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 [1]:

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

In [2]:

```python
diamonds = sns.load_dataset("diamonds")
```

In [3]:

```python
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]:

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

Out[5]:

carat	cut	color	clarity	depth	table	price	x	y	z

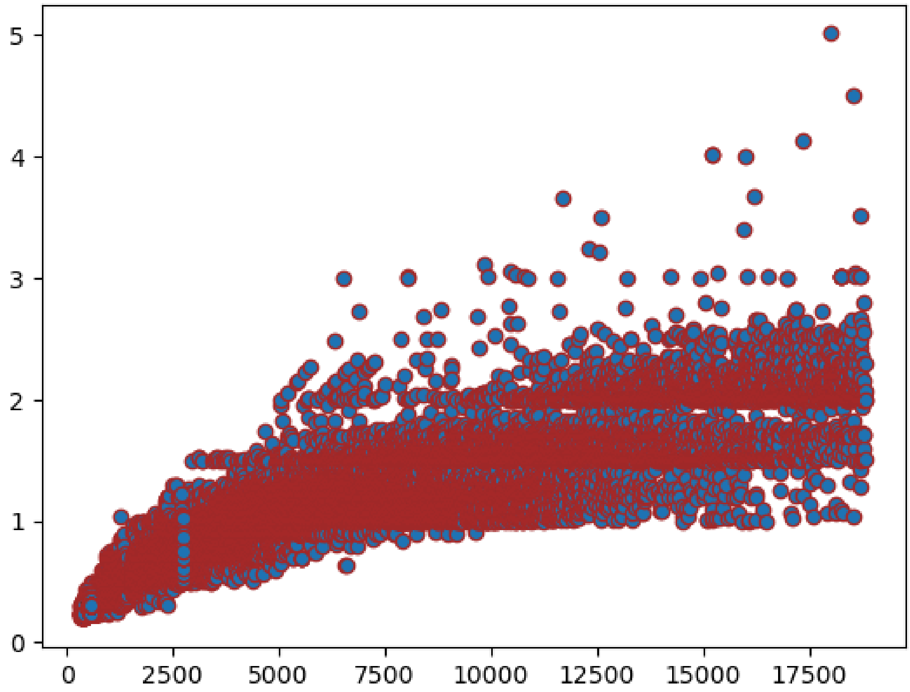
In [6]:
```
diamonds.describe()
```

Out[6]:

	carat	depth	table	price	x	y	z
count	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000
mean	0.797940	61.749405	57.457184	3932.799722	5.731157	5.734526	3.538734
std	0.474011	1.432621	2.234491	3989.439738	1.121761	1.142135	0.705699
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	5324.250000	6.540000	6.540000	4.040000
max	5.010000	79.000000	95.000000	18823.000000	10.740000	58.900000	31.800000

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

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



Trying to take samples from the dataset:

In [8]:
```
new_diamonds = diamonds.sample(frac=0.25)
new_diamonds
```

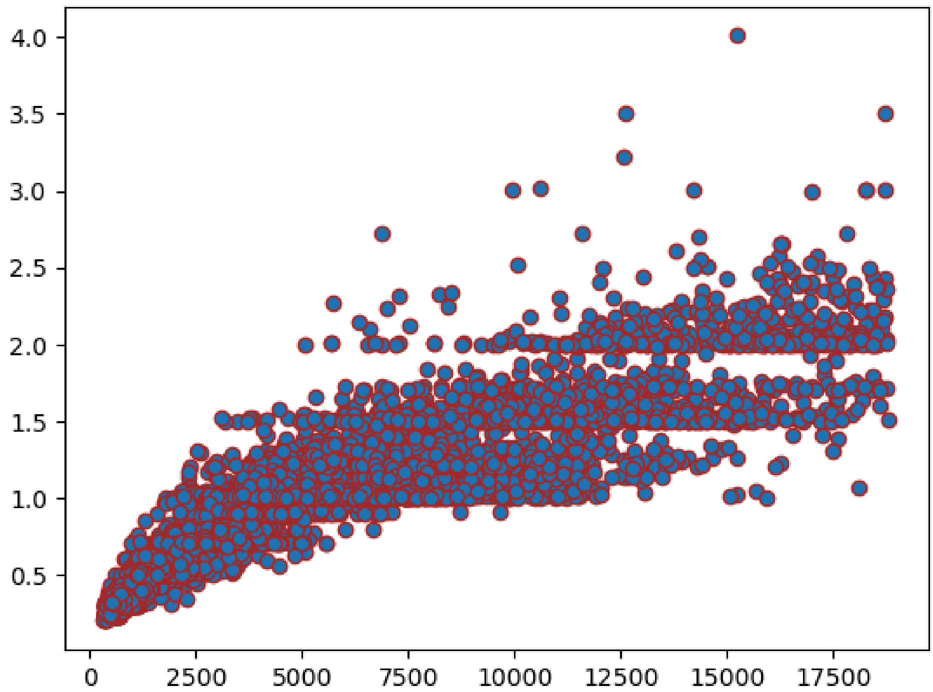
Out[8]:

	carat	cut	color	clarity	depth	table	price	x	y	z
53201	0.71	Ideal	F	SI1	61.5	56.0	2633	5.76	5.69	3.52
24331	2.02	Fair	I	SI2	64.5	60.0	12592	7.94	7.82	5.08
19051	1.06	Ideal	G	VVS2	62.9	56.0	7836	6.60	6.56	4.14
51227	0.70	Fair	D	SI1	64.9	64.0	2352	5.57	5.50	3.59
26795	2.00	Ideal	E	SI2	62.2	57.0	16650	8.11	8.09	5.04
...	...	...	...	...	...	...	...	...	...	...
28430	0.30	Ideal	D	VS2	61.4	58.0	670	4.29	4.31	2.64
7536	0.70	Very Good	D	VVS1	62.7	54.0	4244	5.67	5.71	3.57
23172	1.51	Premium	H	VS2	61.2	58.0	11188	7.40	7.36	4.52
24790	2.00	Premium	J	VS2	60.8	62.0	13162	8.12	8.09	4.93
50914	0.70	Very Good	E	SI1	63.4	56.0	2318	5.70	5.60	3.58

13485 rows × 10 columns

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

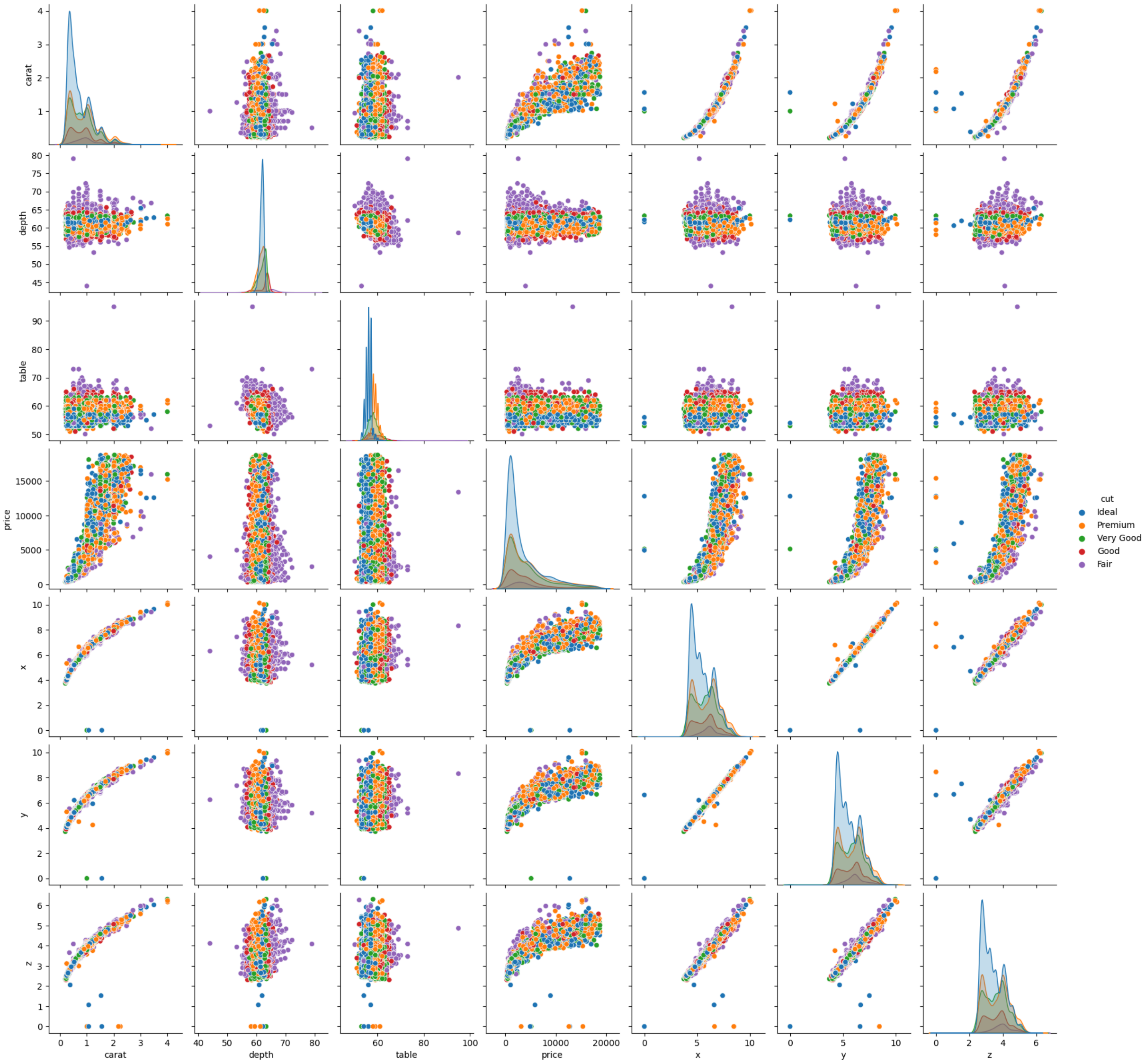
Out[8]: <matplotlib.collections.PathCollection at 0x278999156d0>



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

```
c:\Users\Admin\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The figur
e 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 Variables : {'carat', 'depth', 'table', 'price', 'x', 'y', 'z'}
```

In [9]:
```
list1 = list(new_diamonds.columns)
list1
```

Out[9]: ['carat', 'cut', 'color', 'clarity', 'depth', 'table', 'price', 'x', 'y', 'z']

In [10]:
```
categorical = ['cut', 'clarity', 'color']
numerical = list(set(list1)-set(categorical))
numerical
```

Out[10]: ['carat', 'depth', 'y', 'z', 'table', 'x', 'price']

In [11]:
```
new_diamonds.groupby('carat')[['carat']].count()
```

Out[11]:

carat	carat
0.20	5
0.21	2
0.22	2
0.23	74
0.24	52
...	...
3.04	2
3.40	1
3.51	1
4.01	1
4.50	1

237 rows × 1 columns

In [12]:
```
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'].count())
```

```
              price
carat
0.20     367.000000
0.21     390.000000
0.22     404.000000
0.23     479.972973
0.24     501.692308
...             ...
3.04   16956.500000
3.40   15964.000000
3.51   18701.000000
4.01   15223.000000
4.50   18531.000000

[237 rows x 1 columns]
Mean price =  price    8430.716931
dtype: float64
Median price =  price    7975.277778
dtype: float64
Max price =  price    18701.0
dtype: float64
Min price =  price    367.0
dtype: float64
Total Types of Carats of Diamonds available :  237
```

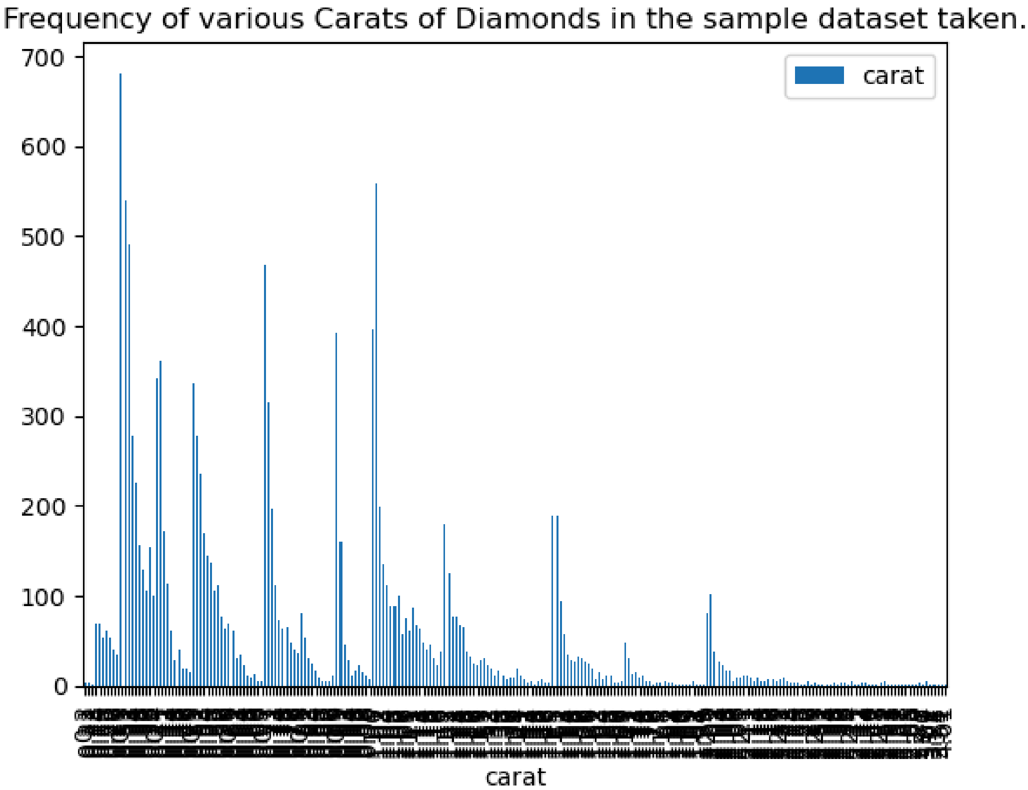
In [13]:

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

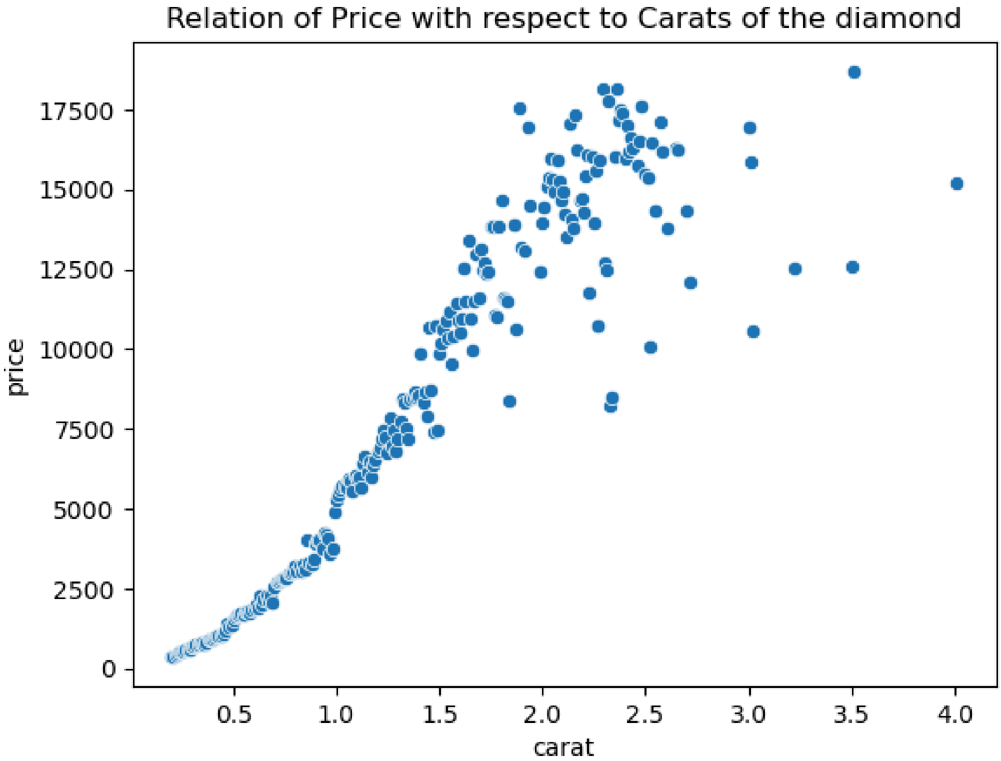
```
[0.2, 0.21, 0.22, 0.23, 0.24, 0.25, 0.26, 0.27, 0.28, 0.29, 0.3, 0.31, 0.32, 0.33, 0.34,
0.35, 0.36, 0.37, 0.38, 0.39, 0.4, 0.41, 0.42, 0.43, 0.44, 0.45, 0.46, 0.47, 0.48, 0.49,
0.5, 0.51, 0.52, 0.53, 0.54, 0.55, 0.56, 0.57, 0.58, 0.59, 0.6, 0.61, 0.62, 0.63, 0.64, 0.
65, 0.66, 0.67, 0.68, 0.69, 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, 0.84, 0.85, 0.86, 0.87, 0.88, 0.89, 0.9, 0.91, 0.92, 0.93, 0.94, 0.9
5, 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, 1.09, 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, 1.34, 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, 1.49, 1.5, 1.51, 1.52, 1.53, 1.54, 1.55, 1.56, 1.5
7, 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, 1.84, 1.85, 1.86, 1.88,
1.89, 1.9, 1.91, 1.93, 1.95, 1.96, 1.97, 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, 2.14, 2.15, 2.16, 2.17, 2.18, 2.19, 2.2, 2.21, 2.22, 2.2
3, 2.24, 2.25, 2.26, 2.27, 2.28, 2.29, 2.3, 2.31, 2.32, 2.33, 2.34, 2.35, 2.37, 2.38, 2.3
9, 2.4, 2.41, 2.43, 2.46, 2.47, 2.48, 2.49, 2.5, 2.51, 2.52, 2.53, 2.54, 2.55, 2.56, 2.58,
2.72, 2.8, 3.0, 3.01, 3.04, 3.4, 3.51, 4.01, 4.5]
```

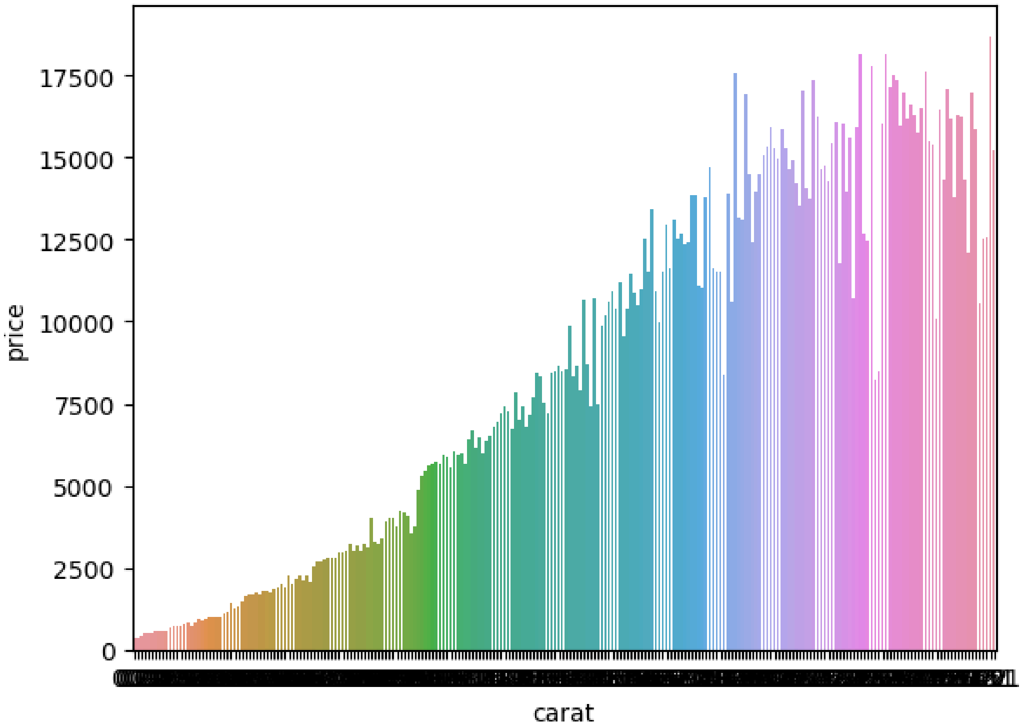
In [14]:

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

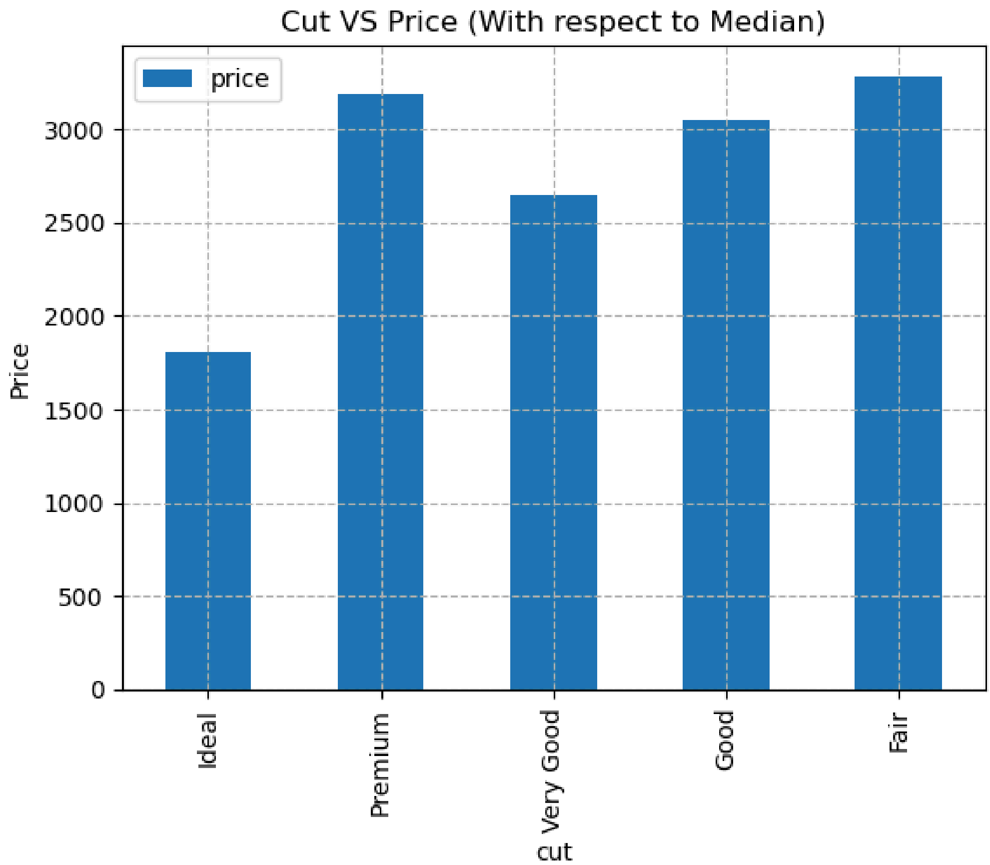
In [14]:

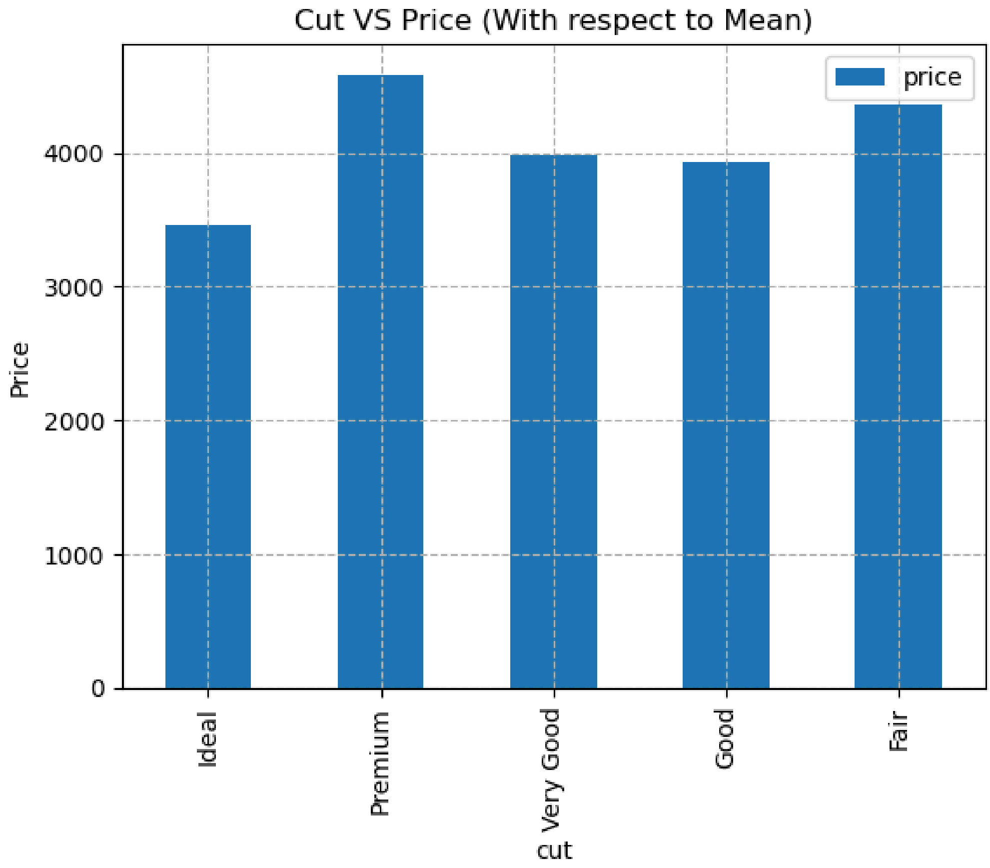
```python
y = new_diamonds.groupby('cut')[["cut", 'price']].agg({
    "cut":"count",
    "price":"mean"
})
new_names = {'cut': 'Grouped Cut count', 'price':'Median of Group of Prices'}

y.rename(columns=new_names)
```

Out[14]:

cut	Grouped Cut count	Median of Group of Prices
Ideal	5401	3491.362155
Premium	3425	4580.379562
Very Good	2977	4115.770910
Good	1290	3863.759690
Fair	392	4302.711735

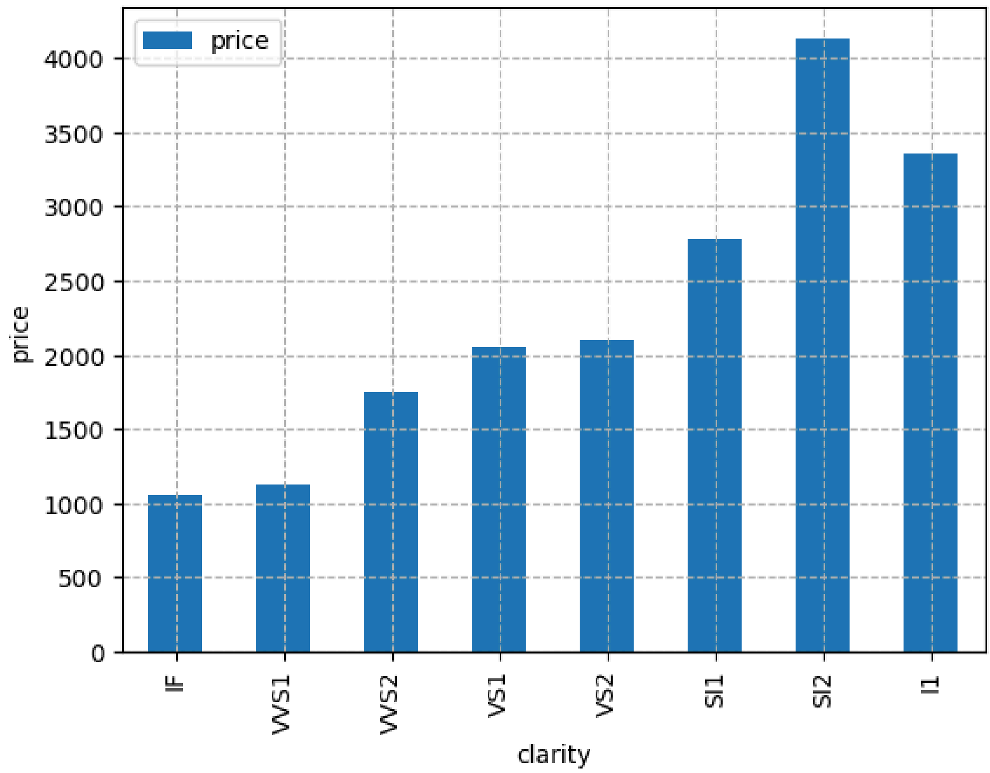
In [15]:
```
diamonds.groupby('cut')[['price']].median().plot(kind="bar")
plt.grid(linestyle="--")
plt.ylabel("Price")
plt.title("Cut VS Price (With respect to Median)")
plt.show()
diamonds.groupby('cut')[['price']].mean().plot(kind="bar")
plt.grid(linestyle="--")
plt.ylabel("Price")
plt.title("Cut VS Price (With respect to Mean)")
plt.show()
```



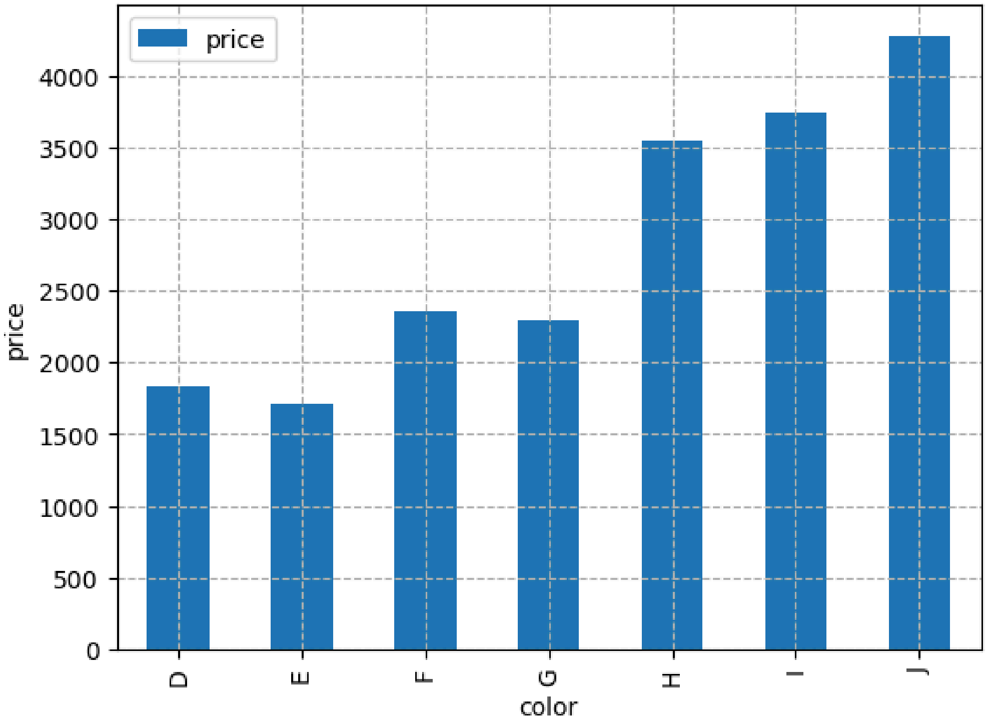


In [16]:

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



In [18]:
```
new_diamonds.groupby("color")[["price"]].median().plot(kind="bar")
plt.ylabel("price")
plt.grid(linestyle='--')
plt.show()
```



In [19]:
```
y = new_diamonds['price']
y
```

Out[19]:
```
53201     2633
24331    12592
19051     7836
51227     2352
26795    16650
         ...
28430      670
7536      4244
23172    11188
24790    13162
50914     2318
Name: price, Length: 13485, dtype: int64
```

In [20]:
```
new_diamonds = new_diamonds.drop('price', axis=1)
new_diamonds
```

Out[20]:

	carat	cut	color	clarity	depth	table	x	y	z
53201	0.71	Ideal	F	SI1	61.5	56.0	5.76	5.69	3.52
24331	2.02	Fair	I	SI2	64.5	60.0	7.94	7.82	5.08
19051	1.06	Ideal	G	VVS2	62.9	56.0	6.60	6.56	4.14
51227	0.70	Fair	D	SI1	64.9	64.0	5.57	5.50	3.59
26795	2.00	Ideal	E	SI2	62.2	57.0	8.11	8.09	5.04
...	...	...	...	...	...	...	...	...	...
28430	0.30	Ideal	D	VS2	61.4	58.0	4.29	4.31	2.64
7536	0.70	Very Good	D	VVS1	62.7	54.0	5.67	5.71	3.57
23172	1.51	Premium	H	VS2	61.2	58.0	7.40	7.36	4.52
24790	2.00	Premium	J	VS2	60.8	62.0	8.12	8.09	4.93
50914	0.70	Very Good	E	SI1	63.4	56.0	5.70	5.60	3.58

13485 rows × 9 columns

In [21]:
```
new_diamonds.insert(9,'price',y,True)
new_diamonds
```

Out[21]:

	carat	cut	color	clarity	depth	table	x	y	z	price
53201	0.71	Ideal	F	SI1	61.5	56.0	5.76	5.69	3.52	2633
24331	2.02	Fair	I	SI2	64.5	60.0	7.94	7.82	5.08	12592
19051	1.06	Ideal	G	VVS2	62.9	56.0	6.60	6.56	4.14	7836
51227	0.70	Fair	D	SI1	64.9	64.0	5.57	5.50	3.59	2352
26795	2.00	Ideal	E	SI2	62.2	57.0	8.11	8.09	5.04	16650
...	...	...	...	...	...	...	...	...	...	...
28430	0.30	Ideal	D	VS2	61.4	58.0	4.29	4.31	2.64	670
7536	0.70	Very Good	D	VVS1	62.7	54.0	5.67	5.71	3.57	4244
23172	1.51	Premium	H	VS2	61.2	58.0	7.40	7.36	4.52	11188
24790	2.00	Premium	J	VS2	60.8	62.0	8.12	8.09	4.93	13162
50914	0.70	Very Good	E	SI1	63.4	56.0	5.70	5.60	3.58	2318

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']).reshape(-1,1))
```

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_diamonds['clarity'])
```

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
53201	0.71	2.0	2	2.0	61.5	56.0	5.76	5.69	3.52	2633
24331	2.02	0.0	5	3.0	64.5	60.0	7.94	7.82	5.08	12592
19051	1.06	2.0	3	7.0	62.9	56.0	6.60	6.56	4.14	7836
51227	0.70	0.0	0	2.0	64.9	64.0	5.57	5.50	3.59	2352
26795	2.00	2.0	1	3.0	62.2	57.0	8.11	8.09	5.04	16650
...	...	...	...	...	...	...	...	...	...	...
28430	0.30	2.0	0	5.0	61.4	58.0	4.29	4.31	2.64	670
7536	0.70	4.0	0	6.0	62.7	54.0	5.67	5.71	3.57	4244
23172	1.51	3.0	4	5.0	61.2	58.0	7.40	7.36	4.52	11188
24790	2.00	3.0	6	5.0	60.8	62.0	8.12	8.09	4.93	13162
50914	0.70	4.0	1	2.0	63.4	56.0	5.70	5.60	3.58	2318

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
53201	0.71	2.0	2	2.0	61.5	56.0	5.76	5.69	3.52
24331	2.02	0.0	5	3.0	64.5	60.0	7.94	7.82	5.08
19051	1.06	2.0	3	7.0	62.9	56.0	6.60	6.56	4.14
51227	0.70	0.0	0	2.0	64.9	64.0	5.57	5.50	3.59
26795	2.00	2.0	1	3.0	62.2	57.0	8.11	8.09	5.04
...	...	...	...	...	...	...	...	...	...
28430	0.30	2.0	0	5.0	61.4	58.0	4.29	4.31	2.64
7536	0.70	4.0	0	6.0	62.7	54.0	5.67	5.71	3.57
23172	1.51	3.0	4	5.0	61.2	58.0	7.40	7.36	4.52
24790	2.00	3.0	6	5.0	60.8	62.0	8.12	8.09	4.93
50914	0.70	4.0	1	2.0	63.4	56.0	5.70	5.60	3.58

13485 rows × 9 columns

In [26]:

```
Y
```

Out[26]:

```
53201     2633
24331    12592
19051     7836
51227     2352
26795    16650
         ...  
28430      670
7536      4244
23172    11188
24790    13162
50914     2318
Name: price, Length: 13485, dtype: int64
```

In [47]:
```
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, random_state=5
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 [48]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train_fit = sc.fit_transform(X_train)
X_test_fit = sc.transform(X_test)
```

In [49]:
```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train_fit, Y_train)
```

Out[49]:
```
▾ LinearRegression
LinearRegression()
```

In [50]:
```
predictions = regressor.predict(X_test_fit)
predictions
```

Out[50]:
```
array([  719.87360582,  2590.59105015, 16399.33580473, ...,
        4593.06028652,  3988.54397506,   297.52181535])
```

In [51]:
```
print("Accuracy on the training data = ",regressor.score(X_train_fit, Y_train))
print("Accuracy on the testing data = ",regressor.score(X_test_fit, Y_test))
```

```
Accuracy on the training data =  0.8874808105246443
Accuracy on the testing data =  0.8875189771036172
```

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 [52]:
```
diamonds.head(5)
```

Out[52]:

	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 [97]:

```
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(-1,1))

# from sklearn.preprocessing import LabelEncoder
encoder_clarity = LabelEncoder()
diamonds['clarity']=encoder_clarity.fit_transform(diamonds['clarity'])

# encoder_clarity = OrdinalEncoder()
# encoder_clarity.fit(np.asarray(diamonds['clarity']).reshape(-1,1))
# diamonds['clarity'] = encoder_clarity.transform(np.asarray(diamonds['clarity']).resha

# 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']).reshape(-1,1)
diamonds
```

Out[97]:

	carat	cut	color	clarity	depth	table	x	y	z	price
0	0.23	2.0	1.0	3	61.5	55.0	3.95	3.98	2.43	326
1	0.21	3.0	1.0	2	59.8	61.0	3.89	3.84	2.31	326
2	0.23	1.0	1.0	4	56.9	65.0	4.05	4.07	2.31	327
3	0.29	3.0	5.0	5	62.4	58.0	4.20	4.23	2.63	334
4	0.31	1.0	6.0	3	63.3	58.0	4.34	4.35	2.75	335
...	...	...	...	...	...	...	...	...	...	...
53935	0.72	2.0	0.0	2	60.8	57.0	5.75	5.76	3.50	2757
53936	0.72	1.0	0.0	2	63.1	55.0	5.69	5.75	3.61	2757
53937	0.70	4.0	0.0	2	62.8	60.0	5.66	5.68	3.56	2757
53938	0.86	3.0	4.0	3	61.0	58.0	6.15	6.12	3.74	2757
53939	0.75	2.0	0.0	3	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 [98]:
```
y = diamonds['price']
diamonds = diamonds.drop('price', axis=1)
diamonds.insert(9,'price',y,True)
diamonds
```

Out[98]:

	carat	cut	color	clarity	depth	table	x	y	z	price
0	0.23	2.0	1.0	3	61.5	55.0	3.95	3.98	2.43	326
1	0.21	3.0	1.0	2	59.8	61.0	3.89	3.84	2.31	326
2	0.23	1.0	1.0	4	56.9	65.0	4.05	4.07	2.31	327
3	0.29	3.0	5.0	5	62.4	58.0	4.20	4.23	2.63	334
4	0.31	1.0	6.0	3	63.3	58.0	4.34	4.35	2.75	335
...	...	...	...	...	...	...	...	...	...	...
53935	0.72	2.0	0.0	2	60.8	57.0	5.75	5.76	3.50	2757
53936	0.72	1.0	0.0	2	63.1	55.0	5.69	5.75	3.61	2757
53937	0.70	4.0	0.0	2	62.8	60.0	5.66	5.68	3.56	2757
53938	0.86	3.0	4.0	3	61.0	58.0	6.15	6.12	3.74	2757
53939	0.75	2.0	0.0	3	62.2	55.0	5.83	5.87	3.64	2757

53940 rows × 10 columns

In [99]:
```
X = diamonds.iloc[:,0:9]
Y = diamonds.iloc[:,-1]
X.head(5)
```

Out[99]:

	carat	cut	color	clarity	depth	table	x	y	z
0	0.23	2.0	1.0	3	61.5	55.0	3.95	3.98	2.43
1	0.21	3.0	1.0	2	59.8	61.0	3.89	3.84	2.31
2	0.23	1.0	1.0	4	56.9	65.0	4.05	4.07	2.31
3	0.29	3.0	5.0	5	62.4	58.0	4.20	4.23	2.63
4	0.31	1.0	6.0	3	63.3	58.0	4.34	4.35	2.75

In [100]:
```
Y.shape
```

Out[100]:
```
(53940,)
```

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

In [101]:
```
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_state=4
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 [102]:
```
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 [103]:
```
from sklearn.linear_model import LinearRegression
regressor1 = LinearRegression()
regressor1.fit(X_train_fit, Y_train)
```

Out[103]:
```
▾ LinearRegression
LinearRegression()
```

In [104]:
```
predictions = regressor1.predict(X_test_fit)
predictions
```

Out[104]:
```
array([  737.67735333,   905.76746573, 10733.52664698, ...,
        5785.34901884,   938.90172826,  7040.33120009])
```

In [105]:
```
print("Accuracy on the training data = ",regressor1.score(X_train_fit, Y_train))
print("Accuracy on the testing data = ",regressor1.score(X_test_fit, Y_test))
```

```
Accuracy on the training data =  0.8840114198094176
Accuracy on the testing data =  0.8874077946790888
```

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: 88.4011%

Testing Accuracy: 88.7407%

No signs of Overfitting have been found yet since the testing accuracy is slightly better than training accuracy.