, Length: 13485, dtype: int64
```

In [44]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2
print("X_train.shape = ", X_train.shape)
print("Y_train.shape = ", Y_train.shape)
print("X_test.shape = ", X_test.shape)
print("Y_test.shape = ", Y_test.shape)
```

```
X_train.shape =  (10788, 9)
Y_train.shape =  (10788,)
X_test.shape =  (2697, 9)
Y_test.shape =  (2697,)
```

In [64]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train_fit = sc.fit_transform(X_train)
X_test_fit = sc.transform(X_test)
```

In [66]:
```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train_fit, Y_train)
```

Out[66]:
```
LinearRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [68]:
```
predictions = regressor.predict(X_test_fit)
predictions
```

Out[68]:
```
array([ 9000.86843573, 12074.58596941,   926.85431458, ...,
         168.02196533,  6181.3200088 ,  1258.81618202])
```

In [70]:
```
print("Accuracy on the training data = ",regressor.score(X_train_fit, Y
print("Accuracy on the testing data = ",regressor.score(X_test_fit, Y_t
```

```
Accuracy on the training data =  0.8875250088875044
Accuracy on the testing data =  0.8922193544232553
```

In [55]:
```
# from sklearn import metrics

# actual = np.array(Y_test)
# print(actual)
# confusion_matrix = metrics.confusion_matrix(actual, predictions)
# confusion_matrix_display = metrics.ConfusionMatrixDisplay(confusion_m

# confusion_matrix_display.plot()
# plt.show()
```

The accuracy we have obtained in the sampled model is around 88-89%. Now, we are attempting to create another model wherein the entire 'Diamonds' Dataset is used and all 53,940 values are put to use to train and test the new Model.

Performing Encoding using LabelEncoder and OrdinalEncoder:

In [71]:
```
diamonds.head(5)
```

Out[71]:

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

In [135]:

```
from sklearn.preprocessing import OrdinalEncoder
encoder = OrdinalEncoder()
encoder.fit(np.asarray(diamonds['cut']).reshape(-1,1))
diamonds['cut'] = encoder.transform(np.asarray(diamonds['cut']).reshape

# from sklearn.preprocessing import LabelEncoder
encoder_clarity = OrdinalEncoder()
# diamonds['clarity']=encoder_clarity.fit_transform(diamonds['color'])
encoder_clarity.fit(np.asarray(diamonds['clarity']).reshape(-1,1))
diamonds['clarity'] = encoder_clarity.transform(np.asarray(diamonds['cl

# from sklearn.preprocessing import LabelEncoder
# encoder = LabelEncoder()
# diamonds['color']=encoder.fit_transform(diamonds['color'])
# diamonds

encoder_color = OrdinalEncoder()
encoder_color.fit(np.asarray(diamonds['color']).reshape(-1,1))
diamonds['color'] = encoder_color.transform(np.asarray(diamonds['color'
diamonds
```

Out[135]:

	carat	cut	color	clarity	depth	table	x	y	z	price
0	0.23	2.0	1.0	1.0	61.5	55.0	3.95	3.98	2.43	326
1	0.21	3.0	1.0	1.0	59.8	61.0	3.89	3.84	2.31	326
2	0.23	1.0	1.0	1.0	56.9	65.0	4.05	4.07	2.31	327
3	0.29	3.0	5.0	5.0	62.4	58.0	4.20	4.23	2.63	334
4	0.31	1.0	6.0	6.0	63.3	58.0	4.34	4.35	2.75	335
...	...	...	...	...	...	...	...	...	...	...
53935	0.72	2.0	0.0	0.0	60.8	57.0	5.75	5.76	3.50	2757
53936	0.72	1.0	0.0	0.0	63.1	55.0	5.69	5.75	3.61	2757
53937	0.70	4.0	0.0	0.0	62.8	60.0	5.66	5.68	3.56	2757
53938	0.86	3.0	4.0	4.0	61.0	58.0	6.15	6.12	3.74	2757
53939	0.75	2.0	0.0	0.0	62.2	55.0	5.83	5.87	3.64	2757

53940 rows × 10 columns

Reshaping the entire table to seperate the independent variables 'X' and the dependent variables 'Y':

In [136]:
```
y = diamonds['price']
diamonds = diamonds.drop('price', axis=1)
diamonds.insert(9,'price',y,True)
diamonds
```

Out[136]:

	carat	cut	color	clarity	depth	table	x	y	z	price
0	0.23	2.0	1.0	1.0	61.5	55.0	3.95	3.98	2.43	326
1	0.21	3.0	1.0	1.0	59.8	61.0	3.89	3.84	2.31	326
2	0.23	1.0	1.0	1.0	56.9	65.0	4.05	4.07	2.31	327
3	0.29	3.0	5.0	5.0	62.4	58.0	4.20	4.23	2.63	334
4	0.31	1.0	6.0	6.0	63.3	58.0	4.34	4.35	2.75	335
...	...	...	...	...	...	...	...	...	...	...
53935	0.72	2.0	0.0	0.0	60.8	57.0	5.75	5.76	3.50	2757
53936	0.72	1.0	0.0	0.0	63.1	55.0	5.69	5.75	3.61	2757
53937	0.70	4.0	0.0	0.0	62.8	60.0	5.66	5.68	3.56	2757
53938	0.86	3.0	4.0	4.0	61.0	58.0	6.15	6.12	3.74	2757
53939	0.75	2.0	0.0	0.0	62.2	55.0	5.83	5.87	3.64	2757

53940 rows × 10 columns

In [137]:
```
X = diamonds.iloc[:,0:9]
Y = diamonds.iloc[:,-1]
X.head(5)
```

Out[137]:

	carat	cut	color	clarity	depth	table	x	y	z
0	0.23	2.0	1.0	1.0	61.5	55.0	3.95	3.98	2.43
1	0.21	3.0	1.0	1.0	59.8	61.0	3.89	3.84	2.31
2	0.23	1.0	1.0	1.0	56.9	65.0	4.05	4.07	2.31
3	0.29	3.0	5.0	5.0	62.4	58.0	4.20	4.23	2.63
4	0.31	1.0	6.0	6.0	63.3	58.0	4.34	4.35	2.75

In [138]:
```
Y.shape
```

Out[138]:
```
(53940,)
```

Spliting the Independent and Dependent Variables into Training and Testing data & performing Standard Scaling operations i.e. Normalization w.r.t columns:

In [149]:
```
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
print("X_train.shape = ", X_train.shape)
print("Y_train.shape = ", Y_train.shape)
print("X_test.shape = ", X_test.shape)
print("Y_test.shape = ", Y_test.shape)
```

```
X_train.shape =  (37758, 9)
Y_train.shape =  (37758,)
X_test.shape =  (16182, 9)
Y_test.shape =  (16182,)
```

In [150]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train_fit = sc.fit_transform(X_train)
X_test_fit = sc.transform(X_test)
```

Creation & training of the Model and Checking of accuracy score on the training and testing data:

In [151]:
```
from sklearn.linear_model import LinearRegression
regressor1 = LinearRegression()
regressor1.fit(X_train_fit, Y_train)
```

Out[151]:
```
LinearRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [152]:
```
predictions = regressor.predict(X_test_fit)
predictions
```

Out[152]:
```
array([-112.25700196, 2245.92109293,  797.76615612, ..., 9825.02834466,
       2853.59749537,  886.51514112])
```

In [153]:
```
print("Accuracy on the training data = ",regressor.score(X_train_fit, Y
print("Accuracy on the testing data = ",regressor.score(X_test_fit, Y_t
```

```
Accuracy on the training data =  0.8530516733195588
Accuracy on the testing data =  0.8535198937847488
```

Hence, we can conclude that we have performed the Exploratory Data Analysis (EDA) upon the `Diamonds Dataset` and tried to find some of the important features and relationships between various schemas/columns present in the dataset. According to the analysis, the dataset consisted of Linear Relationships between the columns consisting of numerical data, as depicted in the pairplot printed above. However, for Categorical Data, pairplots cannot illustrate relations among them.

To solve this problem, we manually tried to plot out relations between the Categorical Columns and the `Price` column in the dataset to check if those columns affected the 'Y' values in our dataset. There were 3 Categorical Columns, namely: 'Cut', 'Color', and

'Clarity'. 'Cut' and 'Color' show some ordinal trend since these factors directly affect the price of the diamonds. Hence, an **Ordinal Encoder** is used here to encode the possible classes present in the respective columns.

StandardScaler is used to perform **Normalization** throughout the table once the Encoding operation is performed. It attempts to scale up or down the features to a common scale which in-turn leads to improvement in training of the model and provides us with better accuracies.

During splitting of the data into training and testing data, we are trying to keep 70% of data for training and 30% of data for testing purposes. The `random_state` value is taken to be 42 in this case. The `random_state` feature defines the number of values from which sampling can be performed or samples can be collected for segregation between training and testing datasets with the provided dataset sizes.

In the model trained, we have achieved the following accuracies:

Training Accuracy: 85.3051%

Testing Accuracy: 85.3519%