DIAMOND DATASET
DATA ANALYSIS

Problem-2
Aditya Kumar Tiwari, Arqam Patel, Kumar Kanishk Singh

Link:

https://www.kaggle.com/datasets/nancyalaswad90/diamonds-prices

The given diamond dataset provides us with 9 different characteristics. of a diamond and the corresponding value(Price) of the diamond.
Main underlying problem of interest for this dataset is to find the relation of different characteristics of the diamond to the price as well as the relations among themselves.
We will also visualise these relations to get more clarity about the effect of different characteristics on the price of diamond.

CODE FOR ANALYSING THE GIVEN DATASET

```
df = pd.read_csv("C:\\Users\\91993\\Desktop\\IITK ACADS\\SEM6\\MTH211A\\Diamonds Dataset\\
df.head()
```

	Unnamed: 0	carat	cut	color	clarity	depth	table	price	x	y	z
0	1	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	2	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	3	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	4	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	5	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

Removing not useful columns first

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

	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

```python
df = df.drop_duplicates()
```

```python
df.shape
```

```
(53794, 10)
```

```python
print(df.info())
```

```
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   carat    53794 non-null  float64
 1   cut      53794 non-null  object 
 2   color    53794 non-null  object 
 3   clarity  53794 non-null  object 
 4   depth    53794 non-null  float64
 5   table    53794 non-null  float64
 6   price    53794 non-null  int64  
 7   x        53794 non-null  float64
 8   y        53794 non-null  float64
 9   z        53794 non-null  float64
dtypes: float64(6), int64(1), object(3)
memory usage: 4.5+ MB
None
```

It is clear that there are None Null-Value cells in any of the columns.

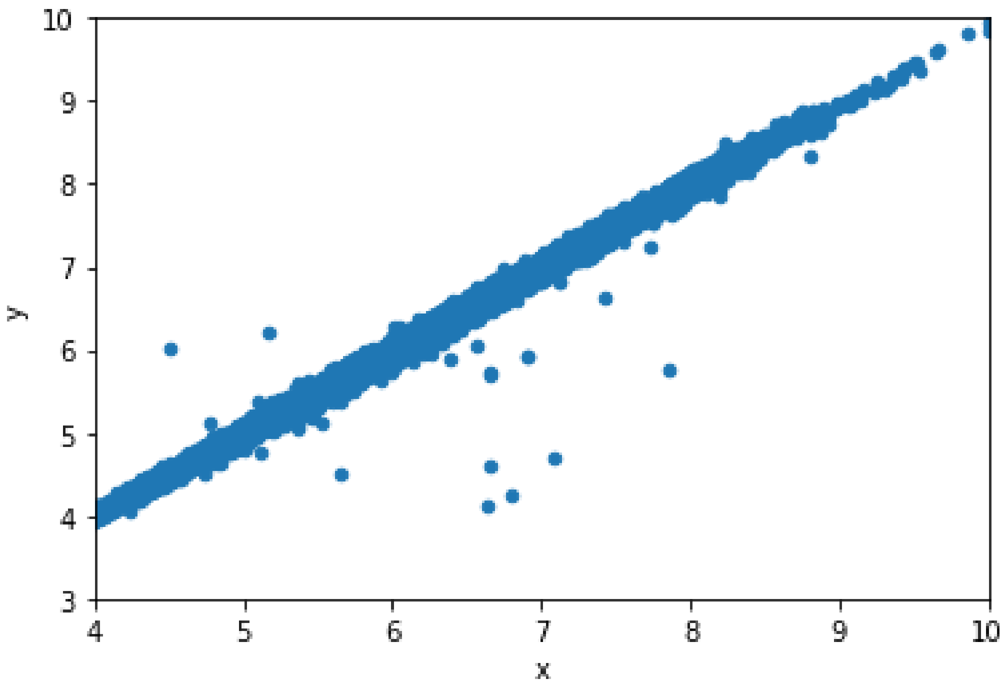
```
df.describe()
```

	carat	depth	table	price	x	y	z
count	53794.00000	53794.000000	53794.000000	53794.000000	53794.000000	53794.000000	53794.000
mean	0.79778	61.748080	57.458109	3933.065082	5.731214	5.734653	3.538714
std	0.47339	1.429909	2.233679	3988.114460	1.120695	1.141209	0.705037
min	0.20000	43.000000	43.000000	326.000000	0.000000	0.000000	0.000000
25%	0.40000	61.000000	56.000000	951.000000	4.710000	4.720000	2.910000
50%	0.70000	61.800000	57.000000	2401.000000	5.700000	5.710000	3.530000
75%	1.04000	62.500000	59.000000	5326.750000	6.540000	6.540000	4.030000
max	5.01000	79.000000	95.000000	18823.000000	10.740000	58.900000	31.800000

Plotting the x-y, y-z, z-x plots to check relation between different dimensions of the diamond

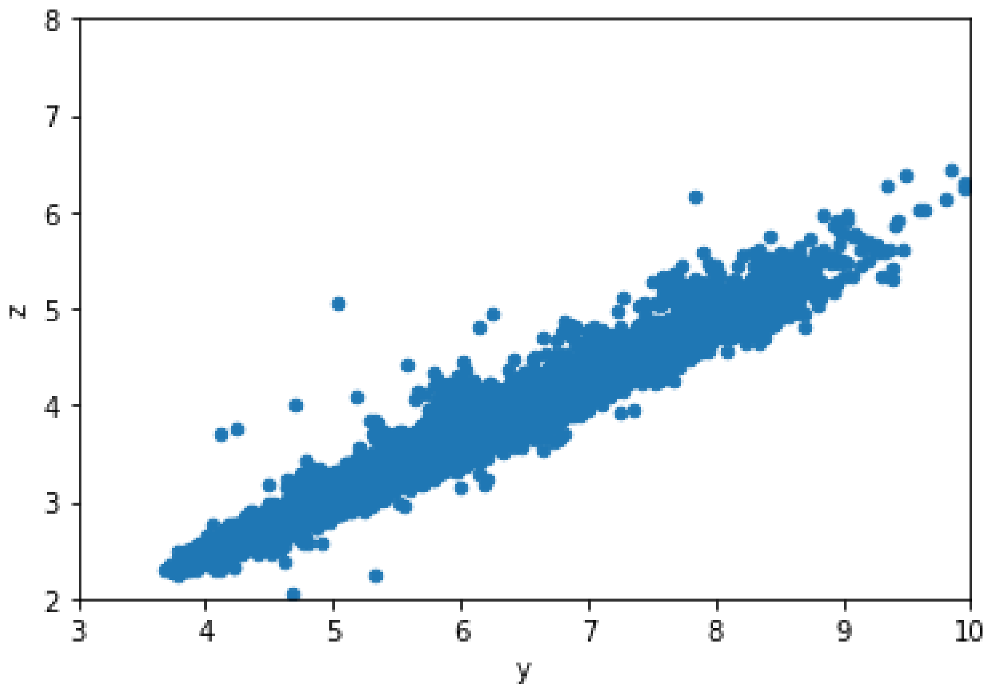
```
df.plot.scatter(x='x', y='y')
plt.xlim([4, 10])
plt.ylim([3,10])
```

```
(3.0, 10.0)
```



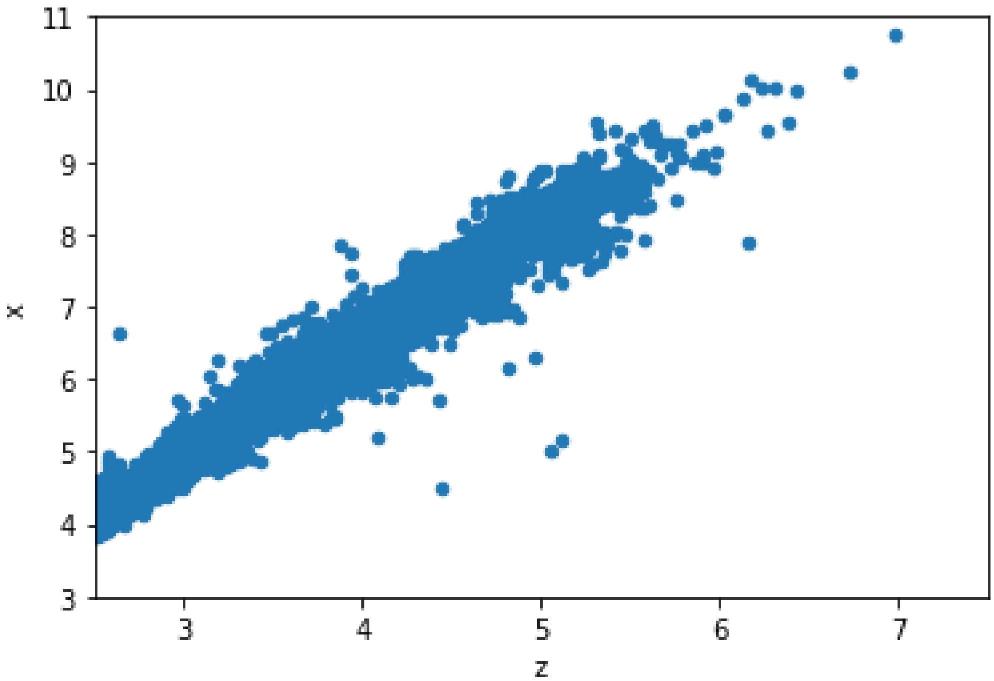

```
df.plot.scatter(x='y', y='z')
plt.xlim([3, 10])
plt.ylim([2,8])
```

```
(2.0, 8.0)
```




```
df.plot.scatter(x='z', y='x')
plt.xlim([2.5, 7.5])
plt.ylim([3,11])
```

```
(3.0, 11.0)
```



It is quite interesting to see that when we do not scale the graphs to predominant region, we can see a lot of zero values for x, y, and z. This isn't possible as Diamond is a 3D object.
Thus, we can eliminate all the values having x, y, or z as 0.

```
df[(df['x']==0) | (df['y']==0) | (df['z']==0)]
```

	carat	cut	color	clarity	depth	table	price	x	y	z
2207	1.00	Premium	G	SI2	59.1	59.0	3142	6.55	6.48	0.0
2314	1.01	Premium	H	I1	58.1	59.0	3167	6.66	6.60	0.0
4791	1.10	Premium	G	SI2	63.0	59.0	3696	6.50	6.47	0.0
5471	1.01	Premium	F	SI2	59.2	58.0	3837	6.50	6.47	0.0
10167	1.50	Good	G	I1	64.0	61.0	4731	7.15	7.04	0.0
11182	1.07	Ideal	F	SI2	61.6	56.0	4954	0.00	6.62	0.0
11963	1.00	Very Good	H	VS2	63.3	53.0	5139	0.00	0.00	0.0
13601	1.15	Ideal	G	VS2	59.2	56.0	5564	6.88	6.83	0.0
15951	1.14	Fair	G	VS1	57.5	67.0	6381	0.00	0.00	0.0
24394	2.18	Premium	H	SI2	59.4	61.0	12631	8.49	8.45	0.0
24520	1.56	Ideal	G	VS2	62.2	54.0	12800	0.00	0.00	0.0
26123	2.25	Premium	I	SI1	61.3	58.0	15397	8.52	8.42	0.0

	carat	cut	color	clarity	depth	table	price	x	y	z
26243	1.20	Premium	D	VVS1	62.1	59.0	15686	0.00	0.00	0.0
27112	2.20	Premium	H	SI1	61.2	59.0	17265	8.42	8.37	0.0
27429	2.25	Premium	H	SI2	62.8	59.0	18034	0.00	0.00	0.0
27503	2.02	Premium	H	VS2	62.7	53.0	18207	8.02	7.95	0.0
27739	2.80	Good	G	SI2	63.8	58.0	18788	8.90	8.85	0.0
49556	0.71	Good	F	SI2	64.1	60.0	2130	0.00	0.00	0.0
51506	1.12	Premium	G	I1	60.4	59.0	2383	6.71	6.67	0.0

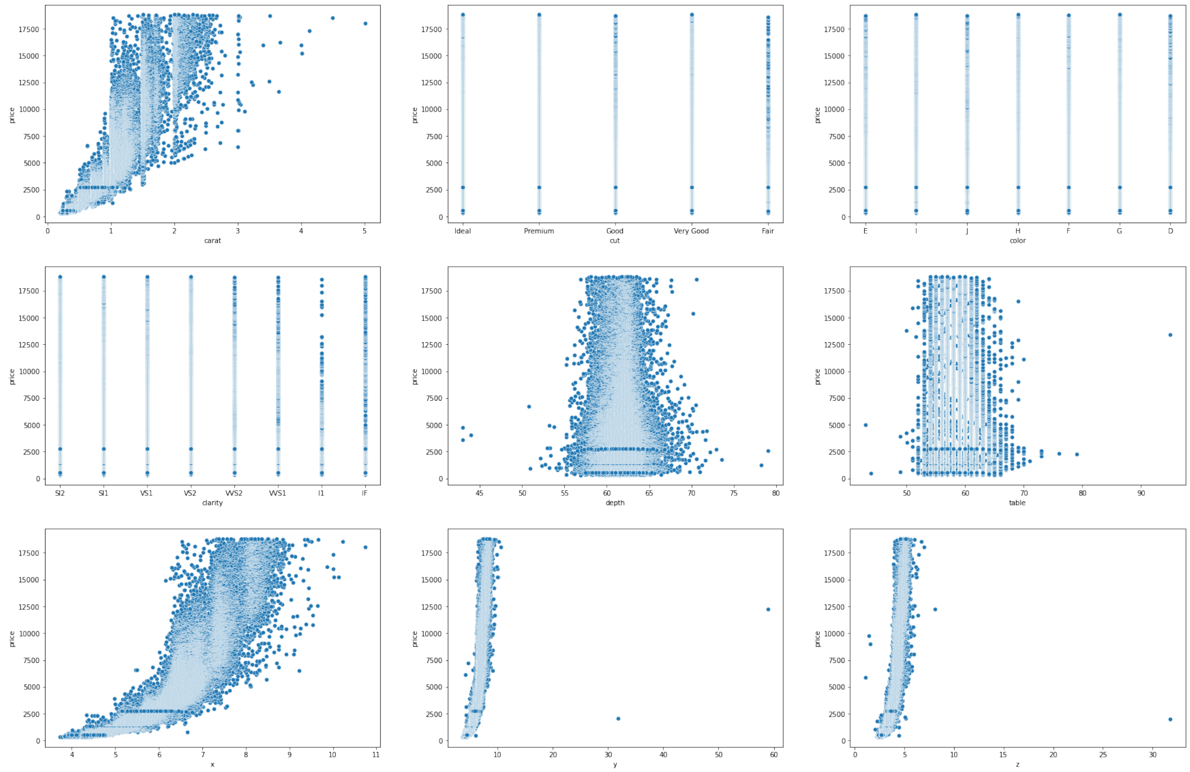
```
df[['x','y','z']] = df[['x','y','z']].replace(0, np.nan)
df.dropna(inplace=True, axis=0)
```

```
df[(df['x']==0) | (df['y']==0) | (df['z']==0)]
```

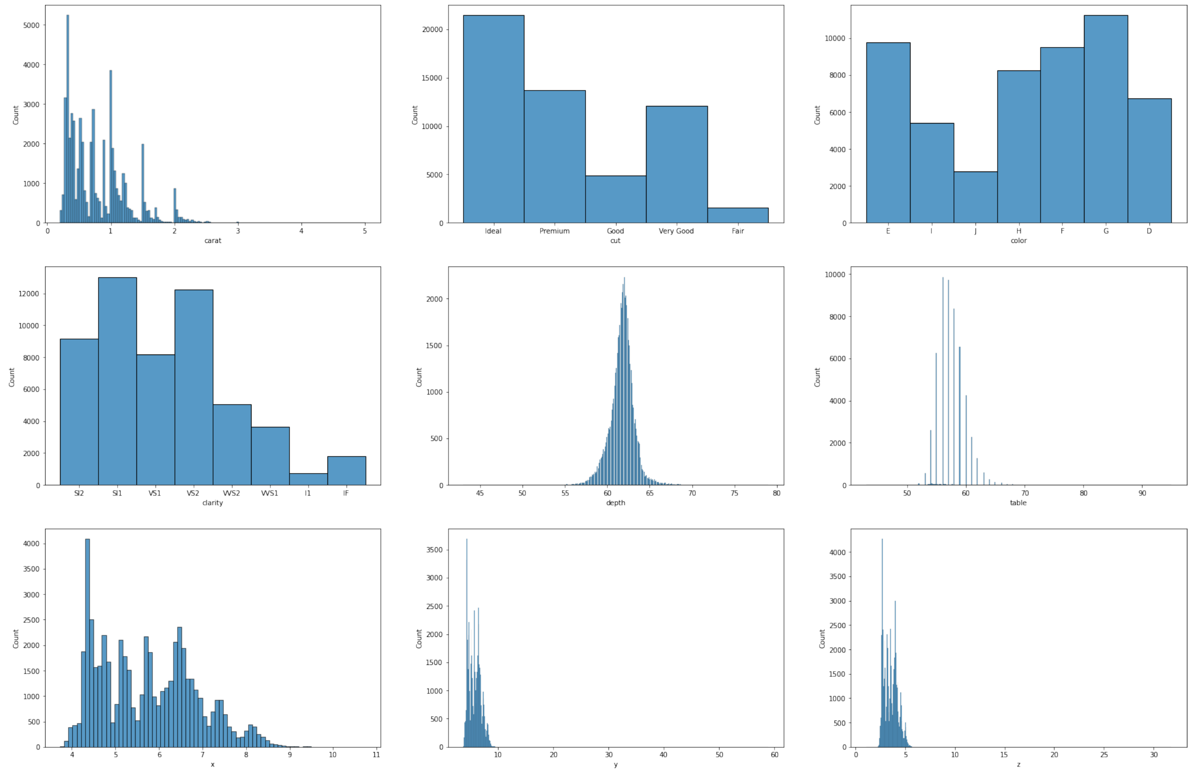
carat	cut	color	clarity	depth	table	price	x	y	z

Plotting different graphs to get more insights over the data.

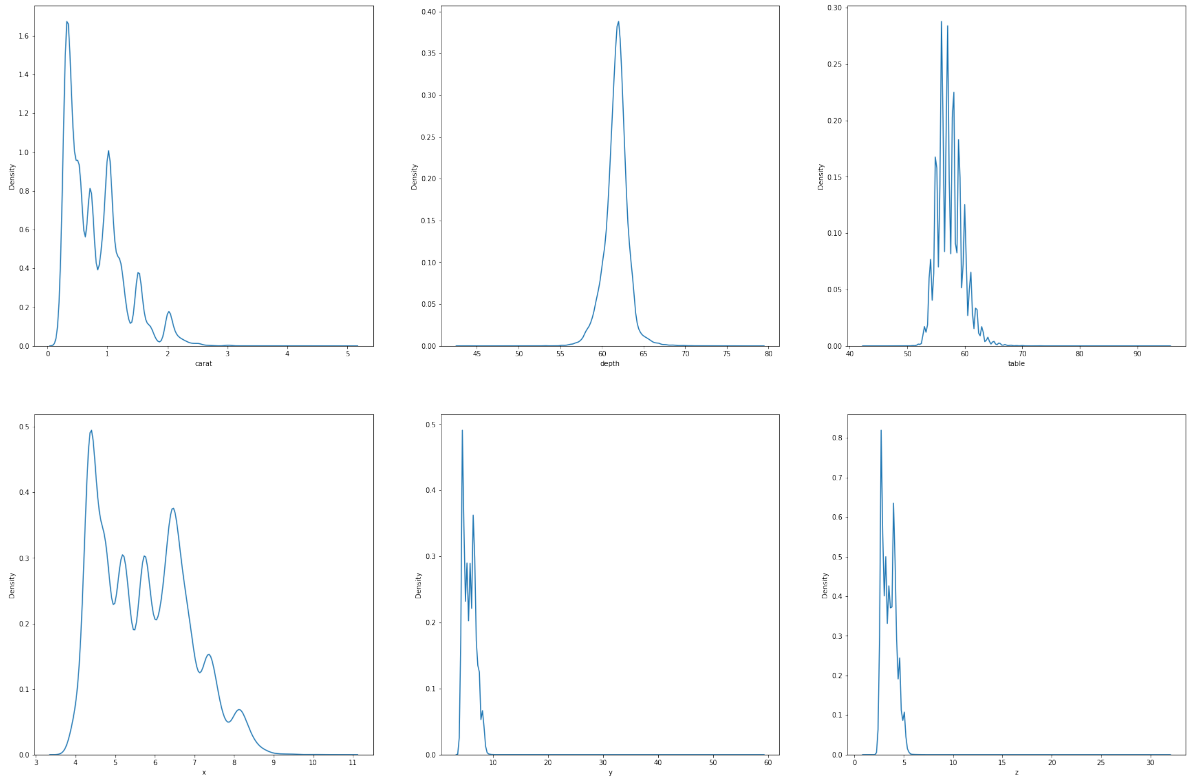
```
<AxesSubplot:xlabel='z', ylabel='price'>
```




```
<AxesSubplot:xlabel='z', ylabel='Count'>
```



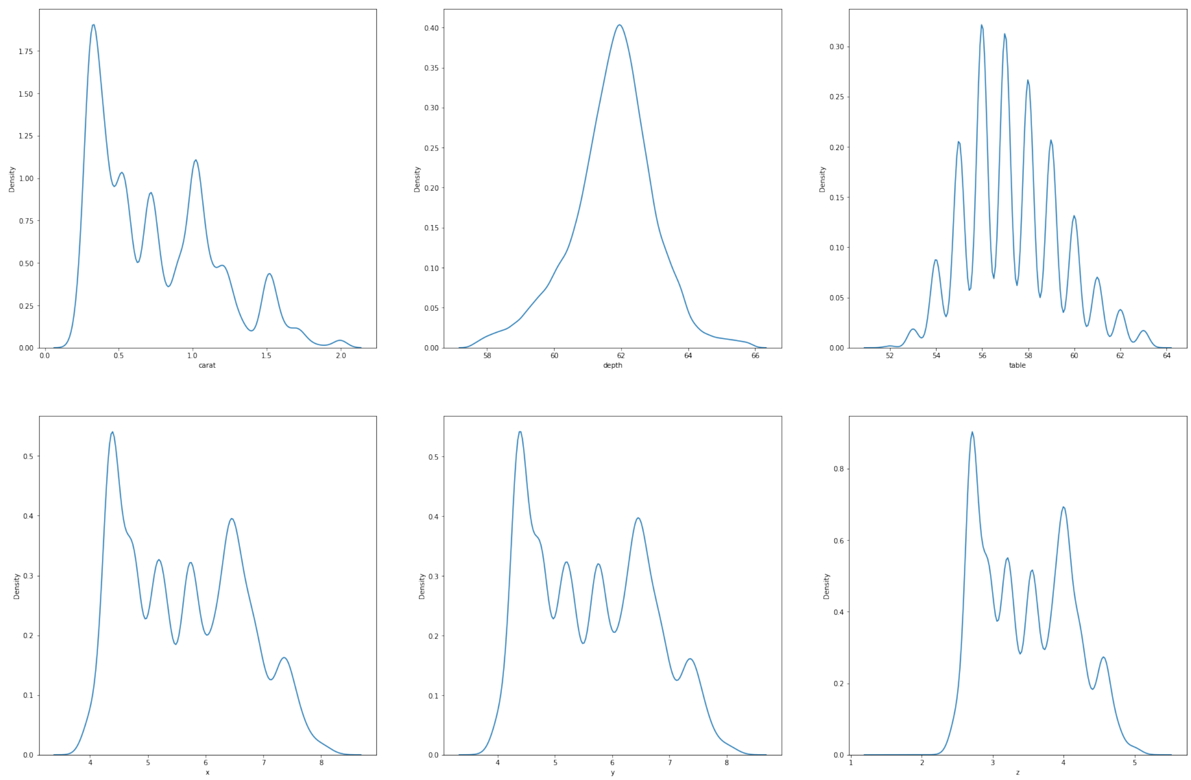

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



From the KDE graphs it can be observed that except 'Depth' all the other graphs are skewed. Thus we can use 'SD' type outlier removal for 'Depth' and 'Quantile' type outlier removal for the other properties.

```
def determine_outlier_thresholds_std(dataframe, col_name):
    upper_boundary = dataframe[col_name].mean() + 3 * dataframe[col_name].std()
    lower_boundary = dataframe[col_name].mean() - 3 * dataframe[col_name].std()
    return dataframe[(dataframe[col_name]<lower_boundary) | (dataframe[col_name]>upper_bou
def determine_outlier_thresholds_iqr(dataframe, col_name):
    quartile1 = dataframe[col_name].quantile(0.25)
    quartile3 = dataframe[col_name].quantile(0.75)
    iqr = quartile3 - quartile1
    upper_boundary = quartile3 + 1.5 * iqr
    lower_boundary = quartile1 - 1.5 * iqr
    return dataframe[(dataframe[col_name]<lower_boundary) | (dataframe[col_name]>upper_bou
#NOW I'LL APPLY SD TYPE FOR DEPTH AND IQR FOR OTHERS
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```

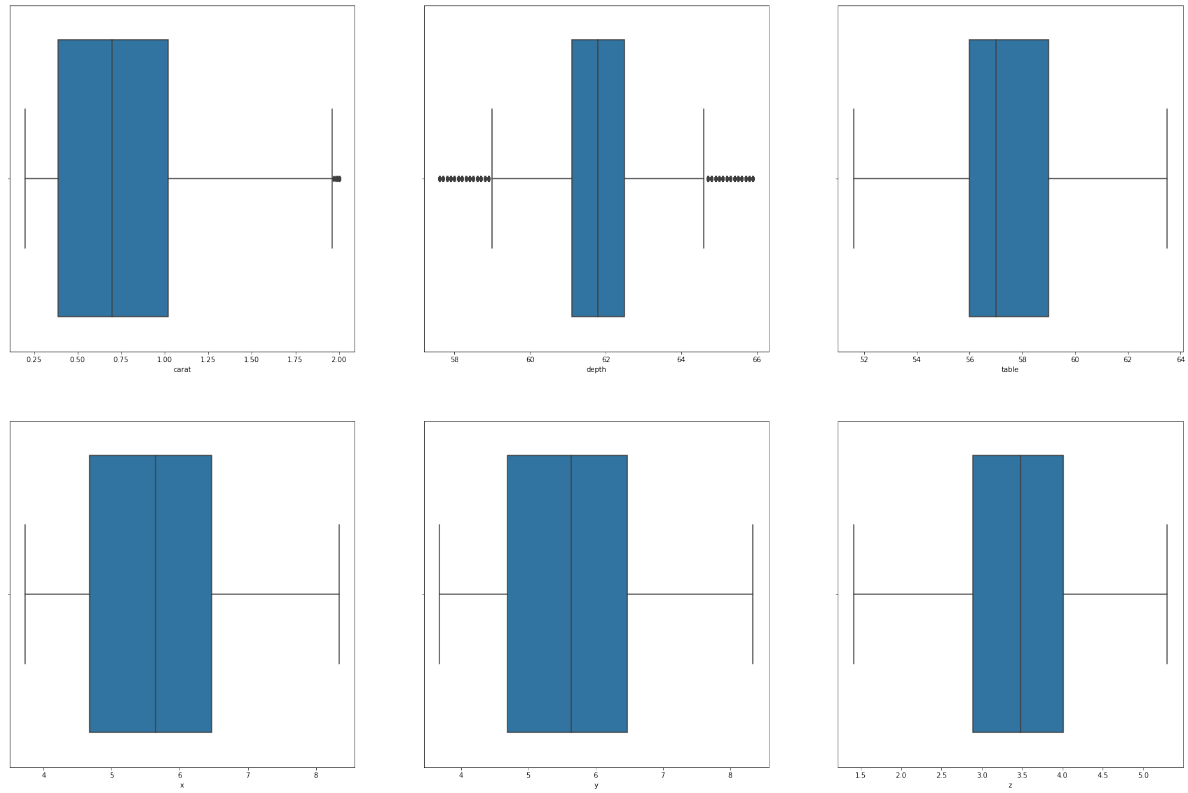



```
df.shape
```

```
(50752, 10)
```

Plotting the box-plot to visualise skew and outliers

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



As we can't use the classifier characteristics if they are in string format thus, we need to encode the classifiers to perform various operations on them as well.
Using labelEncoder we'll encode all the required classifier characteristics of the diamonds.

```
encoding_cut = LabelEncoder()
df['cut'] = encoding_cut.fit_transform(df['cut'])
encoding_color = LabelEncoder()
df['color'] = encoding_cut.fit_transform(df['color'])
encoding_clarity = LabelEncoder()
df['clarity'] = encoding_cut.fit_transform(df['clarity'])
#All the encodings are in alphabetical order
```

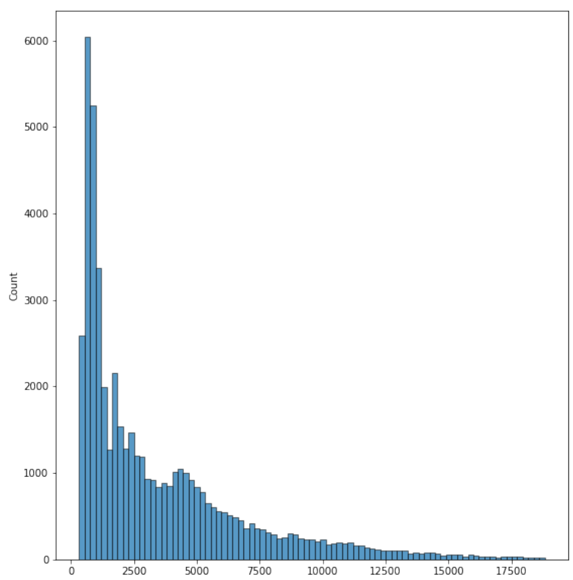
```
df.head()
```

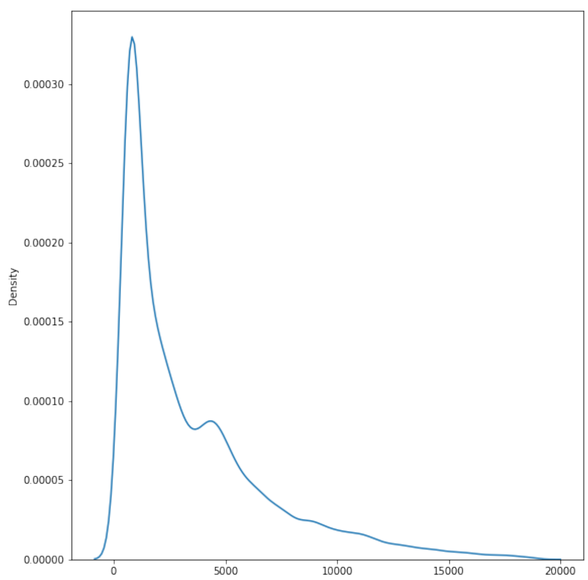
	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	2	1	3	61.5	55.0	326	3.95	3.98	2.43
1	0.21	3	1	2	59.8	61.0	326	3.89	3.84	2.31
3	0.29	3	5	5	62.4	58.0	334	4.20	4.23	2.63
4	0.31	1	6	3	63.3	58.0	335	4.34	4.35	2.75
5	0.24	4	6	7	62.8	57.0	336	3.94	3.96	2.48

As all the data has been cleaned, outliers been removed, and classifiers encoded. Now, we can perform different visualisation and analysis on the data.

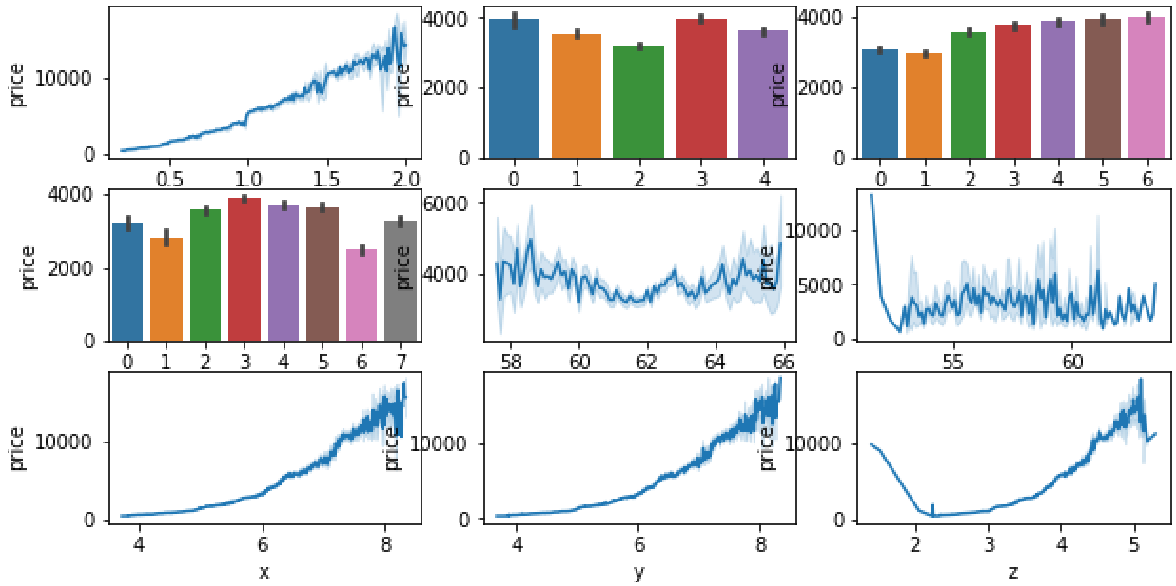
```
df. describe()
```

	carat	cut	color	clarity	depth	table	price
count	50752.000000	50752.000000	50752.000000	50752.000000	50752.000000	50752.000000	50752.00
mean	0.745581	2.598420	2.531664	3.883453	61.755338	57.328834	3535.926
std	0.401990	0.983383	1.677356	1.729265	1.252130	2.054813	3431.755
min	0.200000	0.000000	0.000000	0.000000	57.600000	51.600000	326.0000
25%	0.390000	2.000000	1.000000	2.000000	61.100000	56.000000	919.0000
50%	0.700000	2.000000	3.000000	4.000000	61.800000	57.000000	2268.000
75%	1.020000	3.000000	4.000000	5.000000	62.500000	59.000000	4974.000
max	2.000000	4.000000	6.000000	7.000000	65.900000	63.500000	18818.00





PLOTTING DIFFERENT CHARACTERISTICS WITH PRICE TO SEE RELATIONS



```
plt.figure(figsize=(10,10))
sns.heatmap(df.corr(), annot=True)
```

Plotting the heat map to visualise correlation matrix

```
<AxesSubplot:>
```

OBSERVATIONS:
1. It can be observed that price is linearly and highly correlated to x, y, and z.
2. Depth, Cut, and Color have minimal effect on price.
3. Carat is the biggest defining factor in the price of a diamond.