In [35]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [14]:
```
df=pd.read_csv('cubic_zirconia.csv')
```

In [15]:
```
df
```

Out[15]:

	Unnamed: 0	carat	cut	color	clarity	depth	table	x	y	z	price
0	1	0.30	Ideal	E	SI1	62.1	58.0	4.27	4.29	2.66	499
1	2	0.33	Premium	G	IF	60.8	58.0	4.42	4.46	2.70	984
2	3	0.90	Very Good	E	VVS2	62.2	60.0	6.04	6.12	3.78	6289
3	4	0.42	Ideal	F	VS1	61.6	56.0	4.82	4.80	2.96	1082
4	5	0.31	Ideal	F	VVS1	60.4	59.0	4.35	4.43	2.65	779
...	...	...	...	...	...	...	...	...	...	...	...
26962	26963	1.11	Premium	G	SI1	62.3	58.0	6.61	6.52	4.09	5408
26963	26964	0.33	Ideal	H	IF	61.9	55.0	4.44	4.42	2.74	1114
26964	26965	0.51	Premium	E	VS2	61.7	58.0	5.12	5.15	3.17	1656
26965	26966	0.27	Very Good	F	VVS2	61.8	56.0	4.19	4.20	2.60	682
26966	26967	1.25	Premium	J	SI1	62.0	58.0	6.90	6.88	4.27	5166

26967 rows × 11 columns

Basic EDA steps to understand the data

In [16]:
```
df.head()
```

Out[16]:

	Unnamed: 0	carat	cut	color	clarity	depth	table	x	y	z	price
0	1	0.30	Ideal	E	SI1	62.1	58.0	4.27	4.29	2.66	499
1	2	0.33	Premium	G	IF	60.8	58.0	4.42	4.46	2.70	984
2	3	0.90	Very Good	E	VVS2	62.2	60.0	6.04	6.12	3.78	6289
3	4	0.42	Ideal	F	VS1	61.6	56.0	4.82	4.80	2.96	1082
4	5	0.31	Ideal	F	VVS1	60.4	59.0	4.35	4.43	2.65	779

In [17]:
```
df.tail()
```

Out[17]:

	Unnamed: 0	carat	cut	color	clarity	depth	table	x	y	z	price
26962	26963	1.11	Premium	G	SI1	62.3	58.0	6.61	6.52	4.09	5408
26963	26964	0.33	Ideal	H	IF	61.9	55.0	4.44	4.42	2.74	1114
26964	26965	0.51	Premium	E	VS2	61.7	58.0	5.12	5.15	3.17	1656
26965	26966	0.27	Very Good	F	VVS2	61.8	56.0	4.19	4.20	2.60	682
26966	26967	1.25	Premium	J	SI1	62.0	58.0	6.90	6.88	4.27	5166

In [18]:
```
df.describe()
```

Out[18]:

	Unnamed: 0	carat	depth	table	x	y	
count	26967.000000	26967.000000	26270.000000	26967.000000	26967.000000	26967.000000	26967
mean	13484.000000	0.798375	61.745147	57.456080	5.729854	5.733569	3
std	7784.846691	0.477745	1.412860	2.232068	1.128516	1.166058	0
min	1.000000	0.200000	50.800000	49.000000	0.000000	0.000000	0
25%	6742.500000	0.400000	61.000000	56.000000	4.710000	4.710000	2
50%	13484.000000	0.700000	61.800000	57.000000	5.690000	5.710000	3
75%	20225.500000	1.050000	62.500000	59.000000	6.550000	6.540000	4
max	26967.000000	4.500000	73.600000	79.000000	10.230000	58.900000	31

In [19]:
```
df.isnull().sum()
```

Out[19]:
```
Unnamed: 0      0
carat           0
cut             0
color           0
clarity         0
depth         697
table           0
x               0
y               0
z               0
price           0
dtype: int64
```

Null values are found in 'depth' column

In [20]:
```
df.shape
```

Out[20]:
```
(26967, 11)
```

In [21]:
```
df.dtypes
```

Out[21]:
```
Unnamed: 0      int64
carat         float64
cut            object
color          object
clarity        object
depth         float64
table         float64
x             float64
y             float64
z             float64
price           int64
dtype: object
```

We observed that - 1 Dataset has 26967 rows and 11 columns. 2 The 1st column is an index (Unnamed:0),it contains only serial numbers.We can remove it. 3 The data types shows that the 3 columns(cut,color,clarity) have object features, there are 3 columns having integer feature,and remaining 6 columns have float feature. 4 Column 'depth' is having null values.

In [22]:
```
## To drop Unnamed:0 column ,as it is serial number and no use for model
df1=df.drop('Unnamed: 0',axis=1)
df1
```

Out[22]:

	carat	cut	color	clarity	depth	table	x	y	z	price
0	0.30	Ideal	E	SI1	62.1	58.0	4.27	4.29	2.66	499
1	0.33	Premium	G	IF	60.8	58.0	4.42	4.46	2.70	984
2	0.90	Very Good	E	VVS2	62.2	60.0	6.04	6.12	3.78	6289
3	0.42	Ideal	F	VS1	61.6	56.0	4.82	4.80	2.96	1082
4	0.31	Ideal	F	VVS1	60.4	59.0	4.35	4.43	2.65	779
...	...	...	...	...	...	...	...	...	...	...
26962	1.11	Premium	G	SI1	62.3	58.0	6.61	6.52	4.09	5408
26963	0.33	Ideal	H	IF	61.9	55.0	4.44	4.42	2.74	1114
26964	0.51	Premium	E	VS2	61.7	58.0	5.12	5.15	3.17	1656
26965	0.27	Very Good	F	VVS2	61.8	56.0	4.19	4.20	2.60	682
26966	1.25	Premium	J	SI1	62.0	58.0	6.90	6.88	4.27	5166

26967 rows × 10 columns

In [23]:
```
df1.describe()
```

Out[23]:

	carat	depth	table	x	y	z	
count	26967.000000	26270.000000	26967.000000	26967.000000	26967.000000	26967.000000	26967
mean	0.798375	61.745147	57.456080	5.729854	5.733569	3.538057	3939
std	0.477745	1.412860	2.232068	1.128516	1.166058	0.720624	4024
min	0.200000	50.800000	49.000000	0.000000	0.000000	0.000000	326
25%	0.400000	61.000000	56.000000	4.710000	4.710000	2.900000	945
50%	0.700000	61.800000	57.000000	5.690000	5.710000	3.520000	2375
75%	1.050000	62.500000	59.000000	6.550000	6.540000	4.040000	5360
max	4.500000	73.600000	79.000000	10.230000	58.900000	31.800000	18818

In [ ]:
```
## min of x,y and z seems zero, so there may be faulty diamonds(dimentionless did
```

In [24]:
```
df1.describe(include=['object']) ## statistics summary for object variable
```

Out[24]:

	cut	color	clarity
count	26967	26967	26967
unique	5	7	8
top	Ideal	G	SI1
freq	10816	5661	6571

In [25]:
```
print("Number of rows with x == 0: {} ".format((df1.x==0).sum()))
print("Number of rows with y == 0: {} ".format((df1.y==0).sum()))
print("Number of rows with z == 0: {} ".format((df1.z==0).sum()))
print("Number of rows with depth == 0: {} ".format((df1.depth==0).sum()))
## Check the values which are equal to zero.
```

```
Number of rows with x == 0: 3
Number of rows with y == 0: 3
Number of rows with z == 0: 9
Number of rows with depth == 0: 0
```

In [26]:
```
df1.shape
```

Out[26]: (26967, 10)

In [27]:
```
df1 = df1.drop(df1[df1["x"]==0].index)
df1 = df1.drop(df1[df1["y"]==0].index)
df1 = df1.drop(df1[df1["z"]==0].index)
df1.shape
#Dropping dimentionless diamonds
```

Out[27]:
```
(26958, 10)
```

EDA-Step-1: Checking for duplicate records in the data.

In [28]:
```
dups = df1.duplicated()
print('Number of duplicate rows = %d' % (dups.sum()))
print(df1.shape)
```

```
Number of duplicate rows = 33
(26958, 10)
```

In [29]:
```
print('Before',df1.shape)
df1.drop_duplicates(inplace=True)
print('After',df1.shape)
```

```
Before (26958, 10)
After (26925, 10)
```

In [30]:
```
dups = df1.duplicated()
print('Number of duplicate rows = %d' % (dups.sum()))
##checking duplicates again
```

```
Number of duplicate rows = 0
```

Step 2: Checking Missing value.

In [31]:
```
df1.isnull().sum()
```

Out[31]:
```
carat        0
cut          0
color        0
clarity      0
depth      697
table        0
x            0
y            0
z            0
price        0
dtype: int64
```

In [33]:
```
df1.interpolate()
## null value is replaced with average
```

Out[33]:

	carat	cut	color	clarity	depth	table	x	y	z	price
0	0.30	Ideal	E	SI1	62.1	58.0	4.27	4.29	2.66	499
1	0.33	Premium	G	IF	60.8	58.0	4.42	4.46	2.70	984
2	0.90	Very Good	E	VVS2	62.2	60.0	6.04	6.12	3.78	6289
3	0.42	Ideal	F	VS1	61.6	56.0	4.82	4.80	2.96	1082
4	0.31	Ideal	F	VVS1	60.4	59.0	4.35	4.43	2.65	779
...	...	...	...	...	...	...	...	...	...	...
26962	1.11	Premium	G	SI1	62.3	58.0	6.61	6.52	4.09	5408
26963	0.33	Ideal	H	IF	61.9	55.0	4.44	4.42	2.74	1114
26964	0.51	Premium	E	VS2	61.7	58.0	5.12	5.15	3.17	1656
26965	0.27	Very Good	F	VVS2	61.8	56.0	4.19	4.20	2.60	682
26966	1.25	Premium	J	SI1	62.0	58.0	6.90	6.88	4.27	5166

26925 rows × 10 columns

```
Step 3 : Outlier Checks.
```

In [37]:
```
cols = ['carat','depth', 'table', 'x', 'y', 'z',
        'price' ]
for i in cols:
    sns.boxplot(df1[i],whis=1.5)
    plt.show()
```

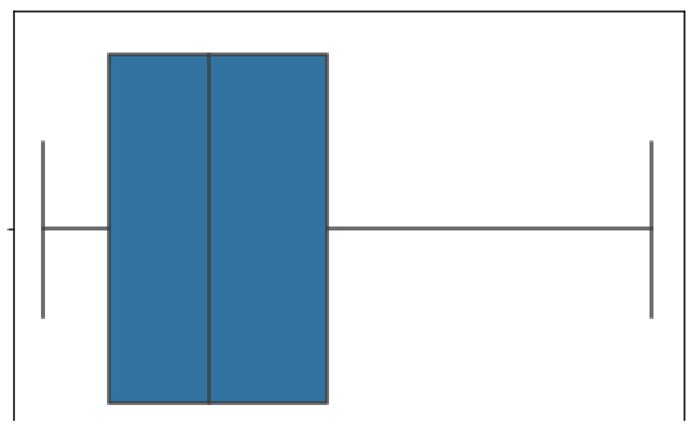
```
C:\Users\HP\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarn
ing: Pass the following variable as a keyword arg: x. From version 0.12, the
only valid positional argument will be `data`, and passing other arguments wi
thout an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

In [38]:
```
def remove_outlier(col):
    sorted(col)
    Q1,Q3=np.percentile(col,[25,75])
    IQR=Q3-Q1
    lower_range= Q1-(1.5 * IQR)
    upper_range= Q3+(1.5 * IQR)
    return lower_range, upper_range
### Outlier treatment :
```

In [42]:
```
for column in df1[cols].columns:
    lr,ur=remove_outlier(df1[column])
    df1[column]=np.where(df1[column]>ur,ur,df1[column])
    df1[column]=np.where(df1[column]<lr,lr,df1[column])
```

In [43]:
```
cols = ['carat','depth', 'table', 'x', 'y', 'z',
        'price' ]
for i in cols:
    sns.boxplot(df1[i],whis=1.5)
    plt.show()
```

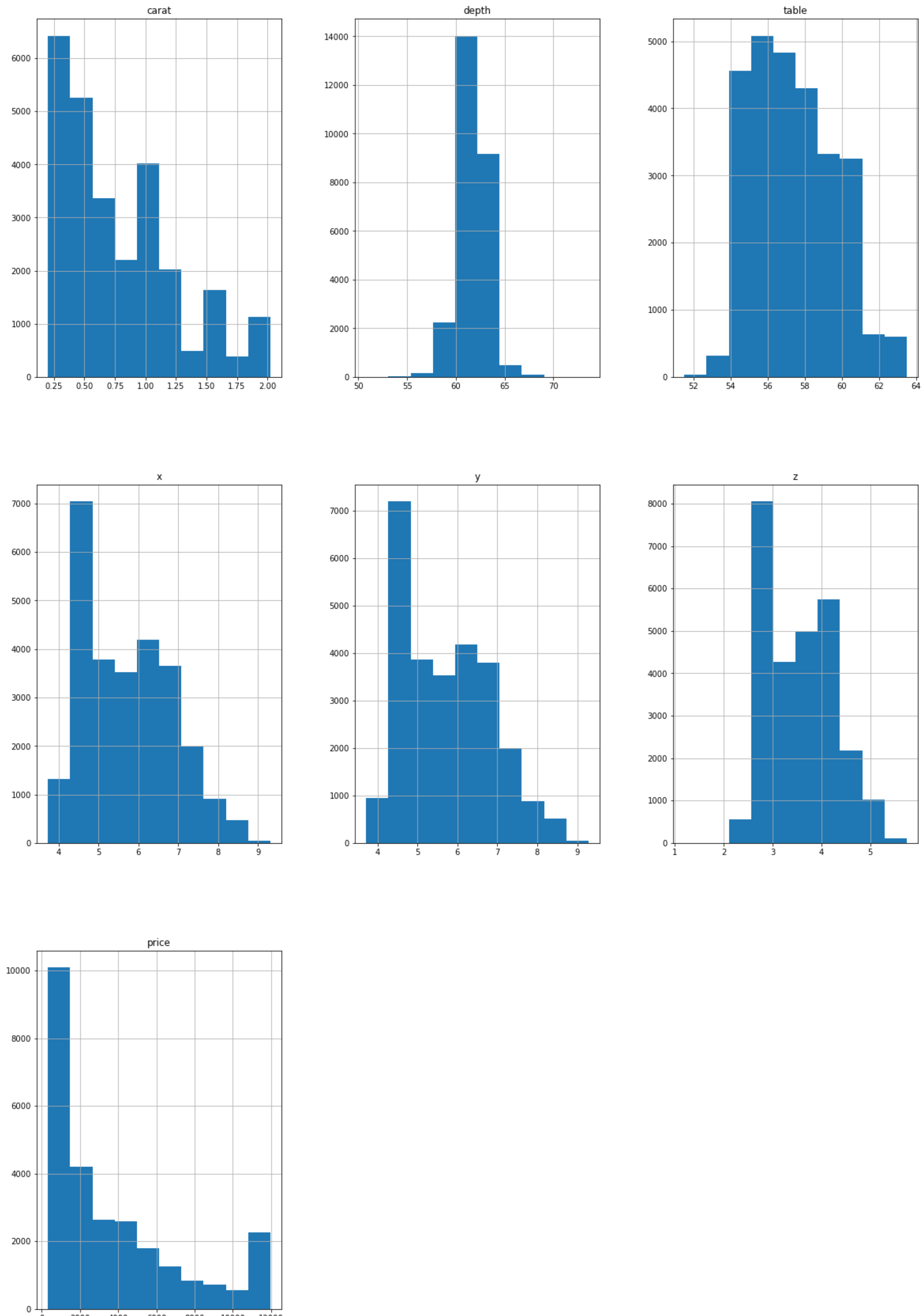
```
C:\Users\HP\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarn
ing: Pass the following variable as a keyword arg: x. From version 0.12, the
only valid positional argument will be `data`, and passing other arguments wi
thout an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



Step 4 : Univariate Analysis.

In [45]:
```
df1.hist(figsize=(20,30))
```

Out[45]:
```
array([[<AxesSubplot:title={'center':'carat'}>,
        <AxesSubplot:title={'center':'depth'}>,
        <AxesSubplot:title={'center':'table'}>],
       [<AxesSubplot:title={'center':'x'}>,
        <AxesSubplot:title={'center':'y'}>,
        <AxesSubplot:title={'center':'z'}>],
       [<AxesSubplot:title={'center':'price'}>, <AxesSubplot:>,
        <AxesSubplot:>]], dtype=object)
```


carat
depth
table
x
y
z
price

In [46]:
```
df1.skew()
```

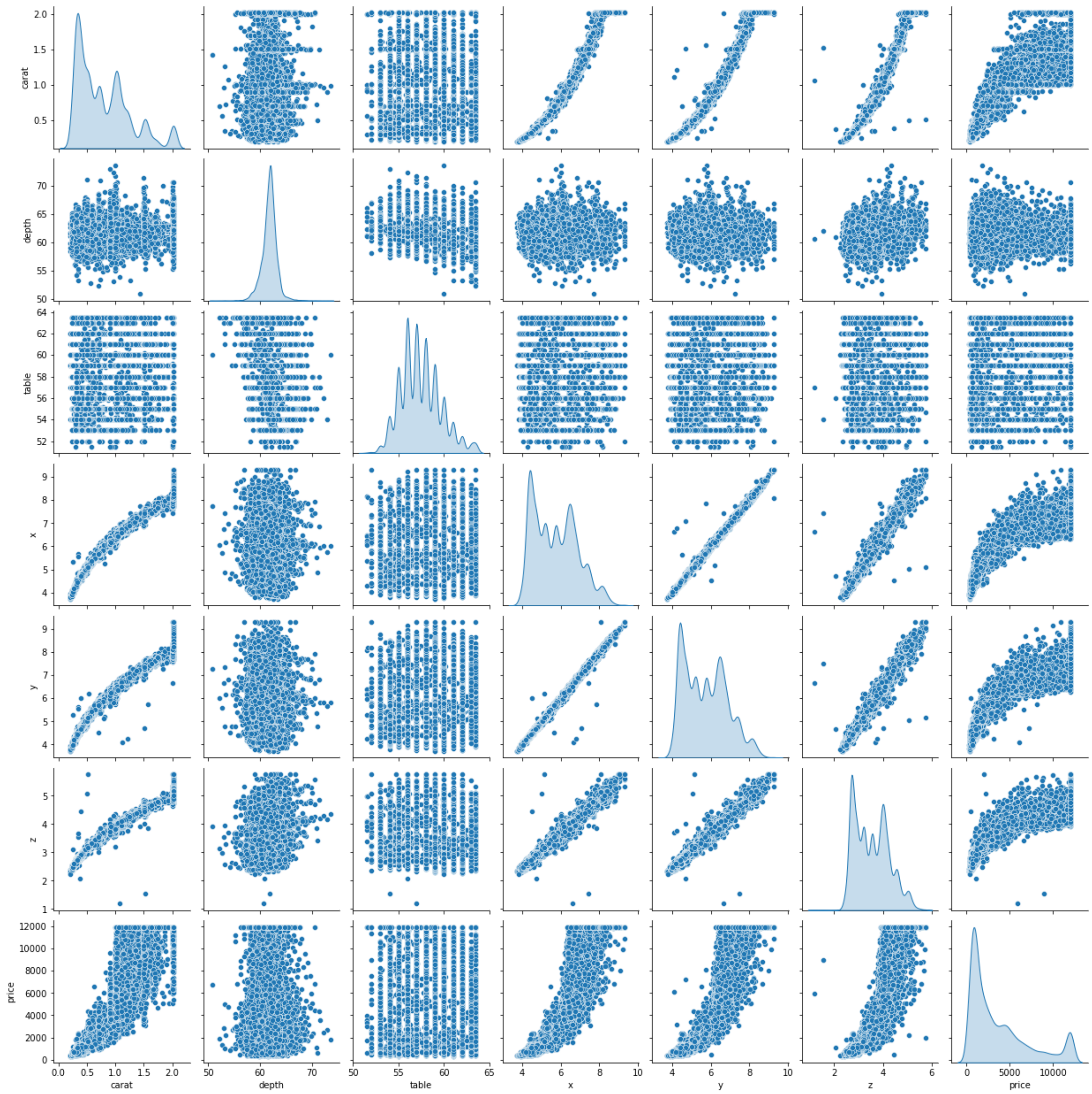
```
C:\Users\HP\AppData\Local\Temp/ipykernel_14164/3620588158.py:1: FutureWarning: 
Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None') 
is deprecated; in a future version this will raise TypeError.  Select only vali
d columns before calling the reduction.
  df1.skew()
```

Out[46]:
```
carat    0.917214
depth   -0.025042
table    0.480476
x        0.397696
y        0.394060
z        0.394819
price    1.157121
dtype: float64
```

We Observed that
There is significant amount of outlier present in some variable.
We can see that the distribution of some quantitative features like "carat" and the target feature "price" are heavily "right-skewed".

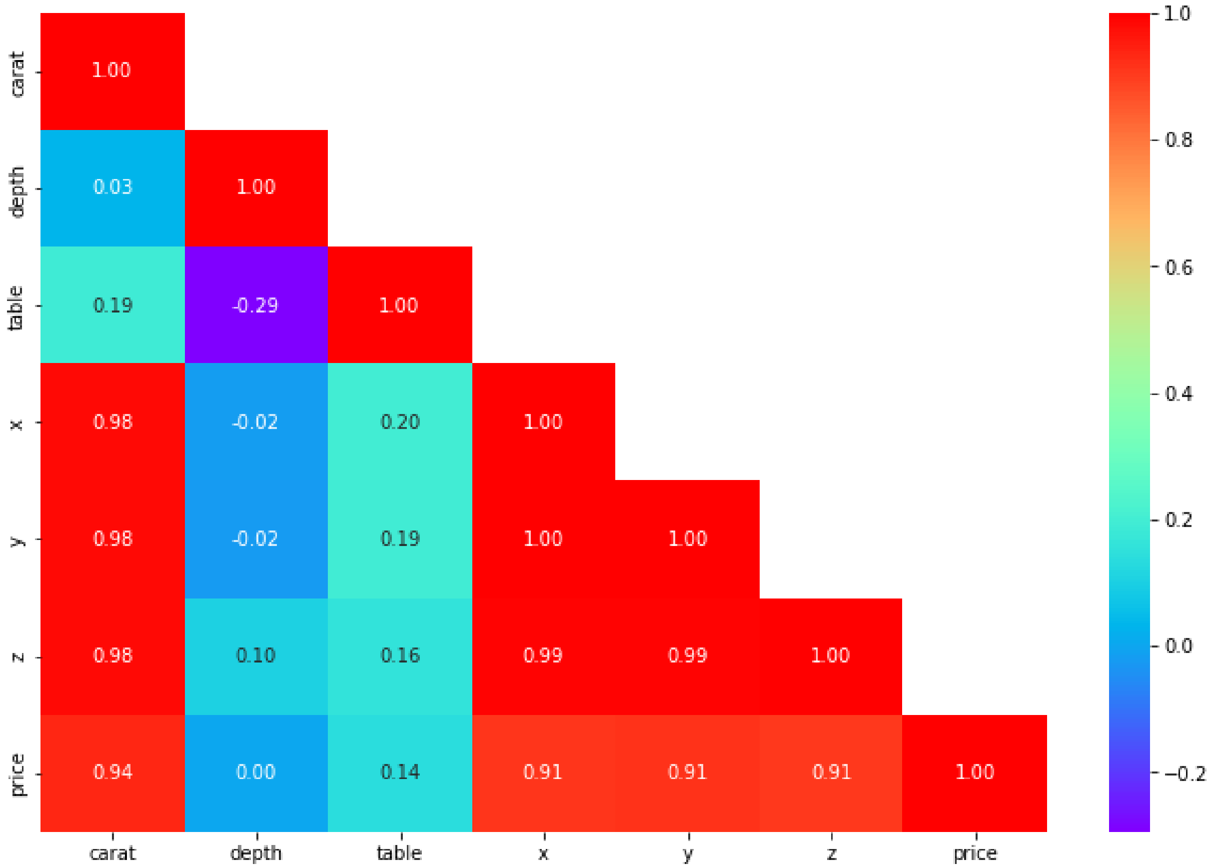
Step 5 : Bivariate Analysis.

In [48]:
```
df_attr = (df1[cols])
sns.pairplot(df_attr, diag_kind='kde')
plt.show()
```



Correlation Heatmap

In [49]:
```
plt.figure(figsize=(12,8))
sns.heatmap(df1.corr(),annot=True,fmt='.2f',cmap='rainbow',mask=np.triu(df1.corr(
plt.show()
```



In [50]:
```
correlations = df1.corr()
correlations["price"].sort_values(ascending=False)
```

Out[50]:
```
price    1.000000
carat    0.936765
y        0.914838
x        0.913409
z        0.908599
table    0.137915
depth    0.000313
Name: price, dtype: float64
```

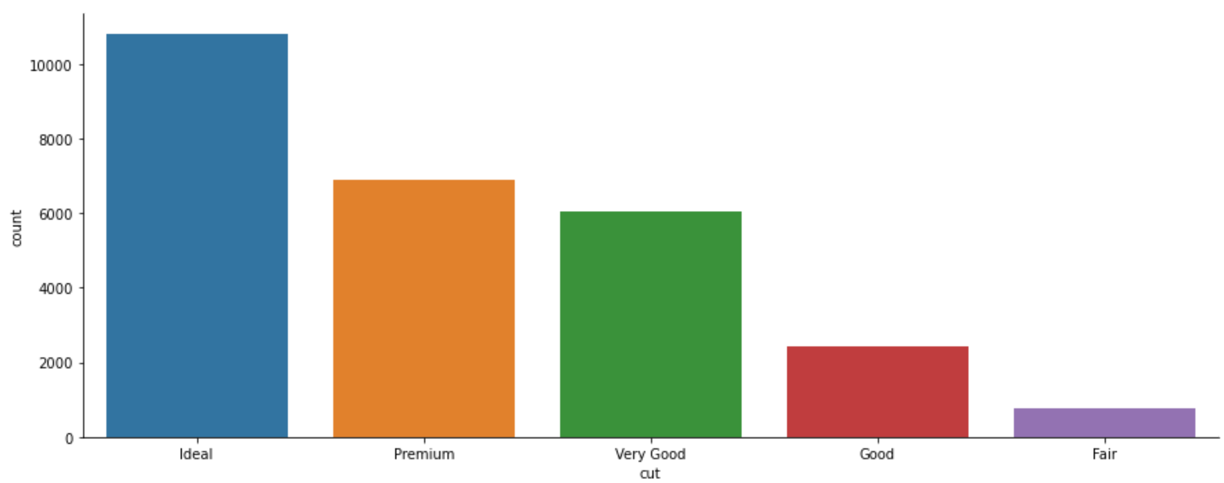
We observed that-
Most features correlate with the price of Diamond.
The notable exception is "depth" which has a negligible correlation (<1%)

EDA for Categorical variable

In [51]:
```
sns.catplot('cut', data=df1, kind='count',aspect=2.5)
```

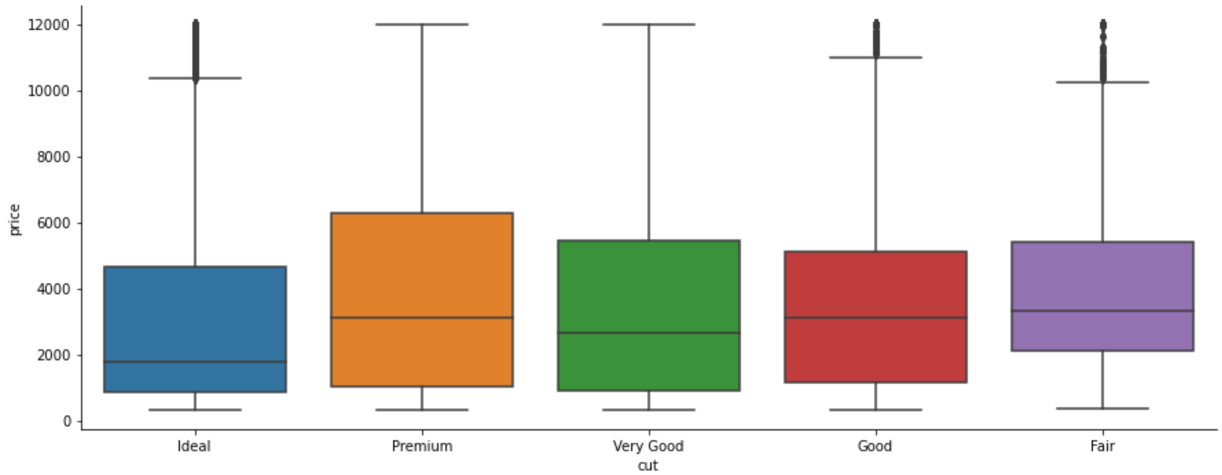
```
C:\Users\HP\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarnin
g: Pass the following variable as a keyword arg: x. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without a
n explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[51]:
```
<seaborn.axisgrid.FacetGrid at 0x239e22fdfd0>
```



In [52]:
```
sns.catplot(x='cut', y='price', kind='box', data=df1, aspect=2.5)
```

Out[52]: <seaborn.axisgrid.FacetGrid at 0x239e21bc9d0>

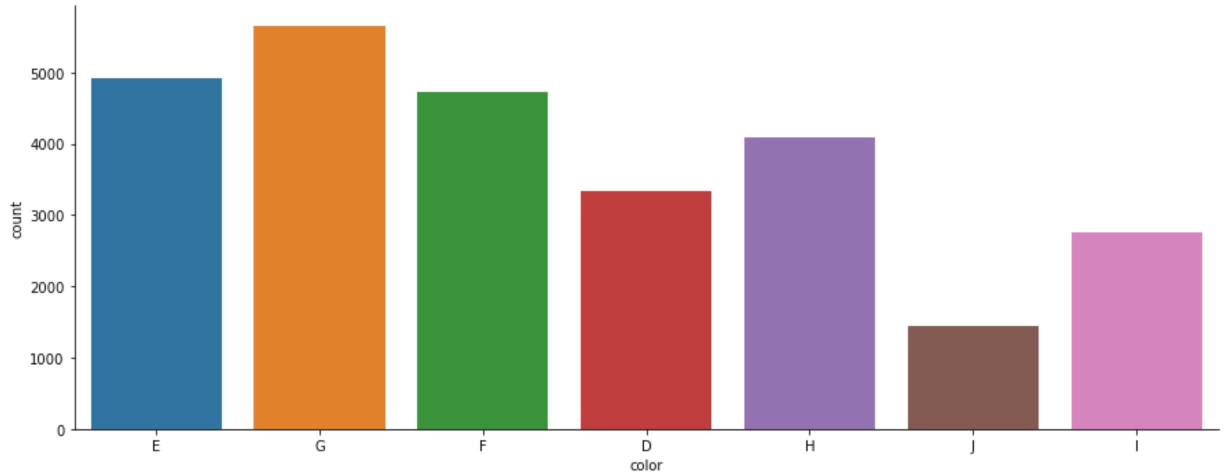



```
We observed that
The Premium Cut on Diamonds are the most Expensive, followed by Very Good Cut.
```

In [54]:
```
sns.catplot('color', kind='count', data=df1, aspect=2.5)
# EDA for categorical columns 'Color'.
```

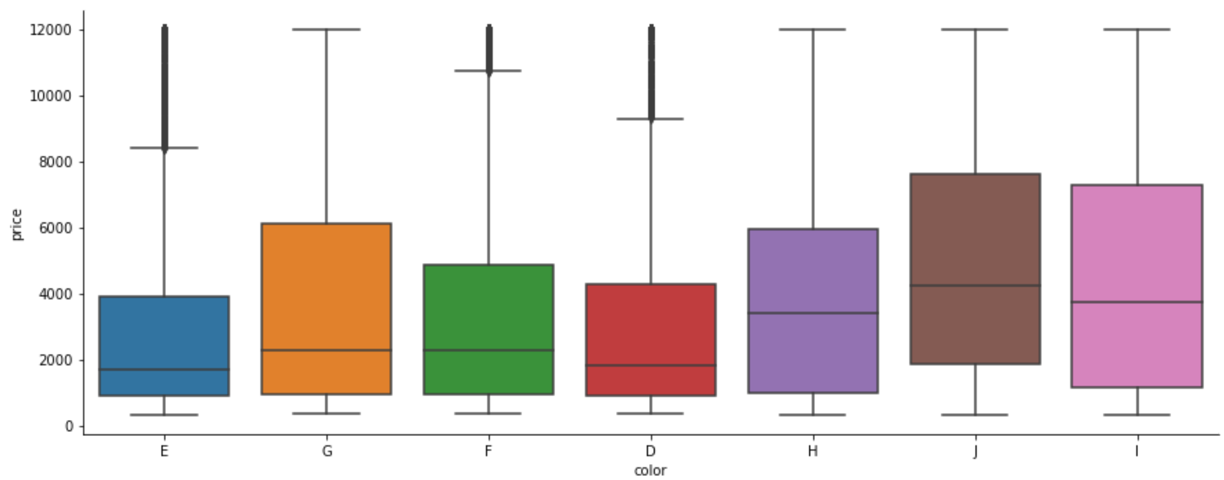
```
C:\Users\HP\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarnin
g: Pass the following variable as a keyword arg: x. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without a
n explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[54]: <seaborn.axisgrid.FacetGrid at 0x239dd4059a0>



In [55]:
```
sns.catplot(x='color', y='price', data=df1, aspect =2.5, kind='box')
```

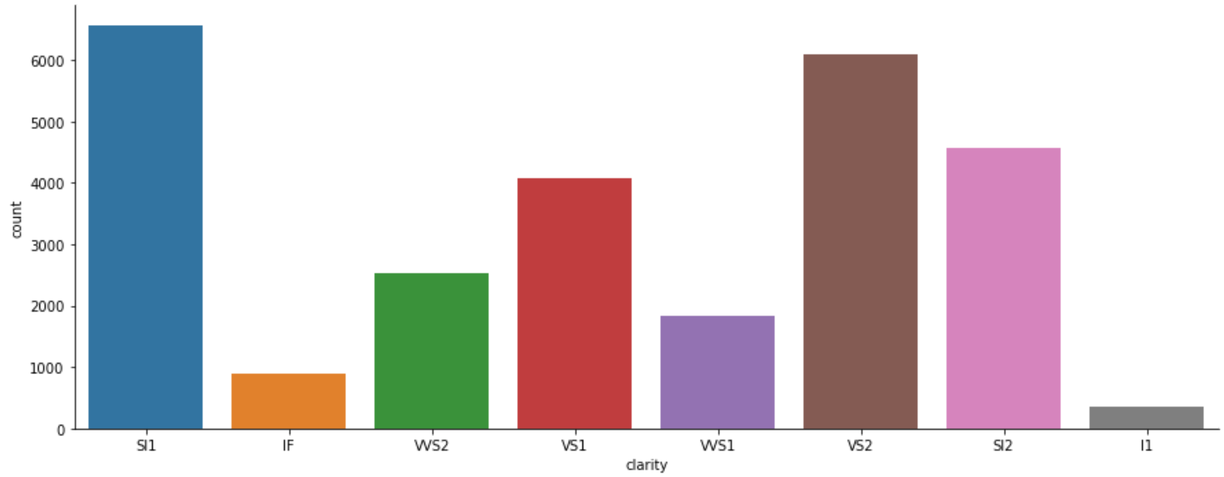
Out[55]: <seaborn.axisgrid.FacetGrid at 0x239e4ac24c0>



In [57]:
```
sns.catplot('clarity', data=df1, kind='count',aspect=2.5)
# EDA for categorical columns 'Clarity'.
```

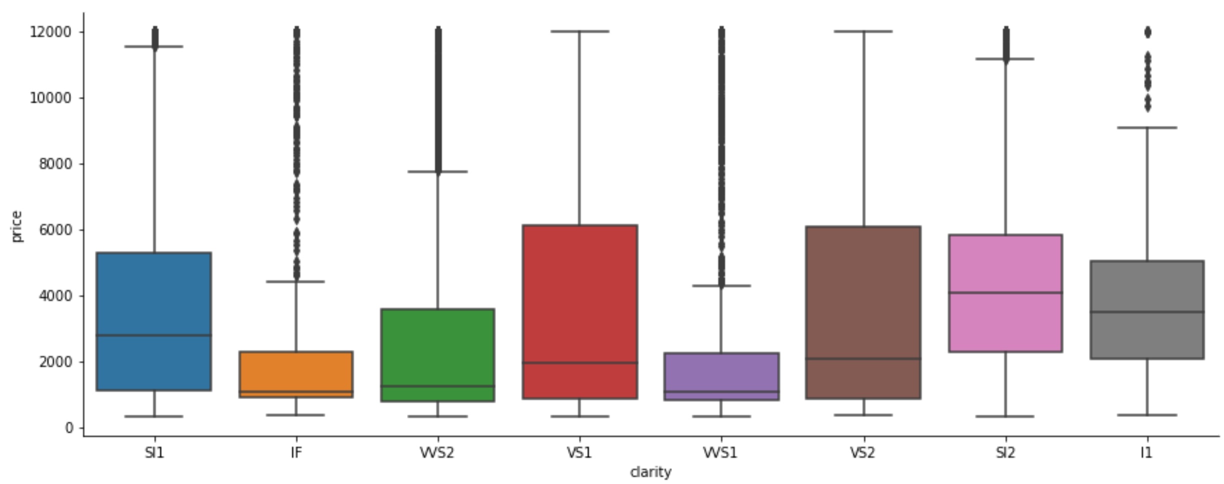
```
C:\Users\HP\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarnin
g: Pass the following variable as a keyword arg: x. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without a
n explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[57]: <seaborn.axisgrid.FacetGrid at 0x239e22f1550>



In [58]:
```
sns.catplot(x='clarity', y='price', data=df1, aspect =2.5, kind='box')
```

Out[58]: <seaborn.axisgrid.FacetGrid at 0x239dd3fc040>



We observed that
The Diamonds clarity with VS1 & VS2 are the most Expensive

The inferences drawn from the above Exploratory Data analysis:
Observation-1:
(1).'Price' is the target variable while all others are the predictors. (2).The data set contains 26967 row, 11 column. (3).In the given data set there are 2 Integer type features,6 Float type features. 3 Object type features. Where 'price' is the target variable and all other are predector variable. (4)The first column is an index ("Unnamed: 0")as this only serial no, we can remove it.

Observation-2:
(1).On the given data set the the mean and median values does not have much differenc. (2).We can observe Min value of "x", "y", "z" are zero this indicates that they are faulty values. As we know dimensionless or 2-dimensional diamonds are not possible. So we have filter out those as it clearly faulty data entries. (3).There are three object data type 'cut', 'color' and 'clarity'.

Observation-3:
we can observe there are 697 missing value in the depth column. There are some duplicate row present. (33 duplicate rows out of 26958). which is nearly 0.12 % of the total data. So on this case we have dropped the duplicated row.

Observation-4: :
There are significant amount of outlier present in some variable,the features with datapoint that are far from the rest of dataset which will affect the outcome of our regression model. So we have treat the outliar. We can see that the distribution of some quantitative features like "carat" and the target feature "price" are heavily "right-skewed".

Observation-5:
It looks like most features do correlate with the price of Diamond. The notable exception is "depth" which has a negligible correlation (~1%). Observation on 'CUT': The Premium Cut on Diamonds are the most Expensive, followed by Very Good Cut.

In [59]:
```
df1.shape
```

Out[59]:
```
(26925, 10)
```

We need to perform label encoding for categorocal values

In [62]:
```
print('cut\n',df1.cut.value_counts())
print('\n')
print('color\n',df1.color.value_counts())
print('\n')
print('clarity\n',df1.clarity.value_counts())
print('\n')
```

```
cut
 Ideal        10805
Premium       6880
Very Good     6027
Good          2434
Fair           779
Name: cut, dtype: int64


color
 G    5650
E    4916
F    4722
H    4091
D    3341
I    2765
J    1440
Name: color, dtype: int64


clarity
 SI1     6564
VS2     6092
SI2     4561
VS1     4086
VVS2    2530
VVS1    1839
IF       891
I1       362
Name: clarity, dtype: int64
```

In [61]:
```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
```

In [63]:
```
df1['cut']=le.fit_transform(df1['cut'])
df1['color']=le.fit_transform(df1['color'])
df1['clarity']=le.fit_transform(df1['clarity'])
df1
```

Out[63]:

	carat	cut	color	clarity	depth	table	x	y	z	price
0	0.30	2	1	2	62.1	58.0	4.27	4.29	2.66	499.0
1	0.33	3	3	1	60.8	58.0	4.42	4.46	2.70	984.0
2	0.90	4	1	7	62.2	60.0	6.04	6.12	3.78	6289.0
3	0.42	2	2	4	61.6	56.0	4.82	4.80	2.96	1082.0
4	0.31	2	2	6	60.4	59.0	4.35	4.43	2.65	779.0
...	...	...	...	...	...	...	...	...	...	...
26962	1.11	3	3	2	62.3	58.0	6.61	6.52	4.09	5408.0
26963	0.33	2	4	1	61.9	55.0	4.44	4.42	2.74	1114.0
26964	0.51	3	1	5	61.7	58.0	5.12	5.15	3.17	1656.0
26965	0.27	4	2	7	61.8	56.0	4.19	4.20	2.60	682.0
26966	1.25	3	6	2	62.0	58.0	6.90	6.88	4.27	5166.0

26925 rows × 10 columns

In [64]:
```
df1.dtypes
```

Out[64]:
```
carat       float64
cut           int32
color         int32
clarity       int32
depth       float64
table       float64
x           float64
y           float64
z           float64
price       float64
dtype: object
```

In [65]:
```
df1.head(10)
```

Out[65]:

	carat	cut	color	clarity	depth	table	x	y	z	price
0	0.30	2	1	2	62.1	58.0	4.27	4.29	2.66	499.0
1	0.33	3	3	1	60.8	58.0	4.42	4.46	2.70	984.0
2	0.90	4	1	7	62.2	60.0	6.04	6.12	3.78	6289.0
3	0.42	2	2	4	61.6	56.0	4.82	4.80	2.96	1082.0
4	0.31	2	2	6	60.4	59.0	4.35	4.43	2.65	779.0
5	1.02	2	0	5	61.5	56.0	6.46	6.49	3.99	9502.0
6	1.01	1	4	2	63.7	60.0	6.35	6.30	4.03	4836.0
7	0.50	3	1	2	61.5	62.0	5.09	5.06	3.12	1415.0
8	1.21	1	4	2	63.8	63.5	6.72	6.63	4.26	5407.0
9	0.35	2	2	5	60.5	57.0	4.52	4.60	2.76	706.0

Performing train_test_split

In [67]:
```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics
```

In [68]:
```
X = df1.drop('price', axis=1)
# Copy all the predictor variables into X dataframe

y = df1[['price']]
# Copy target into the y dataframe.This is the dependent variable
X.head()
```

Out[68]:

	carat	cut	color	clarity	depth	table	x	y	z
0	0.30	2	1	2	62.1	58.0	4.27	4.29	2.66
1	0.33	3	3	1	60.8	58.0	4.42	4.46	2.70
2	0.90	4	1	7	62.2	60.0	6.04	6.12	3.78
3	0.42	2	2	4	61.6	56.0	4.82	4.80	2.96
4	0.31	2	2	6	60.4	59.0	4.35	4.43	2.65

In [126]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30 , randor
# Split X and y into training and test set in 70:30 ratio
```

In [132]:
```
df1=df1.fillna(df1.median())
df1.isnull().sum()
```

Out[132]:
```
carat      0
cut        0
color      0
clarity    0
depth      0
table      0
x          0
y          0
z          0
price      0
dtype: int64
```

In [135]:
```
X_train.isnull().sum()
```

Out[135]:
```
carat        0
cut          0
color        0
clarity      0
depth      465
table        0
x            0
y            0
z            0
dtype: int64
```

In [136]:
```
X_train=X_train.fillna(X_train.median())
X_train.isnull().sum()
```

Out[136]:
```
carat      0
cut        0
color      0
clarity    0
depth      0
table      0
x          0
y          0
z          0
dtype: int64
```

In [138]:
```
regression_model = LinearRegression()
regression_model.fit(X_train, y_train)
```

Out[138]:
```
LinearRegression()
```

In [140]:
```
# Let us explore the coefficients for each of the independent attributes

for idx, col_name in enumerate(X_train.columns):
    print("The coefficient for {} is {}".format(col_name, regression_model.coef_[
```

```
The coefficient for carat is 9187.007840783974
The coefficient for cut is 45.65932310314497
The coefficient for color is -230.0995725482948
The coefficient for clarity is 253.65986090015744
The coefficient for depth is -45.46461293128727
The coefficient for table is -73.75931236519406
The coefficient for x is -1776.5656497683483
The coefficient for y is 1690.1037027767075
The coefficient for z is -939.3767511873145
```

Observation-1:
Y=mx +c (m= m1,m2,m3...m9) here 9 diferent co-efficients will learn aling with the intercept which is "c" from the model.

From the above coefficients for each of the independent attributes we can conclude
The one unit increase in carat increases price by 9187.007.
The one unit increase in cut increases price by 45.659.
The one unit increase in clarity increases price by 253.659.
The one unit increase in y increases price by 1690.103.

But
The one unit increase in color decreases price by -230.099.
The one unit increase in depth decreases price by -45.464,
The one unit increase in table decreases price by -73.759,
The one unit increase in x decreases price by-1776.565,
The one unit increase in z decreases price by -939.376.

In [141]:
```
# Let us check the intercept for the model

intercept = regression_model.intercept_[0]

print("The intercept for our model is {}".format(intercept))
```

```
The intercept for our model is 6827.621257693231
```

In [143]:
```
regression_model.score(X_train, y_train)
 # R square on training data
```

Out[143]:
```
0.9107404234406657
```

In [145]:
```
X_test.isnull().sum()
```

Out[145]:
```
carat         0
cut           0
color         0
clarity       0
depth       232
table         0
x             0
y             0
z             0
dtype: int64
```

In [146]:
```
X_test=X_test.fillna(X_test.median())
X_test.isnull().sum()
```

Out[146]:
```
carat      0
cut        0
color      0
clarity    0
depth      0
table      0
x          0
y          0
z          0
dtype: int64
```

In [147]:
```
regression_model.score(X_test, y_test)
# R square on testing data

# Model score - R2 or coeff of determinant
# R^2=1-RSS / TSS =  RegErr / TSS
```

Out[147]:
```
0.9078673801695107
```

Observation: R-square is the percentage of the response variable variation that is explained by a linear model. Or:

R-square = Explained variation / Total variation

R-squared is always between 0 and 100%: 0% indicates that the model explains none of the variability of the response data around its mean.100% indicates that the model explains all the variability of the response data around its mean. In this regression model we can see the R-square value on Training and Test data respectively 0.9107404234406657 and 0.9078673801695107.

In [151]:
```
predicted_train=regression_model.fit(X_train, y_train).predict(X_train)
np.sqrt(metrics.mean_squared_error(y_train,predicted_train))
#RMSE on Training data
```

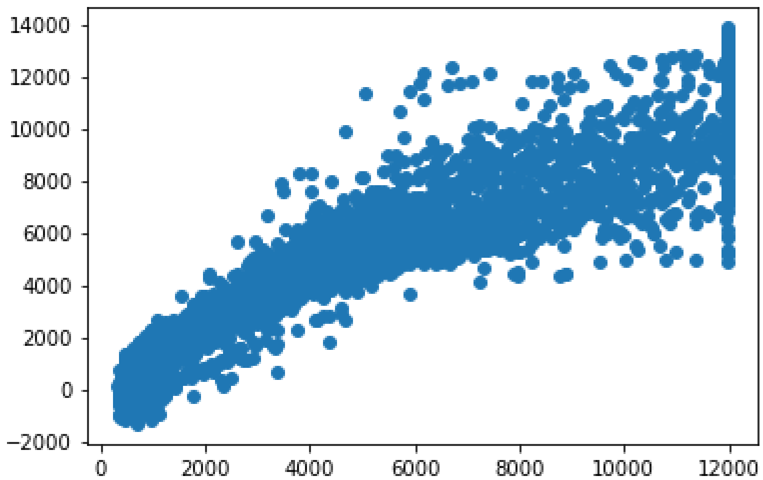
Out[151]:
```
1041.8667819923364
```

In [152]:
```
predicted_test=regression_model.fit(X_train, y_train).predict(X_test)
np.sqrt(metrics.mean_squared_error(y_test,predicted_test))
#RMSE on Testing data
```

Out[152]: 1036.9280810144319

In [153]:
```
# Since this is regression, plot the predicted y value vs actual y values for the
# A good model's prediction will be close to actual leading to high R and R2 valu
y_pred = regression_model.predict(X_test)
plt.scatter(y_test['price'], y_pred)
```

Out[153]: <matplotlib.collections.PathCollection at 0x239def2fdc0>



Observation: we can see that the is a linear plot, very strong corelation between the predicted y and actual y. But there are lots of spread. That indicated some kind noise present on the data set i.e Unexplained variances on the output.

Linear regression Performance Metrics:

intercept for the model: 6827.621257693231 R square on training data: 0.9107404234406657 R square on testing data: 0.9078673801695107 RMSE on Training data: 1041.8667819923364 RMSE on Testing data: 1036.9280810144319 As the training data & testing data score are almost inline, we can conclude this model is a Right-Fit Model.

In [ ]: