In [1]:
```
#importing drive acess of google drive
from google.colab import drive
drive.mount('/content/drive')
#changing directories of drive
import os
os.chdir("drive/My Drive/Assignment_dataset/Assignment regression ")
```

```
Mounted at /content/drive
```

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
data=pd.read_csv("diamonds.csv")
```

In [4]:
```
data
```

Out[4]:

	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 [5]:
```
data.head()
```

Out[5]:

	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 [6]:
```
data.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  object 
 2   color    53940 non-null  object 
 3   clarity  53940 non-null  object 
 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: float64(6), int64(1), object(3)
memory usage: 4.1+ MB
```

In [7]:
```
data.isnull().any()
```

Out[7]:
```
carat      False
cut        False
color      False
clarity    False
depth      False
table      False
price      False
x          False
y          False
z          False
dtype: bool
```

In [8]:
```
data.describe()
```

Out[8]:

	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 [9]:
```
# Droping the minimum value of x, y,z because it contains 0
data = data.drop(data[data["x"]==0].index)
data = data.drop(data[data["y"]==0].index)
data = data.drop(data[data["z"]==0].index)
data.shape
```

Out[9]:
```
(53920, 10)
```

In [10]:

```
data.describe()
# all the dimensional values  containing 0 in x,y,z removed
```

Out[10]:

	carat	depth	table	price	x	y	z
count	53920.000000	53920.000000	53920.000000	53920.000000	53920.000000	53920.000000	53920.000000
mean	0.797698	61.749514	57.456834	3930.993231	5.731627	5.734887	3.540046
std	0.473795	1.432331	2.234064	3987.280446	1.119423	1.140126	0.702530
min	0.200000	43.000000	43.000000	326.000000	3.730000	3.680000	1.070000
25%	0.400000	61.000000	56.000000	949.000000	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	5323.250000	6.540000	6.540000	4.040000
max	5.010000	79.000000	95.000000	18823.000000	10.740000	58.900000	31.800000

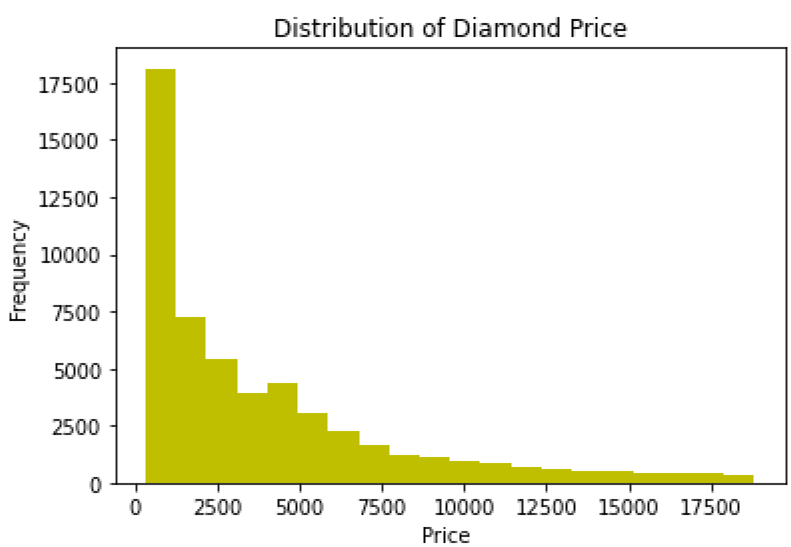
EDA

In [38]:

```
#subplot showing the diamond price distribution
plt.subplot()
plt.hist(data['price'],bins=20,color='y')
plt.xlabel('Price ')
plt.ylabel('Frequency')
plt.title('Distribution of Diamond Price')
```

Out[38]:

```
Text(0.5, 1.0, 'Distribution of Diamond Price')
```

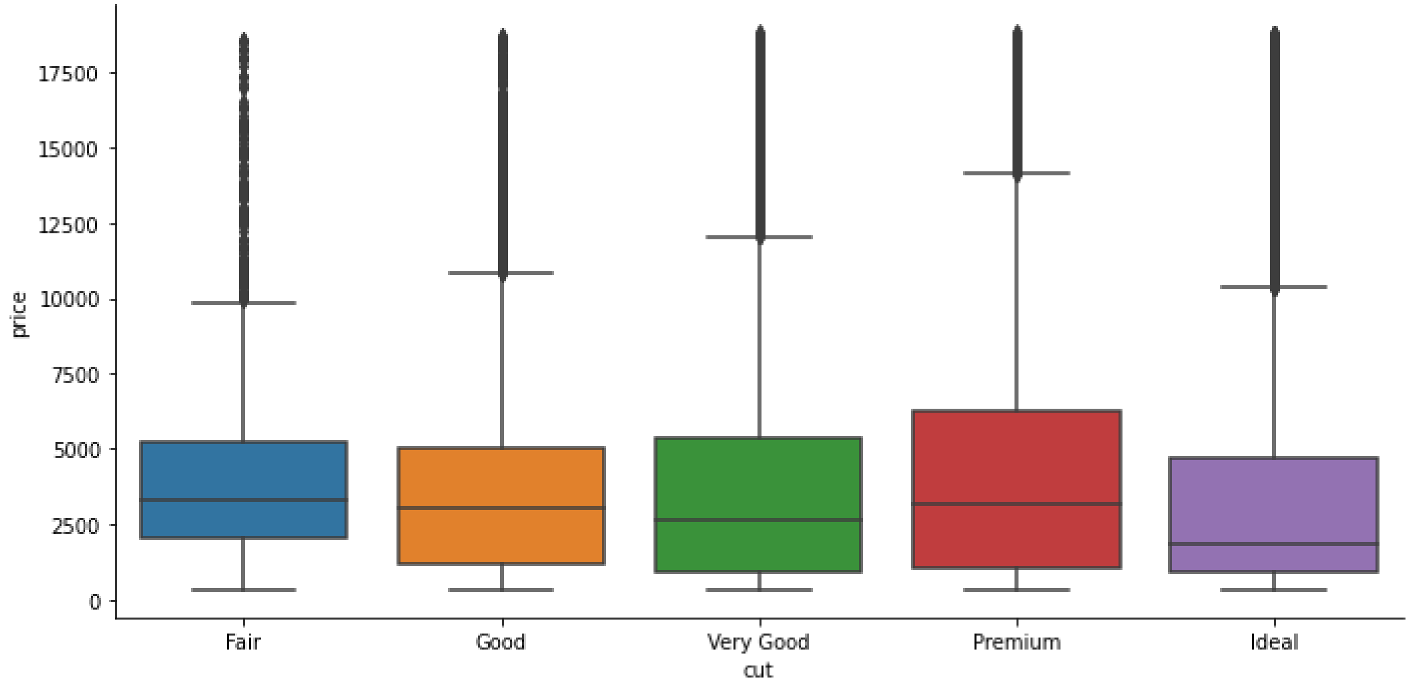


In [11]:

```
sns.catplot(x='cut', y = 'price',data=data ,aspect=2,kind='box' ,order=['Fair','Good','Ver
```

Out[11]:

```
<seaborn.axisgrid.FacetGrid at 0x7f1e2d8cae90>
```

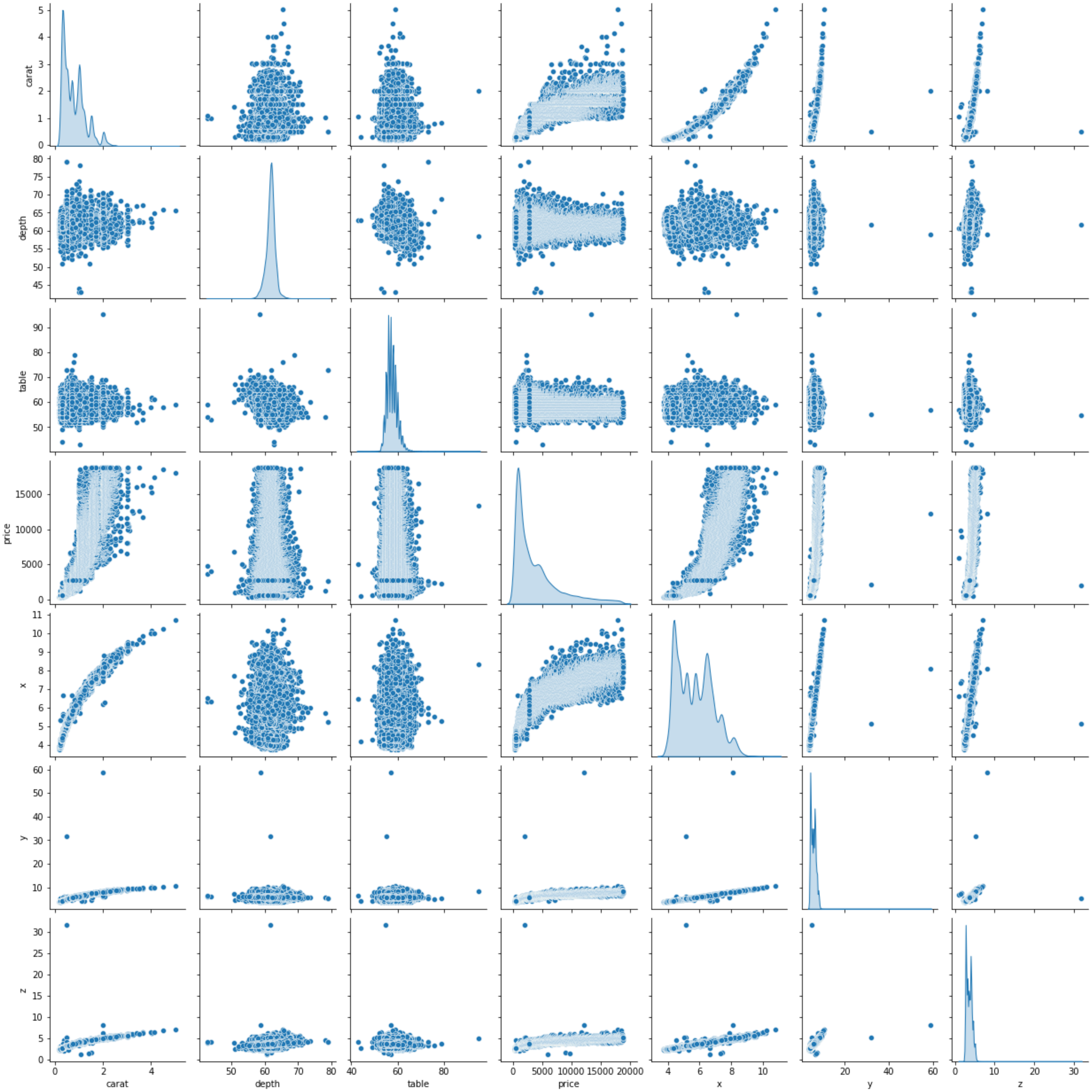


In [12]:

```
sns.pairplot(data, diag_kind = 'kde')
```

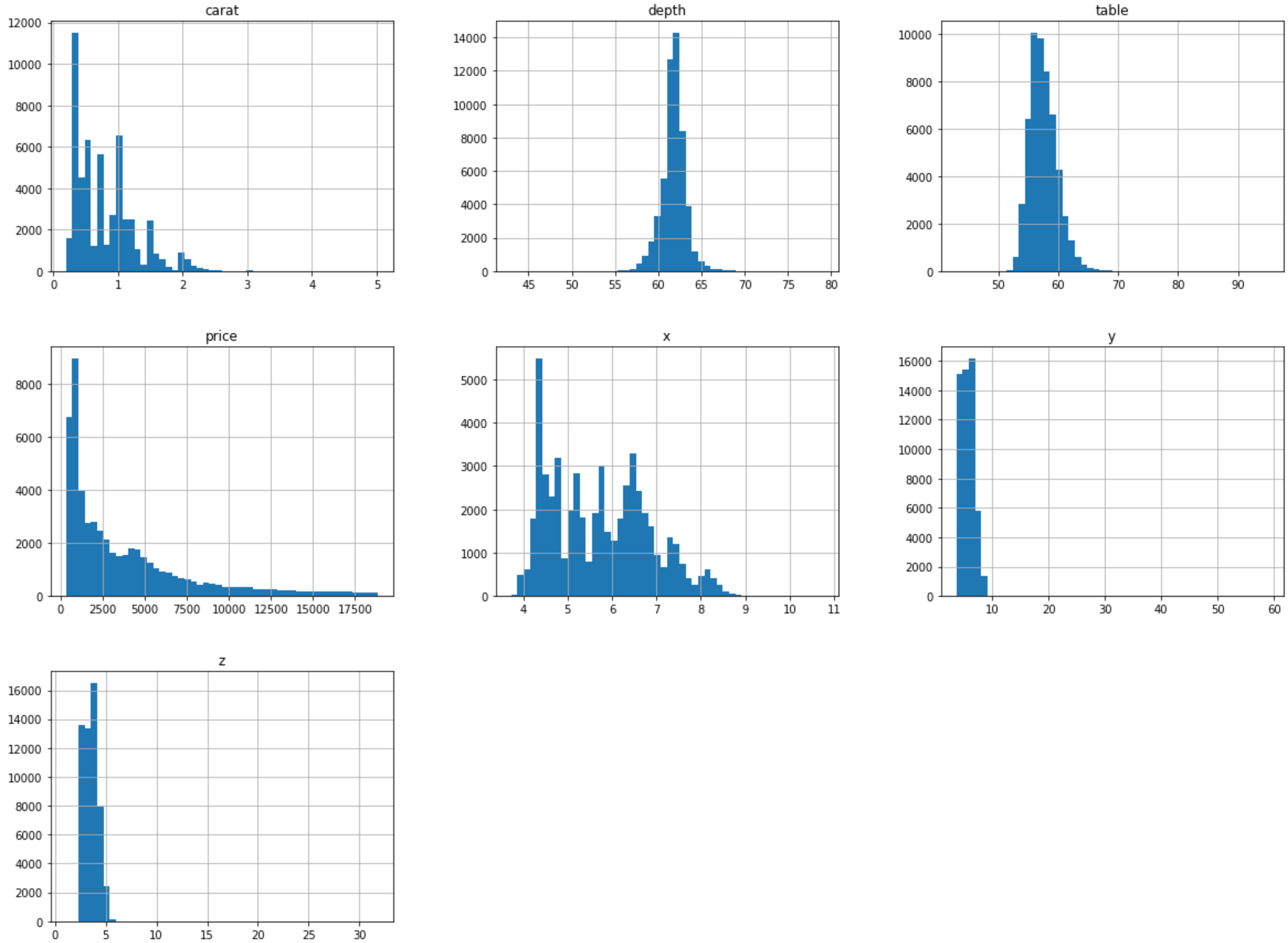
Out[12]:

```
<seaborn.axisgrid.PairGrid at 0x7f1e247bab90>
```



In [13]:

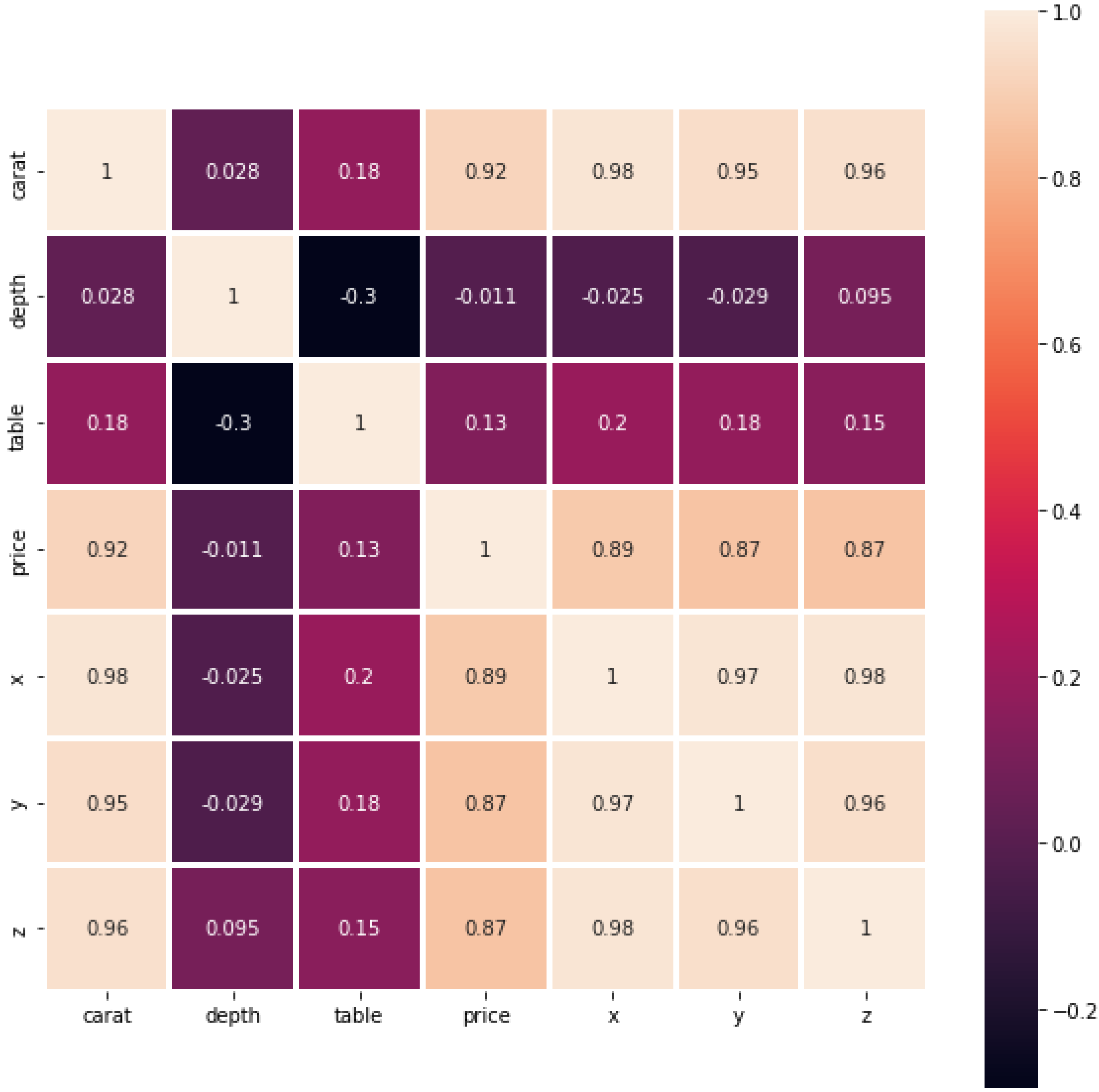
```
data.hist(bins=50,figsize=(20,15))
plt.show()
```



In [14]:

```
# coorelation of x,y,z wrt to price with the help of heat map
plt.figure(figsize=(10,10))
corr = data.corr()
sns.heatmap(data=corr, square=True , annot=True, cbar=True,linewidth=2)
```

Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0x7f1e1eede110>

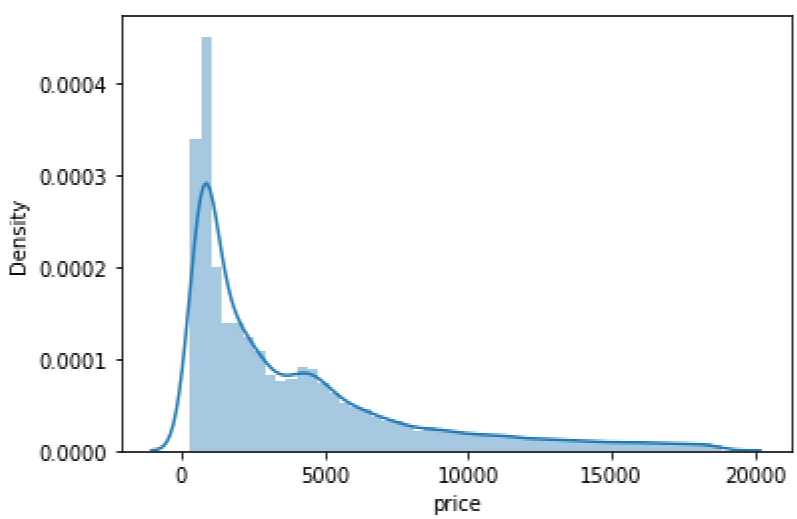


Distribution of the target variable ie. price

In [15]:

```
sns.distplot(data['price'])
```

Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x7f1e1ef08850>



the target variable ie. price is right skewed

In [16]:

```
print("The skewness of the Price in the dataset is {}".format(data['price'].skew()))
```

```
The skewness of the Price in the dataset is 1.6183486340820077
```

Handle categorical values

In [17]:

```
objList = data.select_dtypes(include = "object").columns
print (objList)
```

```
Index(['cut', 'color', 'clarity'], dtype='object')
```

In [18]:

```
#Use of Label encoding
from sklearn.preprocessing import OneHotEncoder, LabelEncoder
label_data=data.copy()
le = LabelEncoder()

for col  in objList:
    label_data[col] = le.fit_transform(label_data[col])

label_data.head()
```

Out[18]:

	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
2	0.23	1	1	4	56.9	65.0	327	4.05	4.07	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

In [19]:

```
data.describe()
```

Out[19]:

	carat	depth	table	price	x	y	z
count	53920.000000	53920.000000	53920.000000	53920.000000	53920.000000	53920.000000	53920.000000
mean	0.797698	61.749514	57.456834	3930.993231	5.731627	5.734887	3.540046
std	0.473795	1.432331	2.234064	3987.280446	1.119423	1.140126	0.702530
min	0.200000	43.000000	43.000000	326.000000	3.730000	3.680000	1.070000
25%	0.400000	61.000000	56.000000	949.000000	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	5323.250000	6.540000	6.540000	4.040000
max	5.010000	79.000000	95.000000	18823.000000	10.740000	58.900000	31.800000

train test

In [20]:

```
## Assigning the featurs as X and trarget as y
X=label_data.drop(['price'], axis=1)
y=label_data['price']
```

In [21]:

```
from sklearn.model_selection import train_test_split
X_train, X_test,y_train,y_test=train_test_split(X,y,test_size=0.25, random_state=7)
```

In [22]:

```
print(X.shape, X_train.shape, X_test.shape)
```

```
(53920, 9) (40440, 9) (13480, 9)
```

In [23]:

```
print(X_train)
print(y_train)
```

```
       carat  cut  color  clarity  depth  table     x     y     z
19372   0.27    2      1        6   62.3   56.0  4.15  4.17  2.59
53321   0.71    4      0        2   63.2   56.0  5.69  5.73  3.61
47554   0.53    4      1        4   61.8   60.0  5.13  5.22  3.20
52695   0.70    2      1        5   61.9   57.0  5.69  5.74  3.54
9843    1.05    2      2        3   60.9   56.0  6.64  6.56  4.02
...      ...  ...    ...      ...    ...    ...   ...   ...   ...
919     0.72    0      2        4   56.9   69.0  5.93  5.77  3.33
53479   0.71    2      5        5   62.2   55.0  5.71  5.73  3.56
38484   0.42    2      3        7   60.7   55.0  4.86  4.89  2.96
10747   1.00    1      2        2   63.4   61.0  6.29  6.35  4.01
49708   0.71    3      3        3   60.6   59.0  5.78  5.74  3.49

[40440 rows x 9 columns]
19372     622
53321    2652
47554    1874
52695    2553
9843     4675
         ...
919      2879
53479    2681
38484    1031
10747    4851
49708    2147
Name: price, Length: 40440, dtype: int64
```

Linear Regression

In [24]:

```
from sklearn.linear_model import LinearRegression
from sklearn import model_selection
from sklearn.metrics import r2_score, mean_squared_error
model=LinearRegression()
model.fit(X_train,y_train)
```

Out[24]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [25]:

```
# model prediction on test data
x_pred=model.predict(X_test)
x_pred
```

Out[25]:

```
array([3335.03388112, 6173.38377446, 2423.91873959, ...,  237.16257322,
       9821.42296742, 1068.83869592])
```

In [26]:

```
lr_score =model.score(X_test, y_test)
```

In [27]:

```
lr_score
```

Out[27]:

```
0.8875471978452595
```

Support Vector Regression

In [30]:

```
from sklearn import preprocessing, svm
X_svm = X.copy()
X_svm = preprocessing.scale(X_svm)

X_svm_train, X_svm_test, y_svm_train, y_svm_test = train_test_split(X_svm, y, test_size=0.
clf= svm.SVR(kernel='linear')
clf.fit(X_svm_train, y_svm_train)
```

Out[30]:
```
SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='scale',
    kernel='linear', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
```

In [31]:
```
svr_score = clf.score(X_svm_test,y_svm_test)
svr_score
```

Out[31]:
```
0.8360762856315977
```

Decision tree

In [32]:
```
from sklearn.tree import DecisionTreeRegressor
dt_regressor = DecisionTreeRegressor(random_state=0)
dt_regressor.fit(X_train, y_train)
```

Out[32]:
```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=None,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=0, splitter='best')
```

In [48]:
```
rt_score = dt_regressor.score(X_test, y_test)
rt_score
```

Out[48]:
```
0.9643764491979789
```

Random forest

In [43]:
```
from sklearn.ensemble import RandomForestRegressor
regressor_rf = RandomForestRegressor(n_estimators=100, random_state=0)
regressor_rf.fit(X_train, y_train)
```

Out[43]:
```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=None, oob_score=False,
                      random_state=0, verbose=0, warm_start=False)
```

In [44]:
```
rf_score = regressor_rf.score(X_test, y_test)
```

In [45]:
```
rf_score
```

Out[45]:
```
0.9805942678805571
```

We can conclude that the Random Forest Regression model performed the best with an accuracy of 98.05%

In [28]: