DIAMOND PRICE PREDICTION

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
from sklearn.model_selection import cross_val_score,train_test_split
from sklearn.metrics import r2_score,mean_absolute_error,
mean_squared_error
from sklearn.preprocessing import LabelEncoder,
RobustScaler,StandardScaler
from sklearn.linear_model import LinearRegression,Ridge
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import GradientBoostingRegressor,
RandomForestRegressor,AdaBoostRegressor
from sklearn.svm import LinearSVR

data=pd.read_csv("diamonds.csv.zip")

data.head()
```

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


      z
0  2.43
1  2.31
2  2.31
3  2.63
4  2.75
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53940 entries, 0 to 53939
Data columns (total 11 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  53940 non-null  int64
```

```
 1   carat       53940 non-null  float64
 2   cut         53940 non-null  object
 3   color       53940 non-null  object
 4   clarity     53940 non-null  object
 5   depth       53940 non-null  float64
 6   table       53940 non-null  float64
 7   price       53940 non-null  int64
 8   x           53940 non-null  float64
 9   y           53940 non-null  float64
 10  z           53940 non-null  float64
dtypes: float64(6), int64(2), object(3)
memory usage: 4.5+ MB
```

```
data.describe()
```

```
         Unnamed: 0         carat         depth         table
price  \
count  53940.000000  53940.000000  53940.000000  53940.000000
53940.000000
mean   26970.500000      0.797940     61.749405     57.457184
3932.799722
std    15571.281097      0.474011      1.432621      2.234491
3989.439738
min        1.000000      0.200000     43.000000     43.000000
326.000000
25%    13485.750000      0.400000     61.000000     56.000000
950.000000
50%    26970.500000      0.700000     61.800000     57.000000
2401.000000
75%    40455.250000      1.040000     62.500000     59.000000
5324.250000
max    53940.000000      5.010000     79.000000     95.000000
18823.000000
```

```
                 x             y             z
count  53940.000000  53940.000000  53940.000000
mean       5.731157      5.734526      3.538734
std        1.121761      1.142135      0.705699
min        0.000000      0.000000      0.000000
25%        4.710000      4.720000      2.910000
50%        5.700000      5.710000      3.530000
75%        6.540000      6.540000      4.040000
max       10.740000     58.900000     31.800000
```

DROPPING THE UNNAMED:0 COLUMNS AS IT IS HAVING NO IMPORTANCE

```
data.drop(columns='Unnamed: 0', inplace=True)
```

```
data.shape
```

```
(53940, 10)
```

```
data.isnull().mean()*100
```

```
carat      0.0
cut        0.0
color      0.0
clarity    0.0
depth      0.0
table      0.0
price      0.0
x          0.0
y          0.0
z          0.0
dtype: float64
```

```
data.duplicated().sum()
```

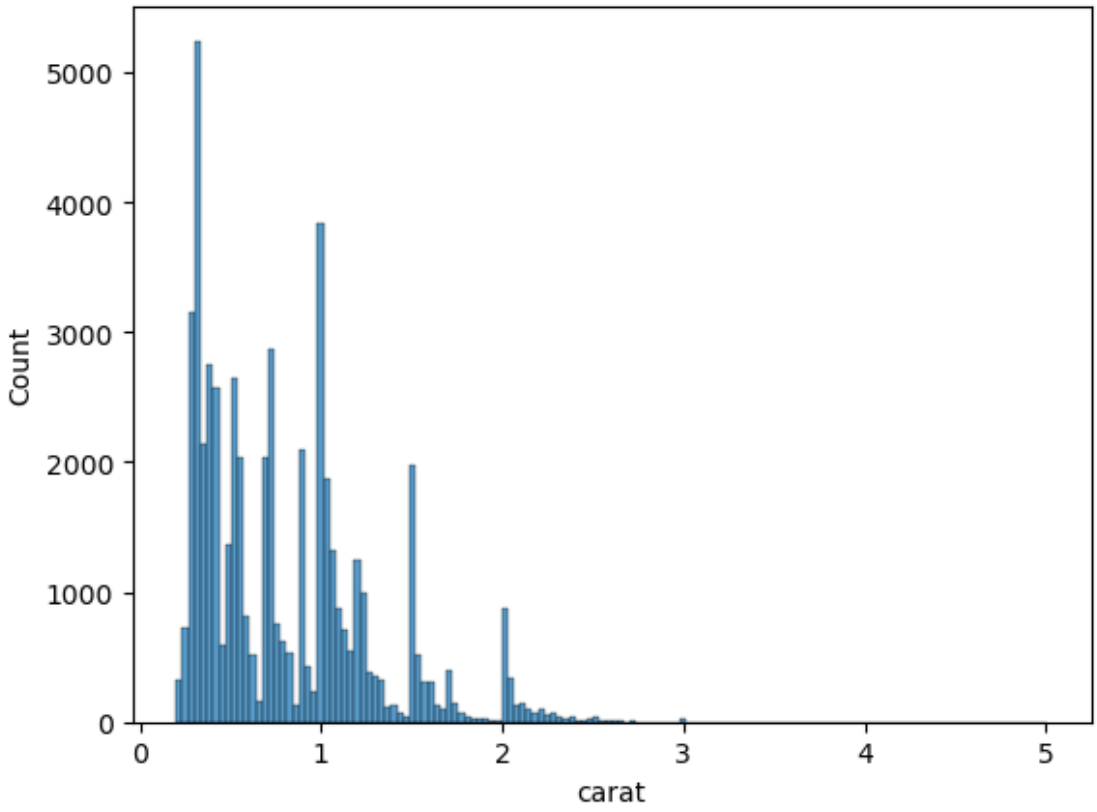
```
146
```

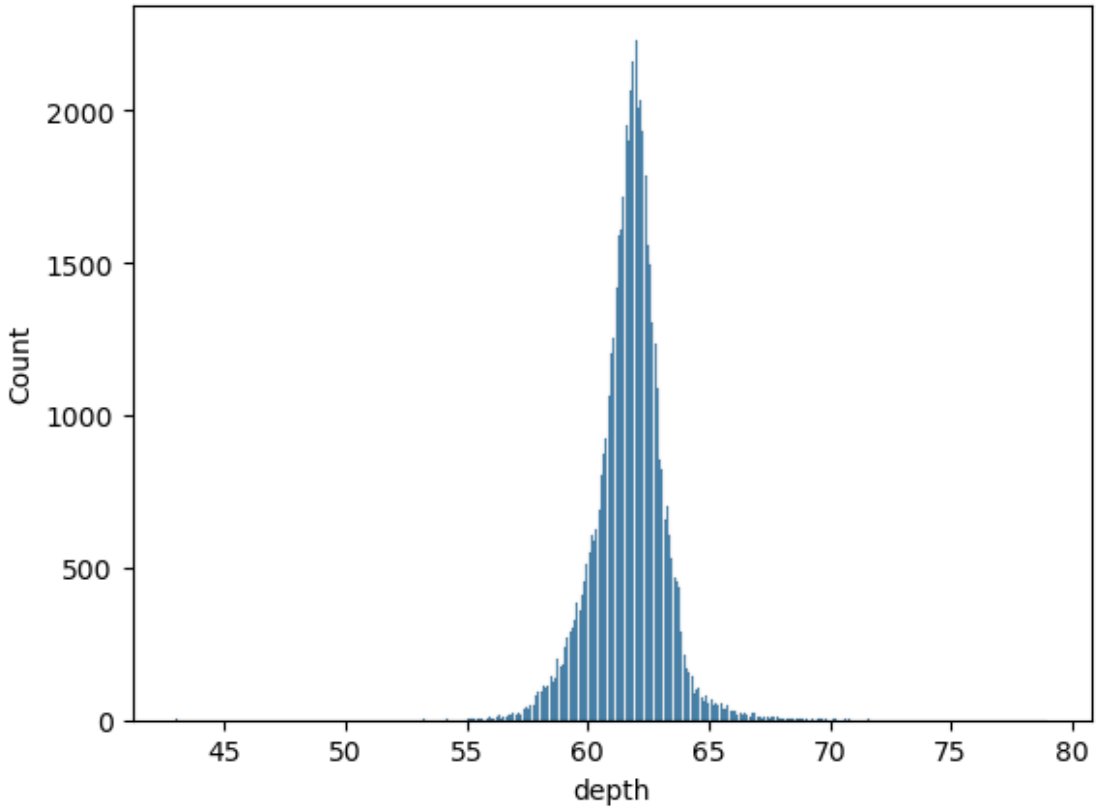
```
data.drop_duplicates(inplace=True)
```

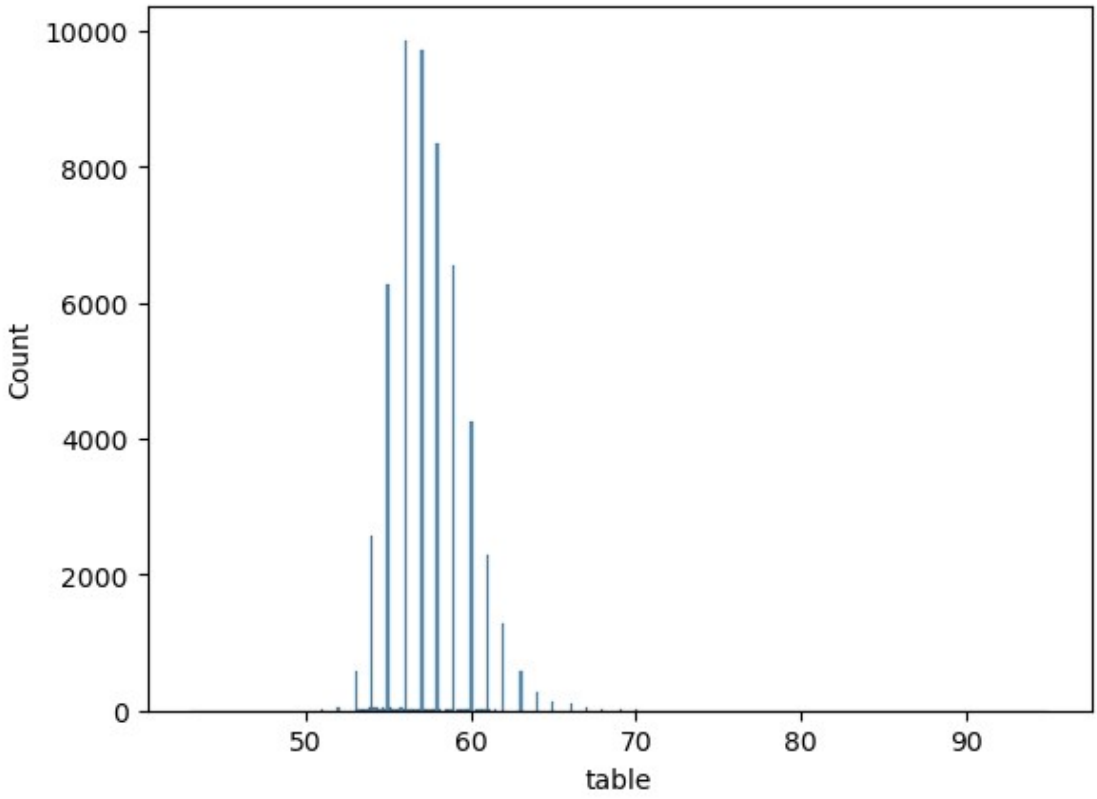
UNIVARIATE ANALYSIS

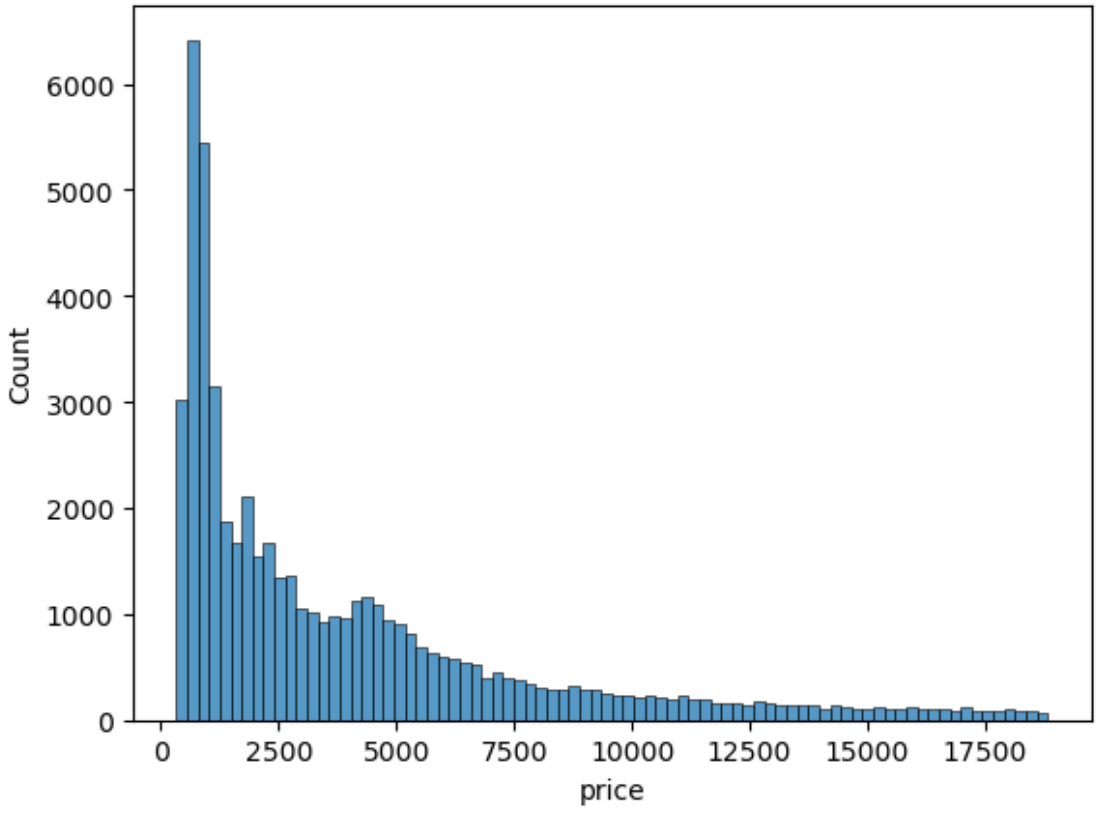
HIST PLOT

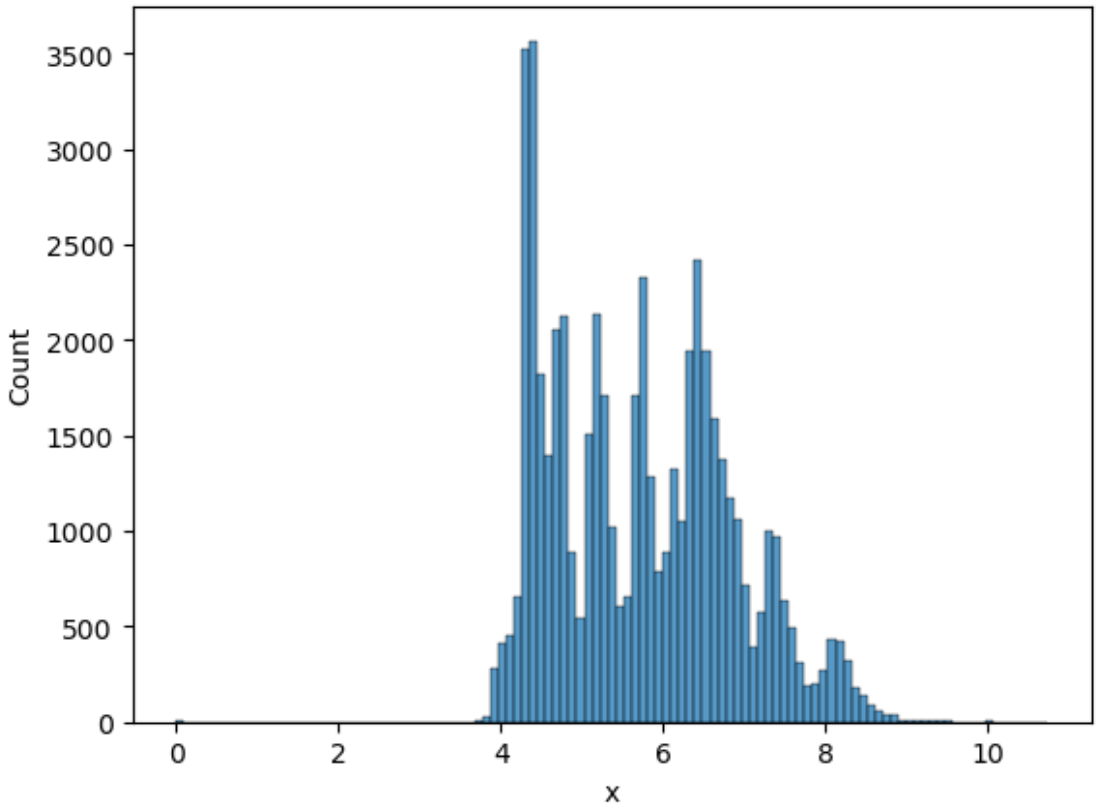
```
for i in data.columns:
    if data[i].dtypes != "object":
        sns.histplot(x =data[i] )
        plt.show()
```

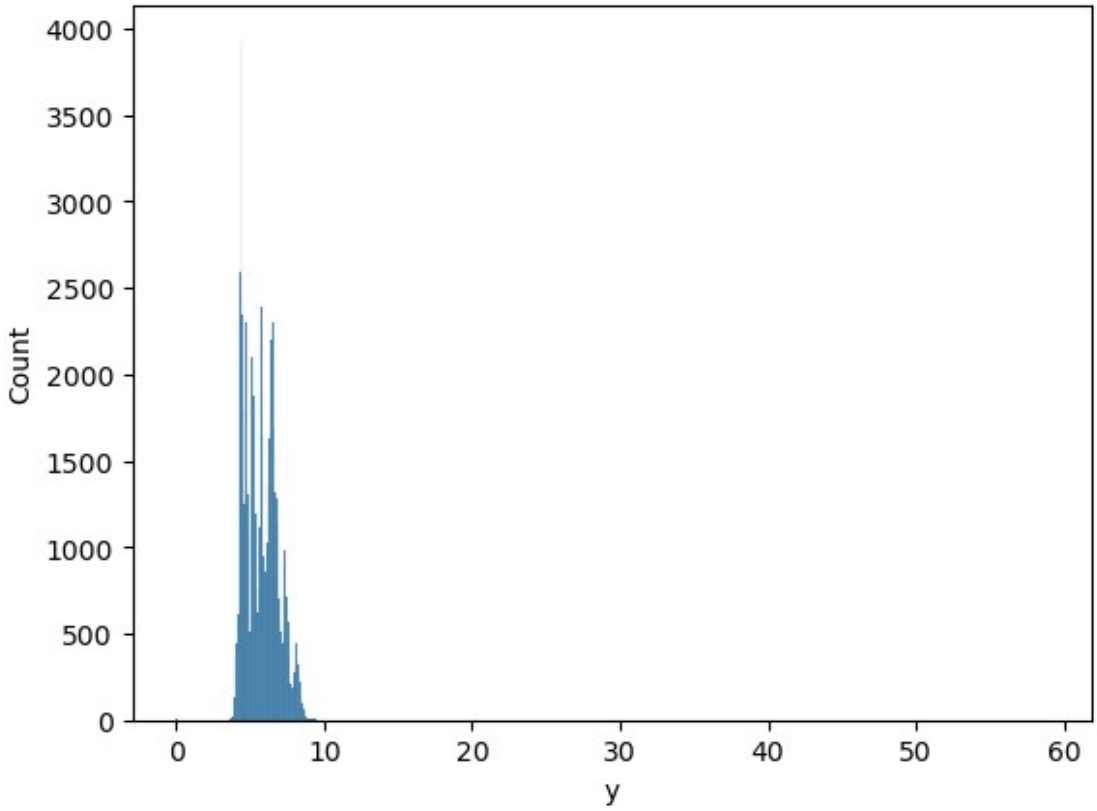

Count
5000
4000
3000
2000
1000
0
0
1
2
3
4
5
carat

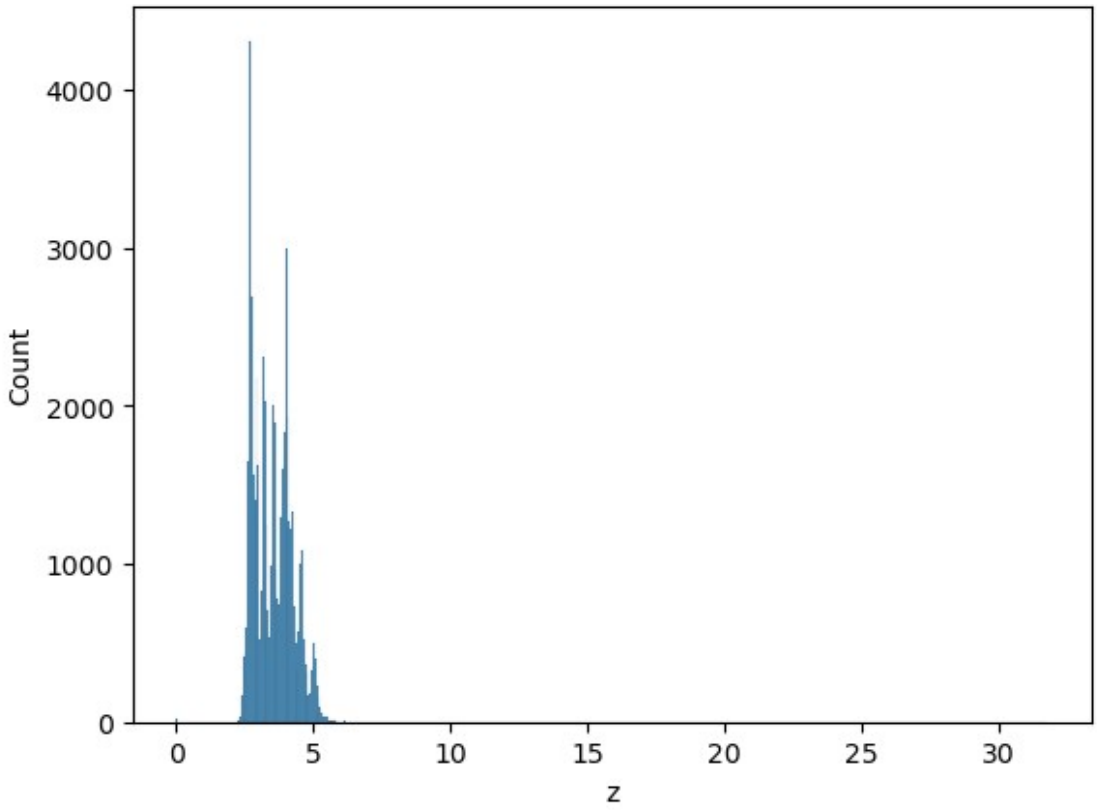

Count
2000
1500
1000
500
0
45
50
55
60
65
70
75
80
depth


10000
8000
6000
4000
2000
0
Count
50
60
70
80
90
table


6000
5000
4000
3000
2000
1000
0
Count
0
2500
5000
7500
10000
12500
15000
17500
price

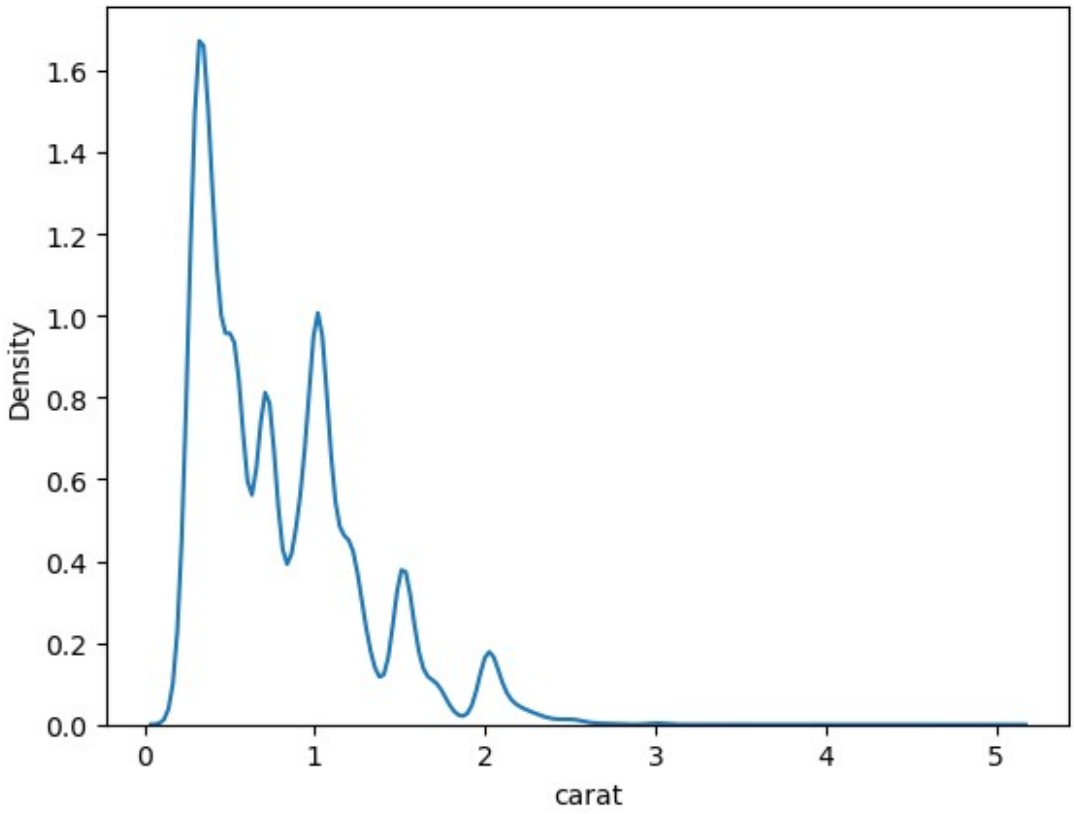

Count
3500
3000
2500
2000
1500
1000
500
0
0
2
4
6
8
10
x

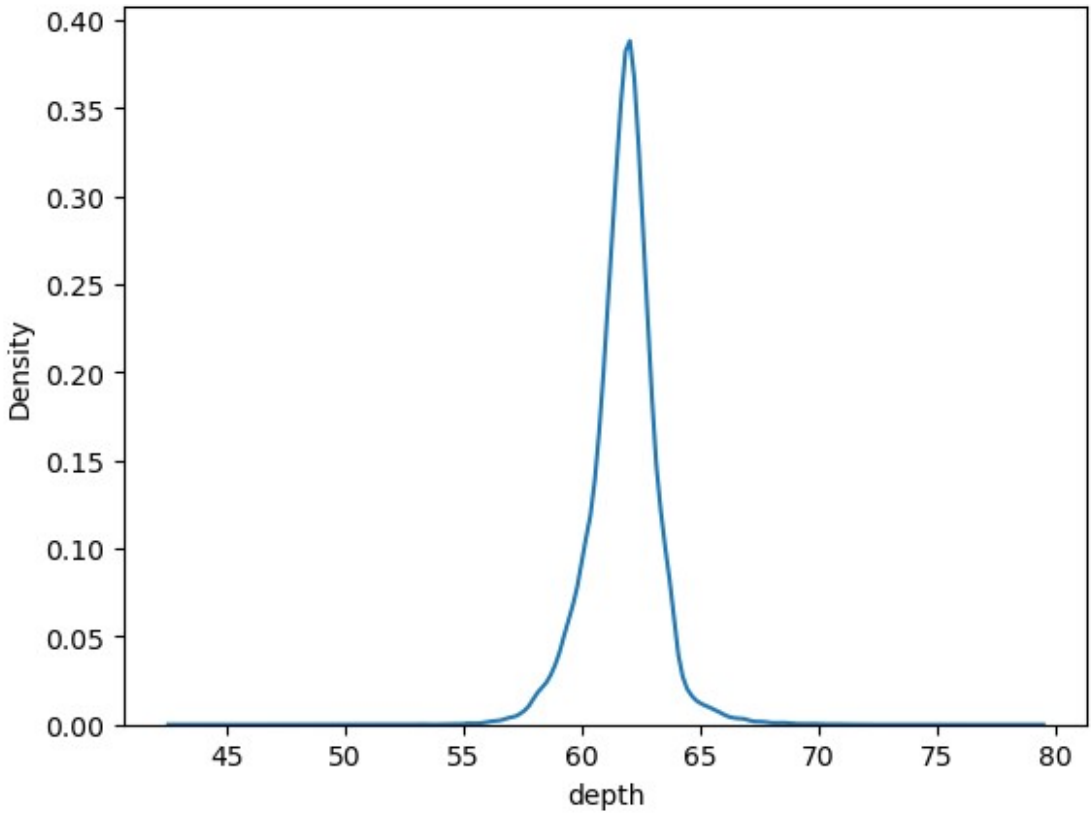

Count
4000
3500
3000
2500
2000
1500
1000
500
0
0
10
20
30
40
50
60
y

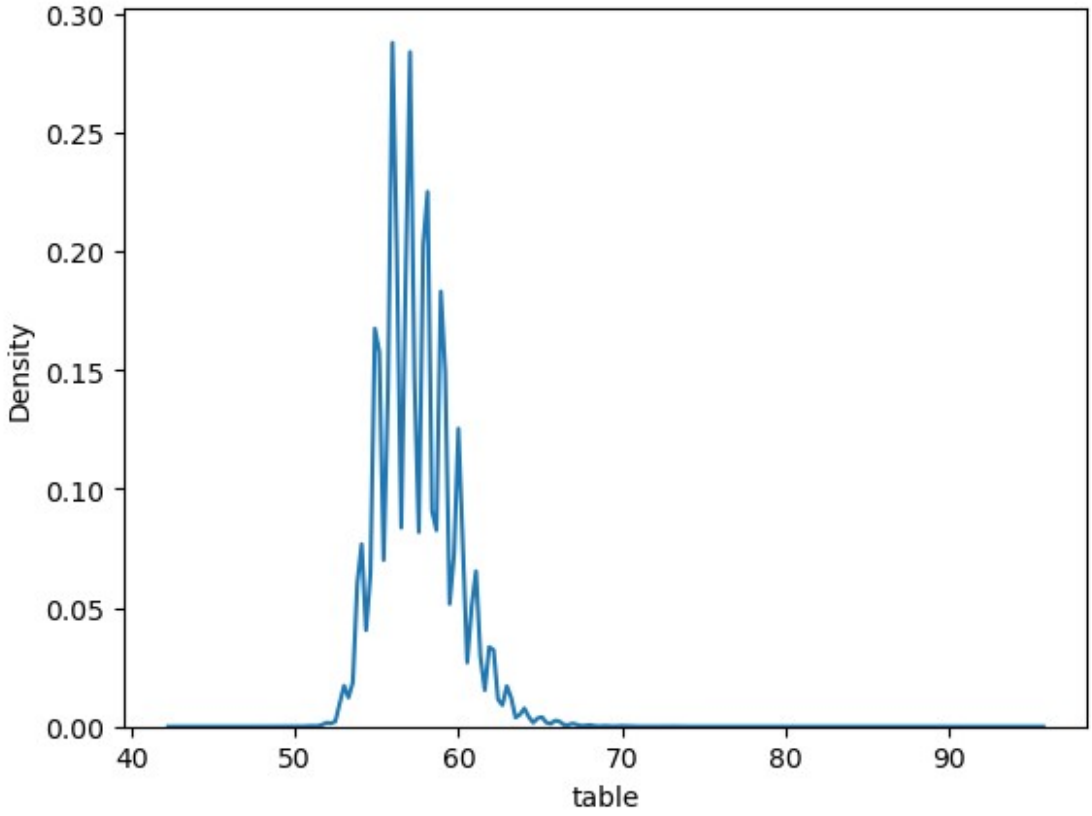


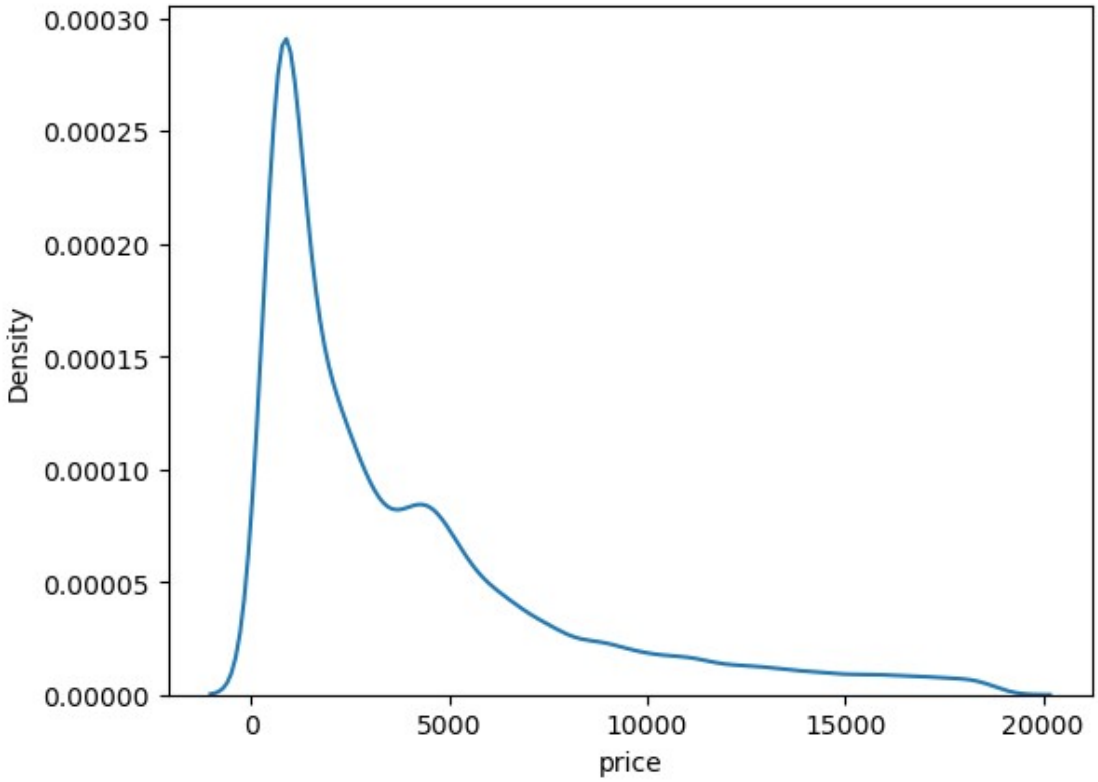
KDE PLOT

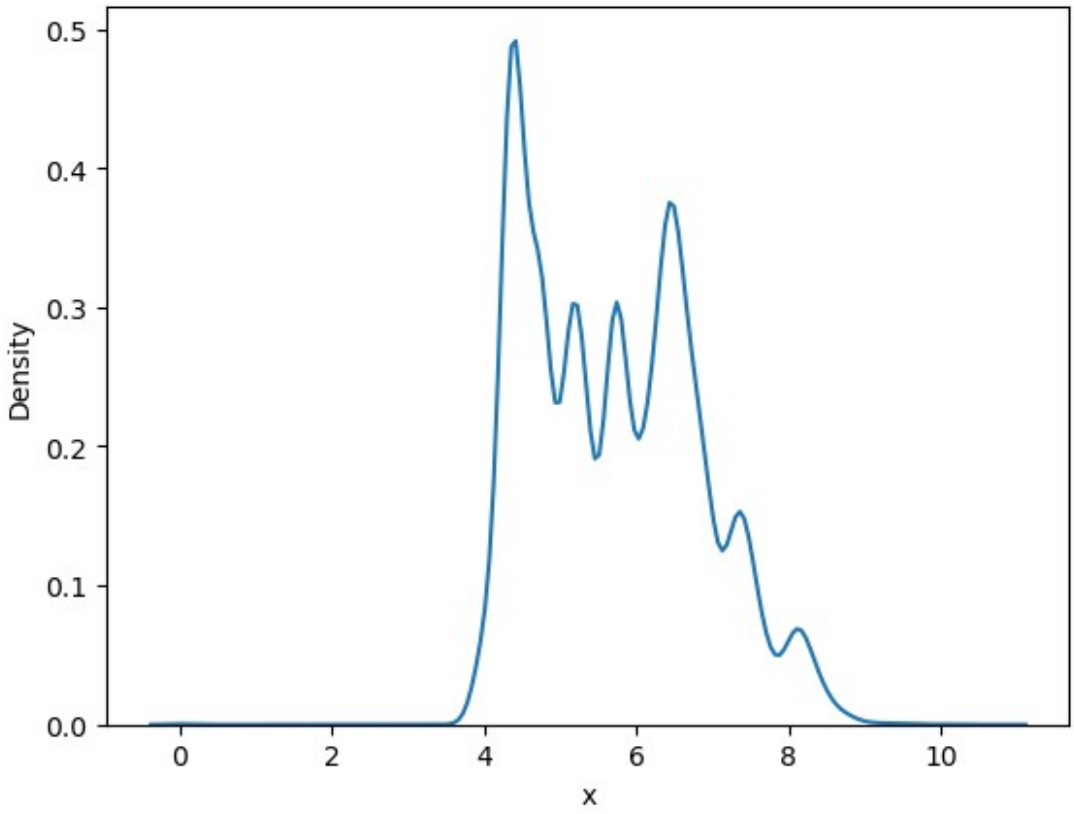
```
for i in data.columns:
    if data[i].dtypes != "object":
        sns.kdeplot(x =data[i] )
        plt.show()
```

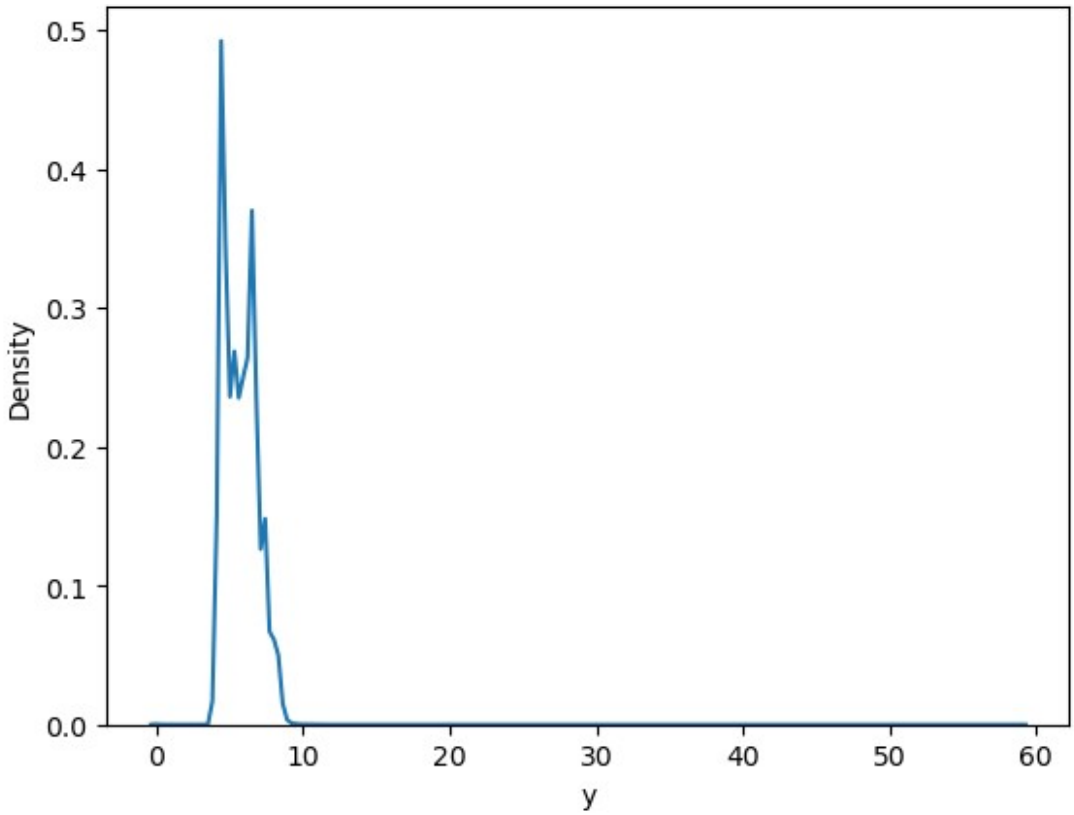

Density
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
5
carat

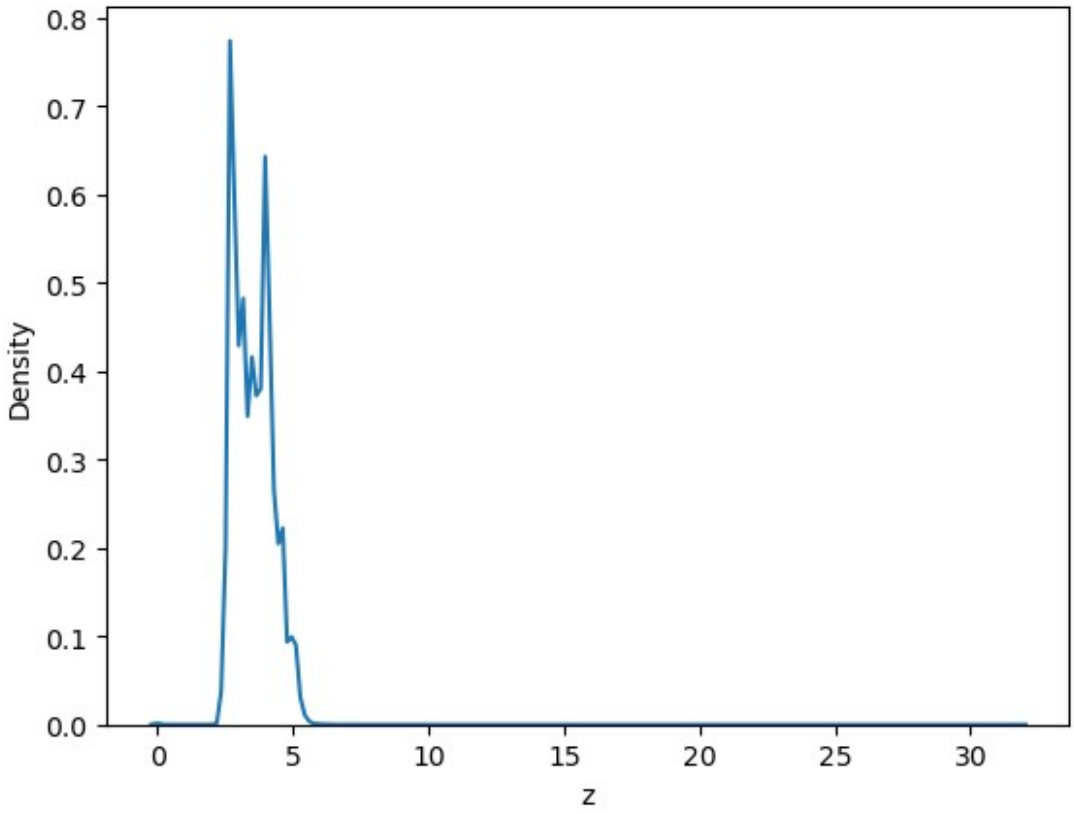

Density
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
45
50
55
60
65
70
75
80
depth


0.30
0.25
0.20
0.15
0.10
0.05
0.00
Density
40
50
60
70
80
90
table


0.00030
0.00025
0.00020
0.00015
0.00010
0.00005
0.00000
Density
0
5000
10000
15000
20000
price

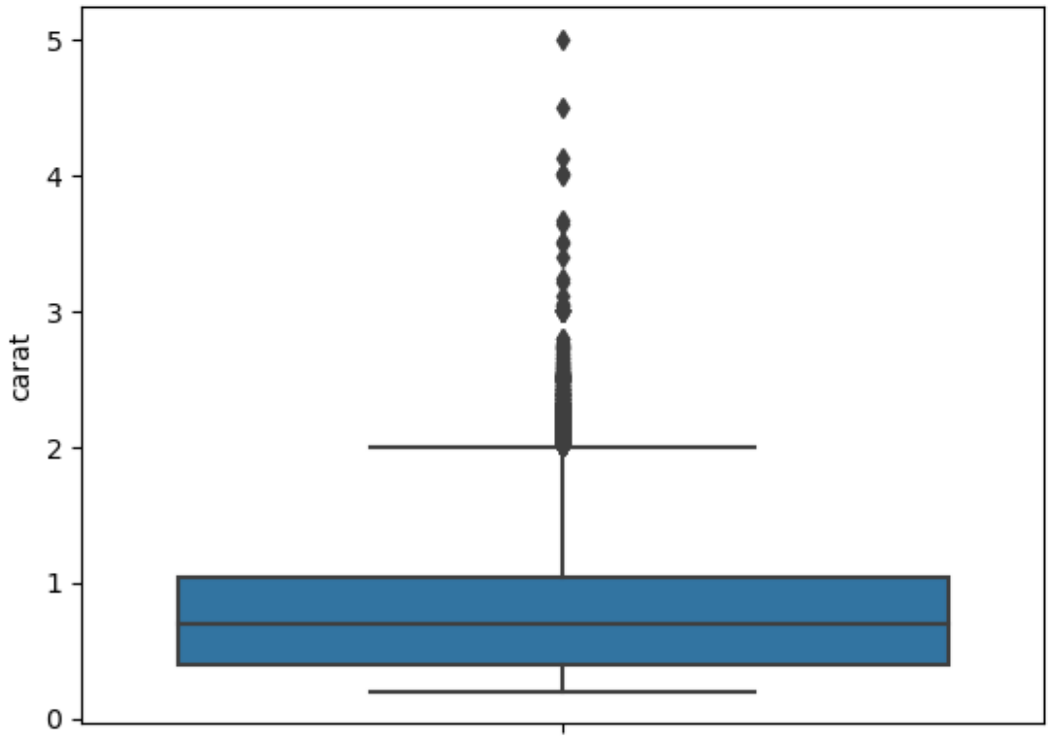

0.5
0.4
0.3
0.2
0.1
0.0
Density
0
2
4
6
8
10
x

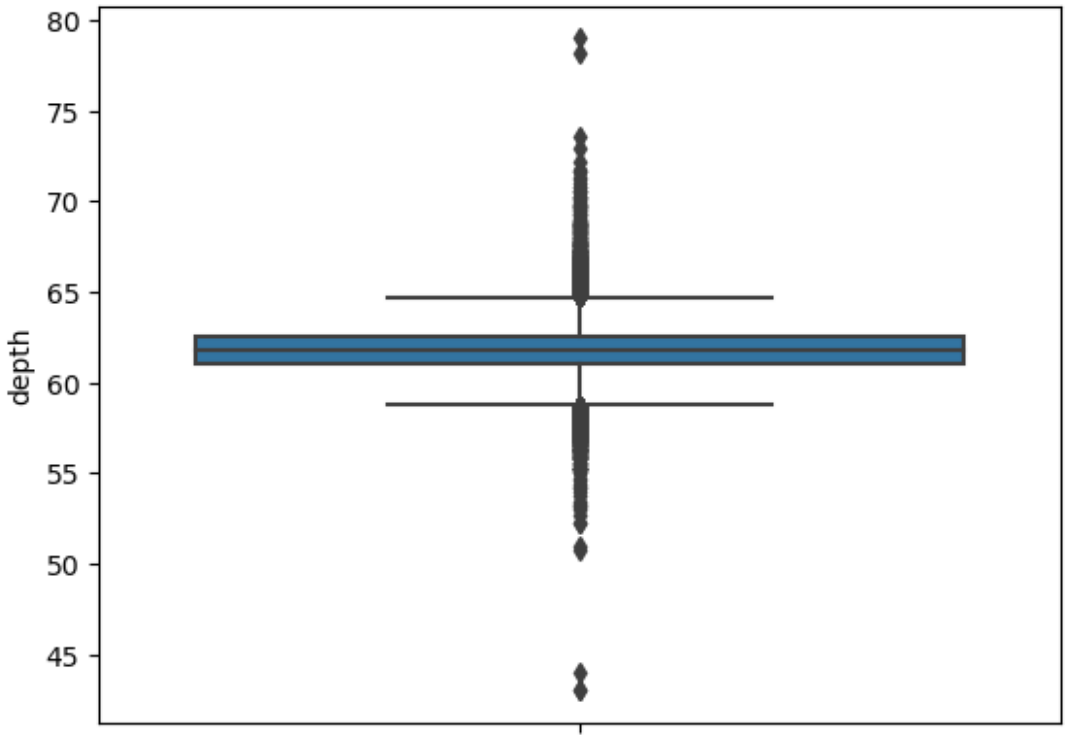

0.5
0.4
0.3
0.2
0.1
0.0
Density
0
10
20
30
40
50
60
y



BOX PLOT

```
for i in data.columns:
    if data[i].dtype !="object":
        sns.boxplot(y =data[i])
        plt.show();
```


carat
0
1
2
3
4
5


depth
45
50
55
60
65
70
75
80

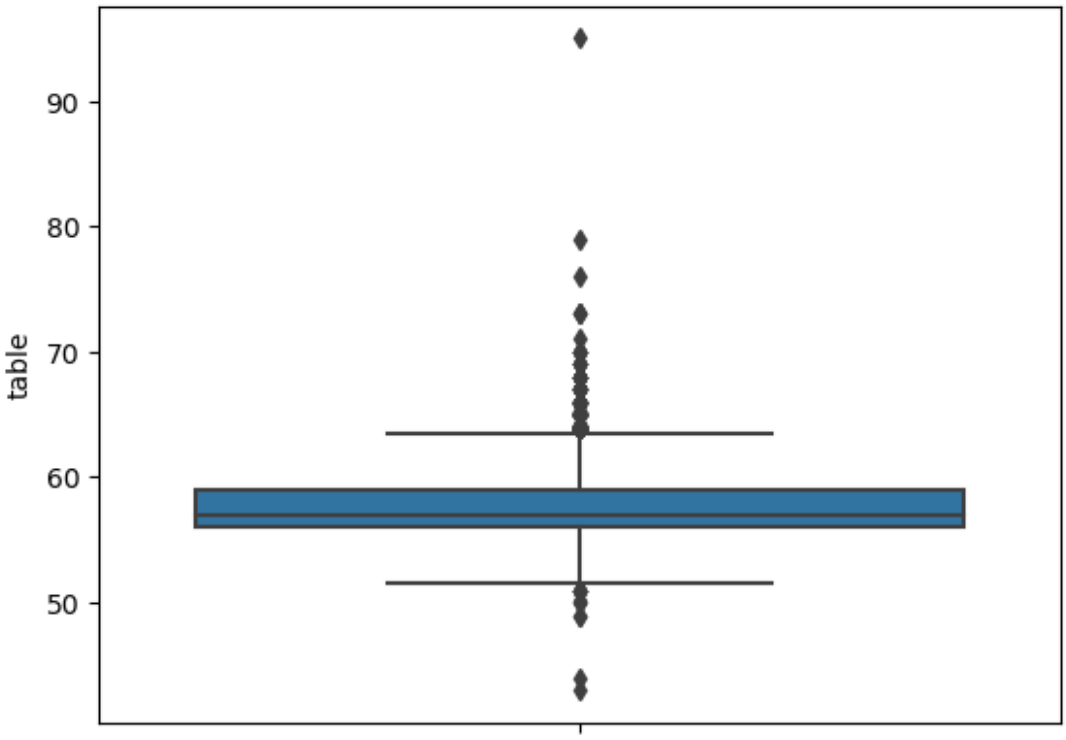
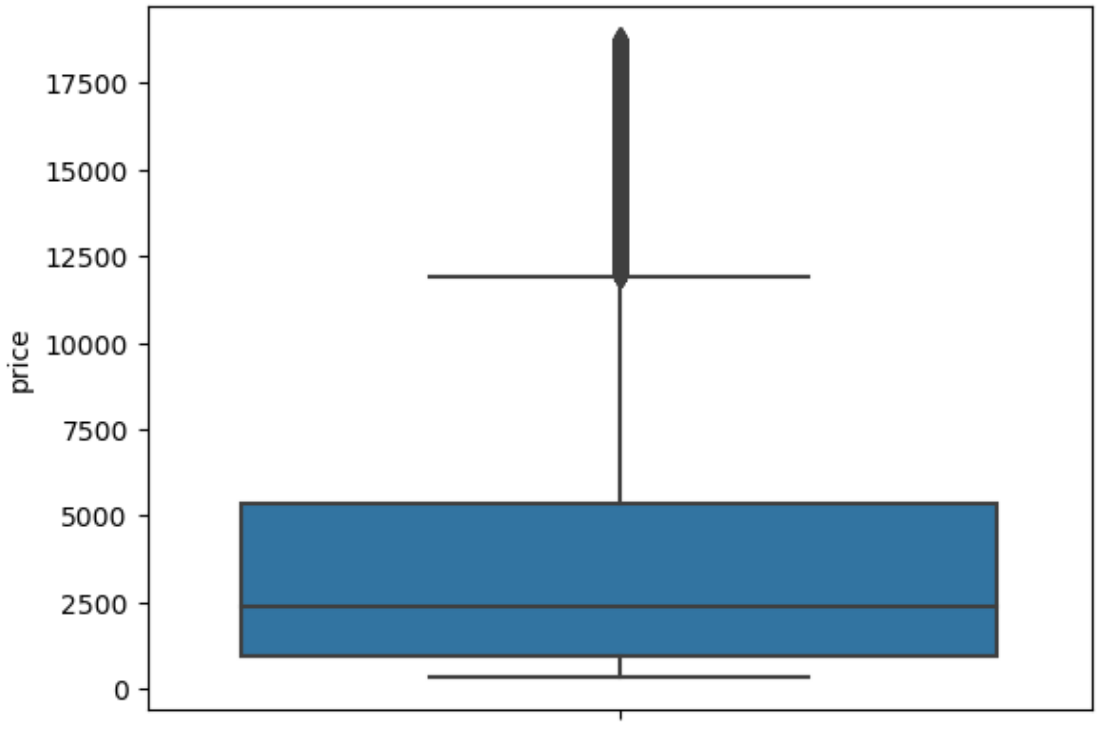
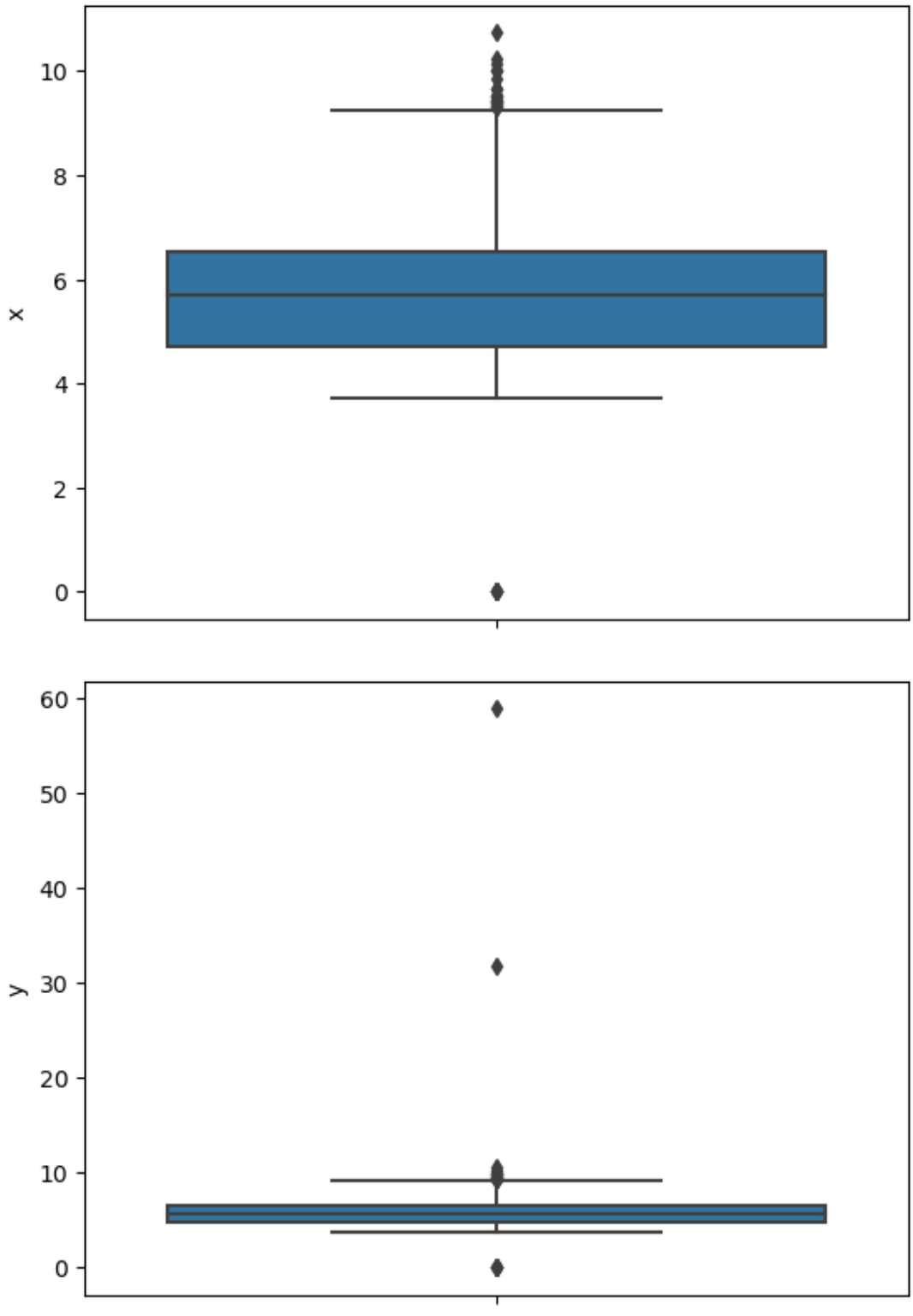
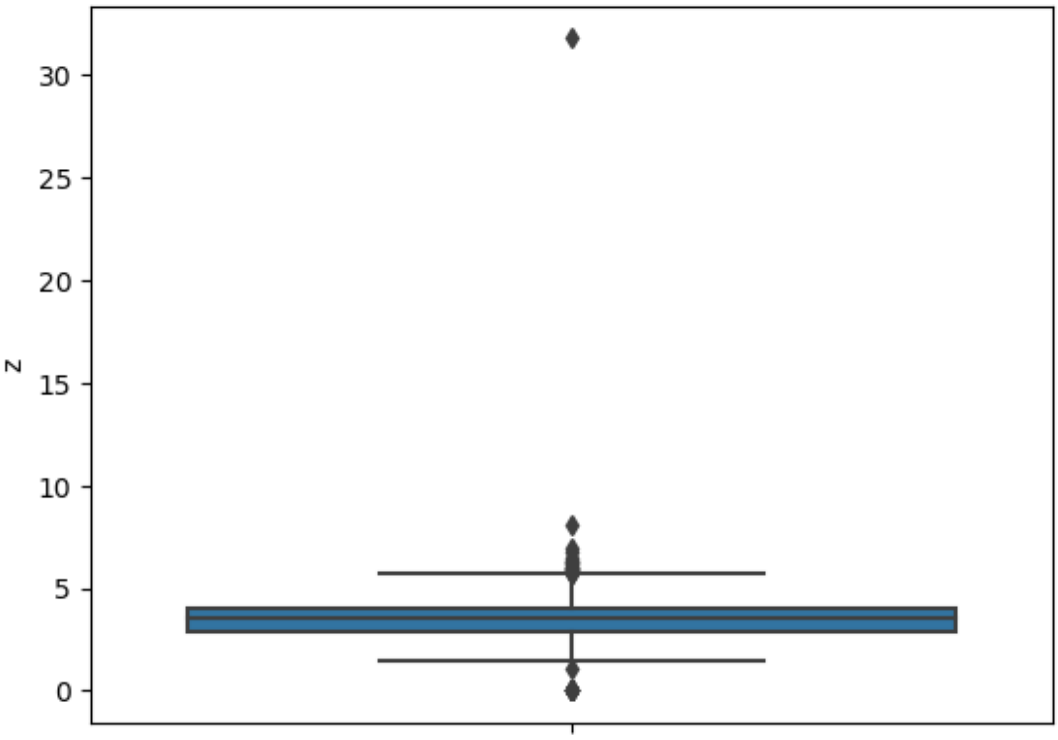


table
90
80
70
60
50

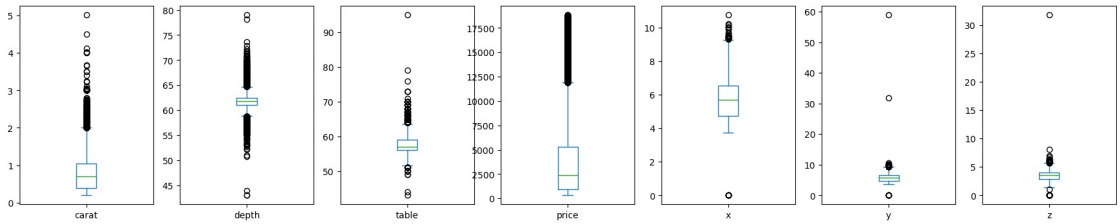

price
17500
15000
12500
10000
7500
5000
2500
0


x
10
8
6
4
2
0
y
60
50
40
30
20
10
0



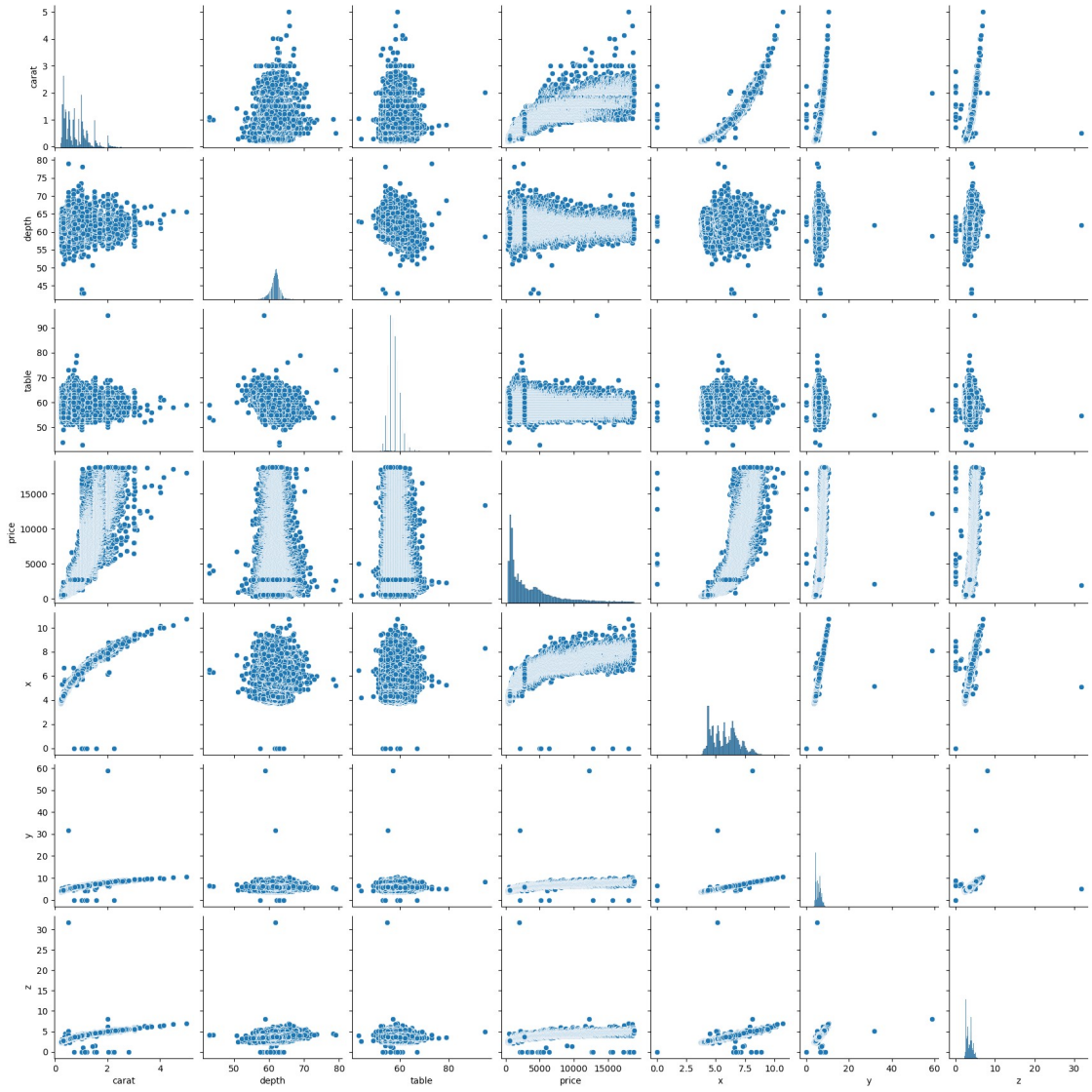
SUB PLOT

```
data.plot.box(figsize=(22,4),subplots=True);
```



PAIR PLOT

```
sns.pairplot(data=data)
plt.show()
```



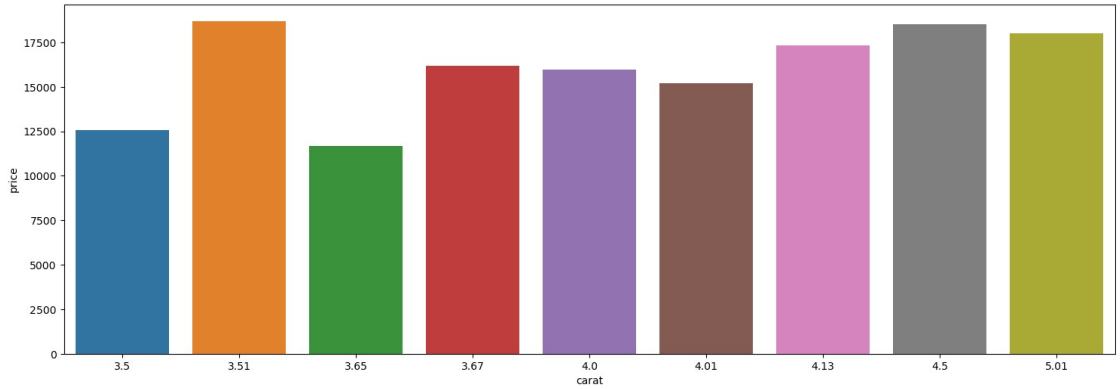
MULTI VARIATE ANALYSIS

```
data.columns
```

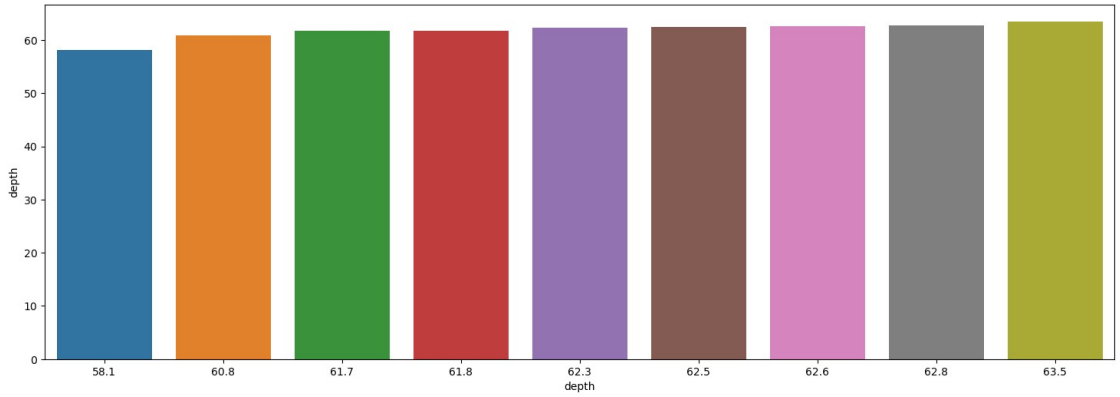
```
Index(['carat', 'cut', 'color', 'clarity', 'depth', 'table', 'price',
'x', 'y',
       'z'],
      dtype='object')
```

BAR PLOT

```
plt.figure(figsize=(18,6))
sns.barplot(x='carat', y='price',
data=data.sort_values(by='carat',ascending=False)[:10]);
```

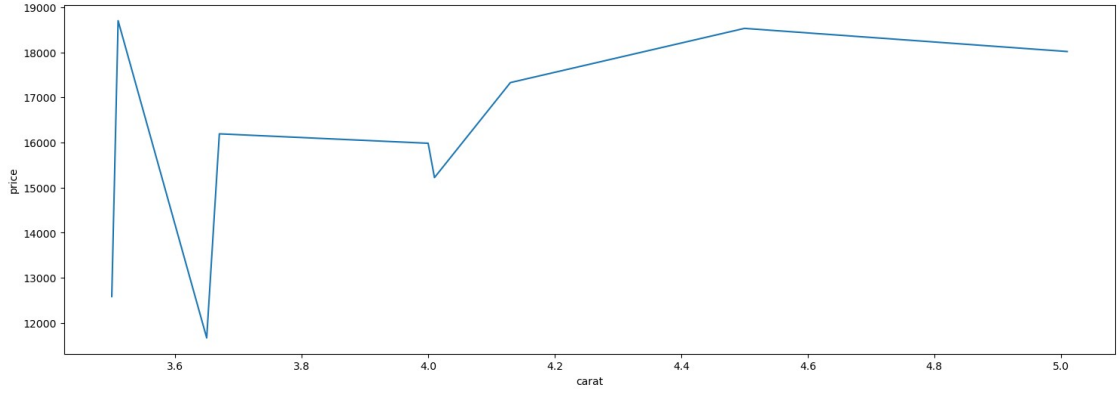



```
plt.figure(figsize=(18,6))
sns.barplot(x='depth', y='depth',
data=data.sort_values(by='price',ascending=False)[:10]);
```



LINE CHART

```
plt.figure(figsize=(18,6))
sns.lineplot(x='carat', y='price',
data=data.sort_values(by='carat',ascending=False)[:10]);
```



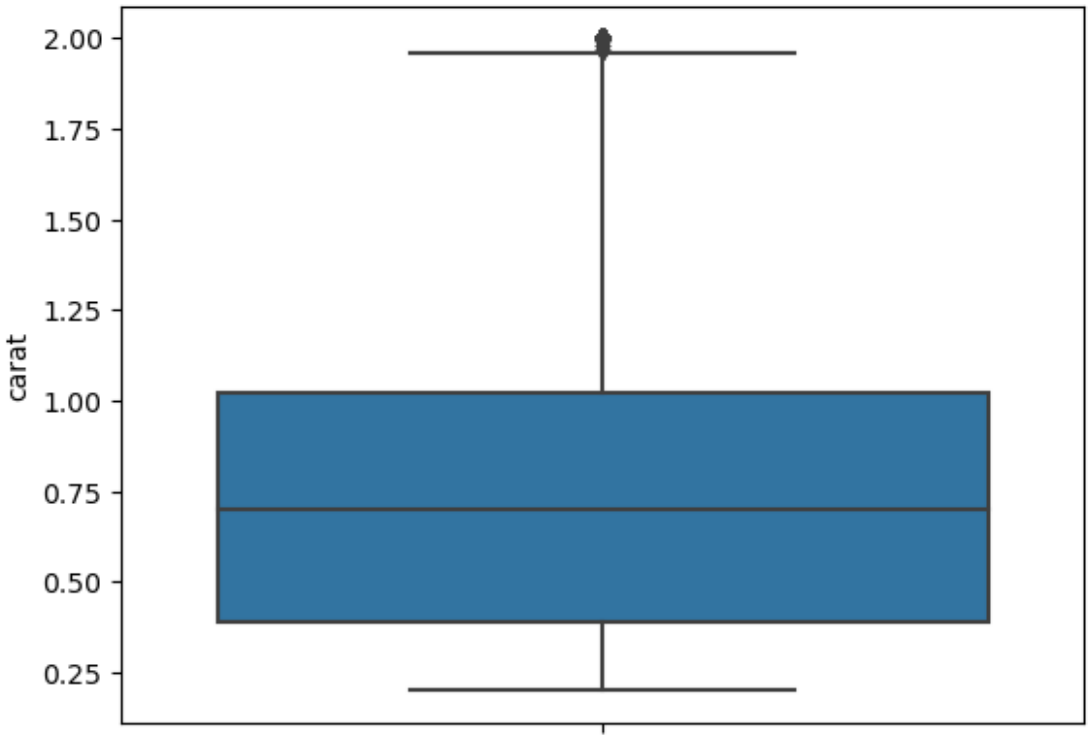

```
data.info()
```

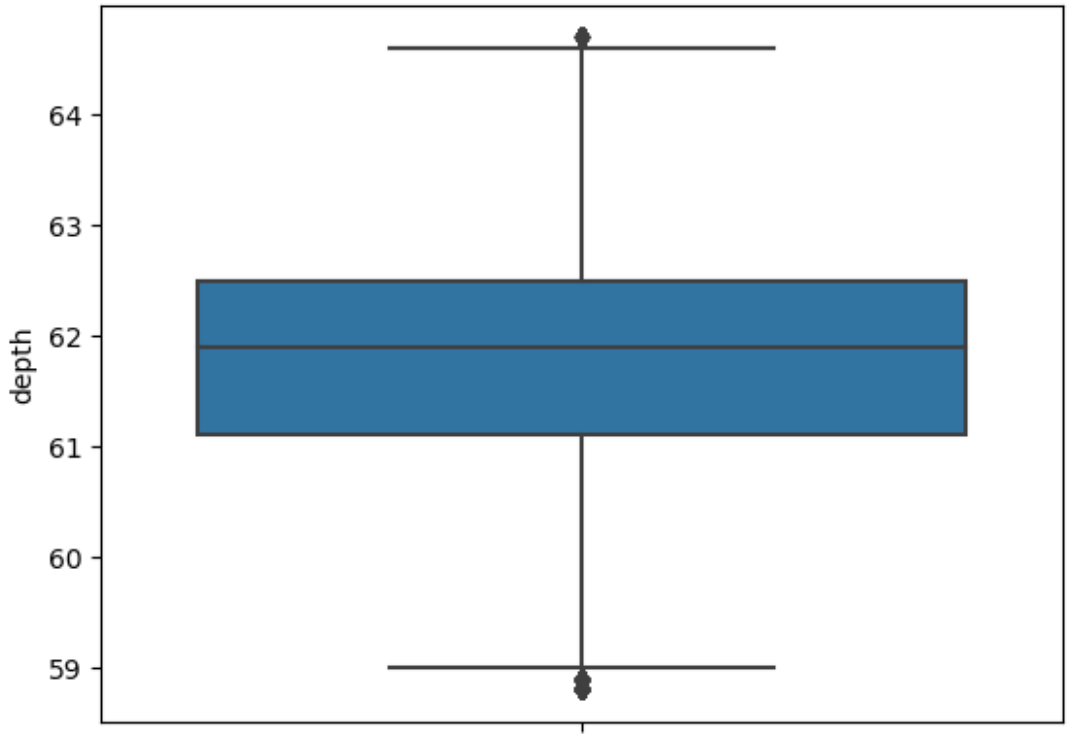
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 53794 entries, 0 to 53939
```

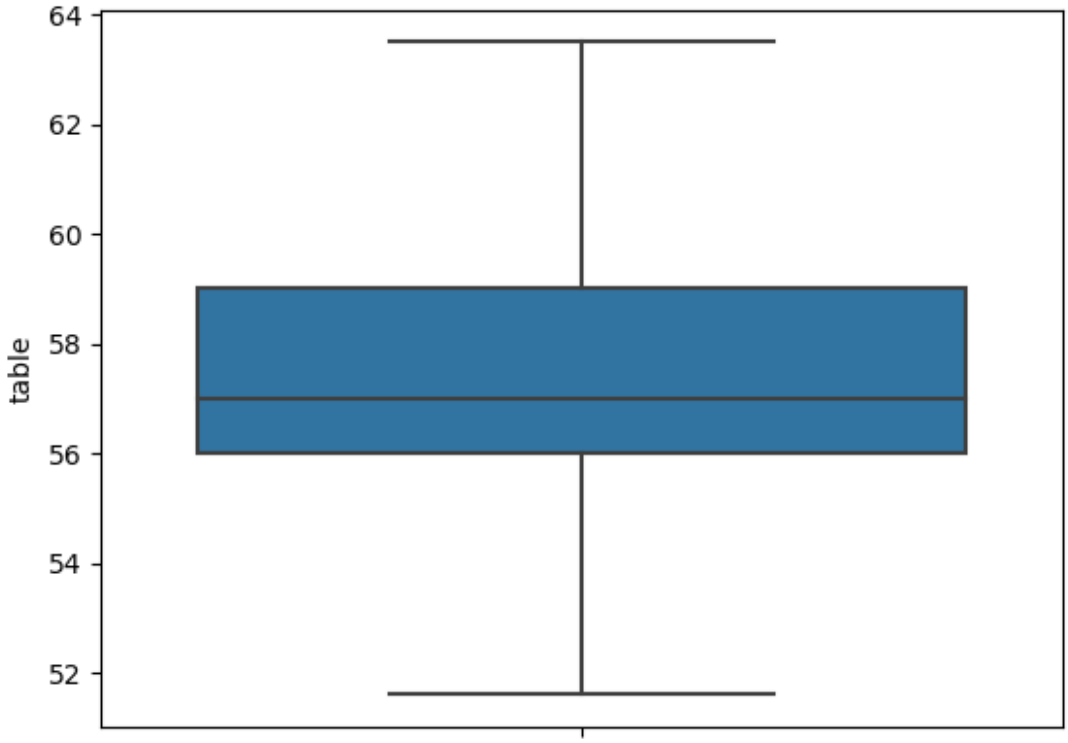
```
Data columns (total 10 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   carat    53794 non-null  float64
 1   cut      53794 non-null  object
 2   color    53794 non-null  object
 3   clarity  53794 non-null  object
 4   depth    53794 non-null  float64
 5   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: 6.5+ MB
```

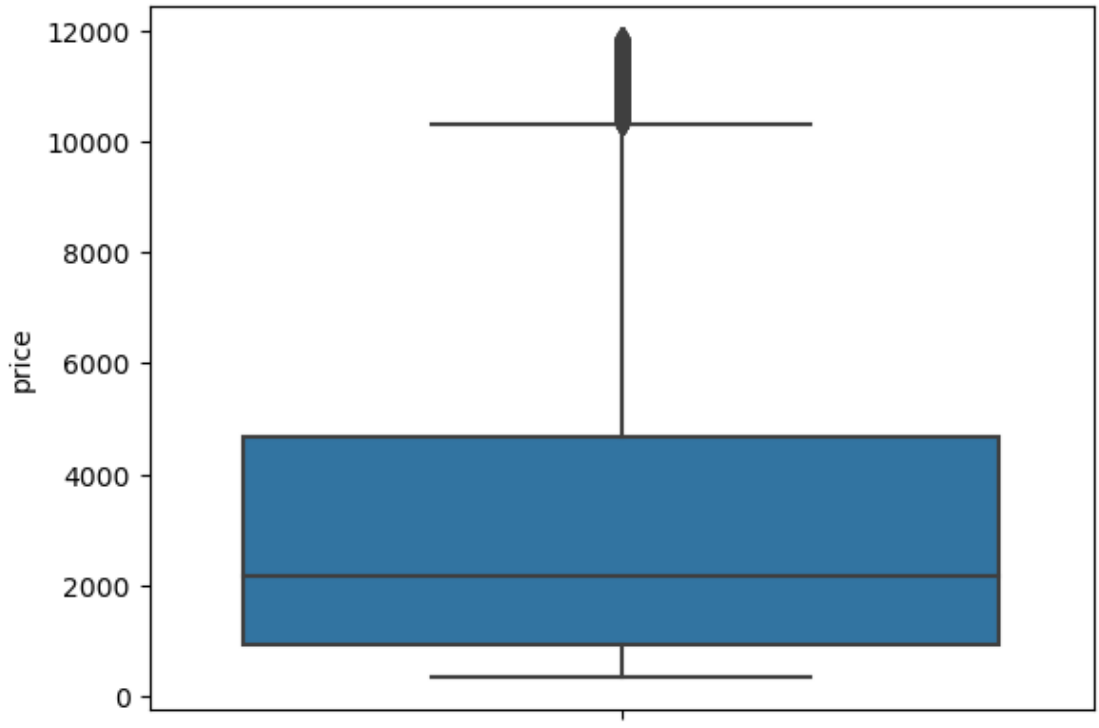
OUTLIER TREATMENT

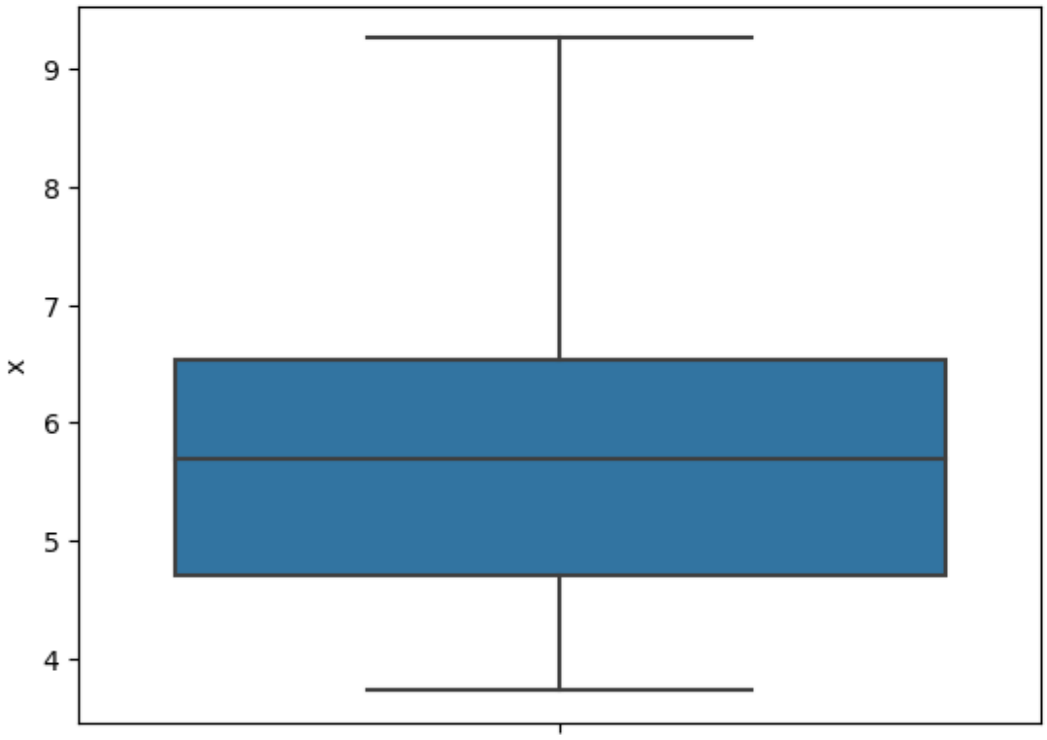
```
def outlier_limit(col):
    Q3,Q1=np.nanpercentile (col,[75,25])
    IQR=Q3-Q1
    UL=Q3+1.5*IQR
    LL=Q1-1.5*IQR
    return UL,LL

for column in data.columns:
    if data[column].dtype!='object':
        UL,LL=outlier_limit(data[column])
        data[column]=np.where((data[column]>UL)|
(data[column]<LL),np.nan,data[column])

for i in data.columns:
    if data[i].dtype!='object':
        sns.boxplot(y=data[i])
        plt.show();
```

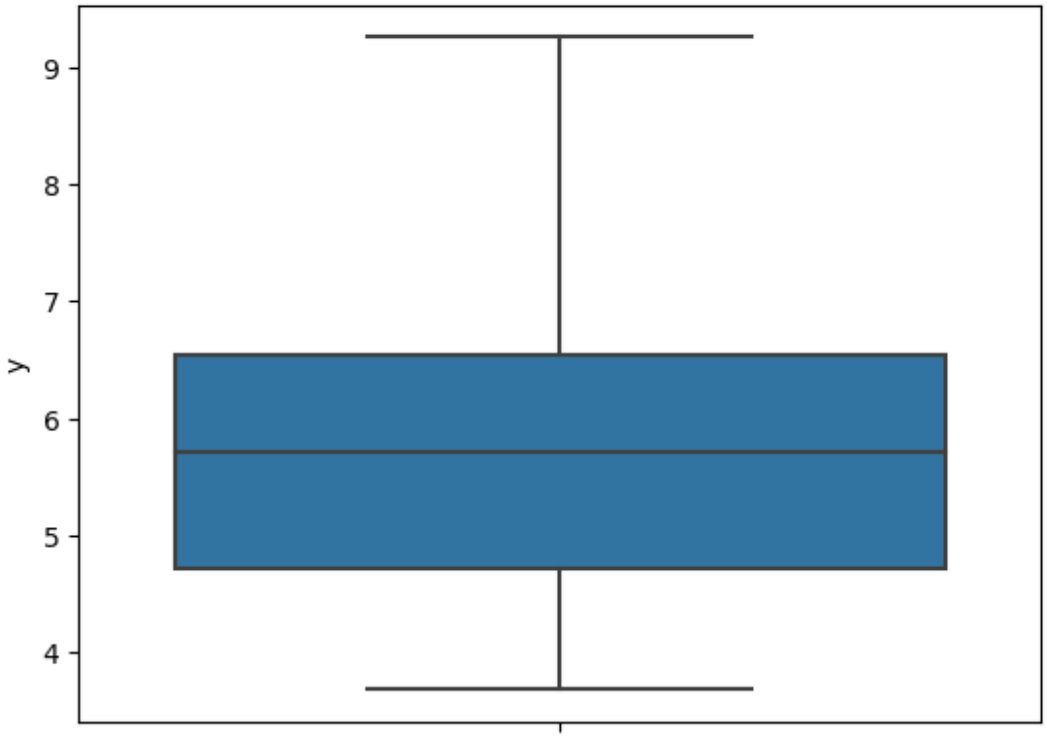

carat
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25

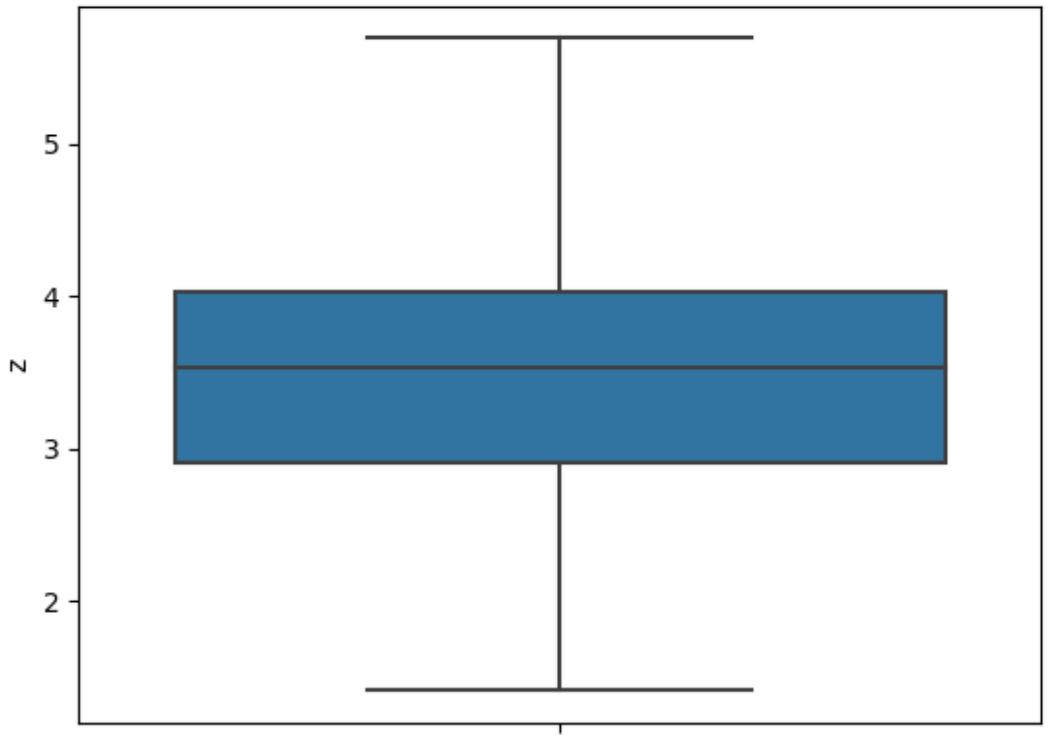

depth
64
63
62
61
60
59


64
62
60
58
56
54
52
table


12000
10000
8000
6000
4000
2000
0
price


x
9
8
7
6
5
4


y
9
8
7
6
5
4



LABEL ENCODING THE CATEGORICAL COLUMN

```
cols=["cut","color","clarity"]
le=LabelEncoder()
data[cols]=data[cols].apply(le.fit_transform)

data.head()
```

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	2	1	3	61.5	55.0	326.0	3.95	3.98	2.43
1	0.21	3	1	2	59.8	61.0	326.0	3.89	3.84	2.31
2	0.23	1	1	4	NaN	NaN	327.0	4.05	4.07	2.31
3	0.29	3	5	5	62.4	58.0	334.0	4.20	4.23	2.63
4	0.31	1	6	3	63.3	58.0	335.0	4.34	4.35	2.75

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 53794 entries, 0 to 53939
Data columns (total 10 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   carat    51921 non-null  float64
 1   cut      53794 non-null  int32
 2   color    53794 non-null  int32
 3   clarity  53794 non-null  int32
 4   depth    51269 non-null  float64
 5   table    53190 non-null  float64
 6   price    50271 non-null  float64
```

```
 7   x        53763 non-null  float64
 8   y        53766 non-null  float64
 9   z        53746 non-null  float64
dtypes: float64(7), int32(3)
memory usage: 5.9 MB
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 53794 entries, 0 to 53939
Data columns (total 10 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   carat    51921 non-null  float64
 1   cut      53794 non-null  int32
 2   color    53794 non-null  int32
 3   clarity  53794 non-null  int32
 4   depth    51269 non-null  float64
 5   table    53190 non-null  float64
 6   price    50271 non-null  float64
 7   x        53763 non-null  float64
 8   y        53766 non-null  float64
 9   z        53746 non-null  float64
dtypes: float64(7), int32(3)
memory usage: 5.9 MB
```

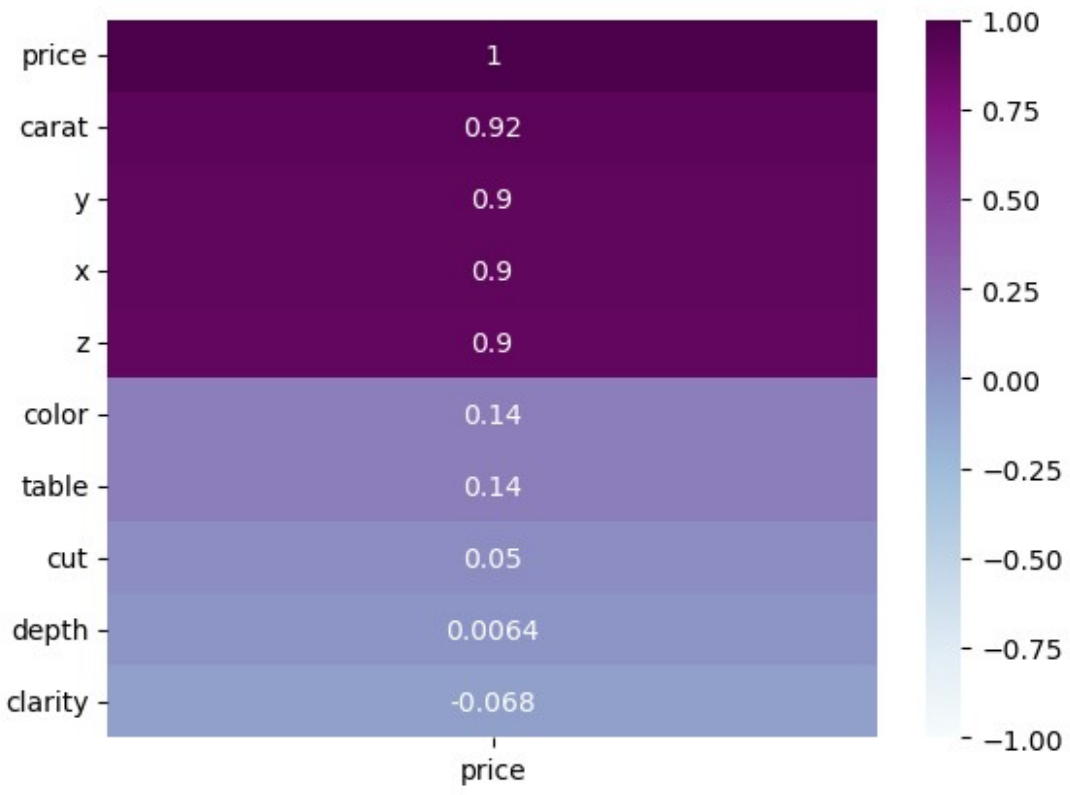
```
data.dropna(inplace=True)
```

```
data.columns
```

```
Index(['carat', 'cut', 'color', 'clarity', 'depth', 'table', 'price',
'x', 'y',
       'z'],
      dtype='object')
```

CHECKING THE CORRELATION WITH THE TARGET

```
sns.heatmap(data.corr()
[['price']].sort_values(by='price',ascending=False),vmin=-
1,vmax=1,annot=True,cmap="BuPu");
```




```
x=data[["y","carat",'x','y','z']].values
y=data[["price"]].values
```

FEATURE SCALING

```
sc = StandardScaler()

x=sc.fit_transform(x)

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,rando
m_state=0)
```

CROSS VALIDATION

```
models={
        'LinearRegression':LinearRegression(),
        "Ridge":Ridge(),
       "LinearSVR":LinearSVR(),
       "DecisionTreeRegressor":DecisionTreeRegressor(),
       "GradientBoostingRegressor":GradientBoostingRegressor(),
       "AdaBoostRegressor":AdaBoostRegressor(),
        "RandomForestRegressor":RandomForestRegressor()
}

for name,model in models.items():
    scores=cross_val_score(model, x,y,
scoring="neg_mean_squared_error",cv=10,n_jobs=-1)
    print("cross validation model:{}".format(name))
```

```
    rmse=np.sqrt(-scores)
    rmse_average=np.mean(rmse)
    print("AVERAGE RMSE:",rmse_average)
    print("*"*100)
```

```
cross validation model:LinearRegression
AVERAGE RMSE: 949.7403531965407
****************************************************************************
****************************
cross validation model:Ridge
AVERAGE RMSE: 949.8009020535379
****************************************************************************
****************************
cross validation model:LinearSVR
AVERAGE RMSE: 950.9088286214558
****************************************************************************
****************************
cross validation model:DecisionTreeRegressor
AVERAGE RMSE: 1351.0643057838838
****************************************************************************
****************************
cross validation model:GradientBoostingRegressor
AVERAGE RMSE: 1032.018693359752
****************************************************************************
****************************
cross validation model:AdaBoostRegressor
AVERAGE RMSE: 1111.4924122281423
****************************************************************************
****************************
cross validation model:RandomForestRegressor
AVERAGE RMSE: 1099.123386025782
****************************************************************************
****************************
```

MODEL BUILDING

```
model=LinearRegression()

model.fit(x_train,y_train)

LinearRegression()

y_pred=model.predict(x)
```

PREDICTING Y

```
OUTPUT=pd.DataFrame(zip(y,y_pred),columns=("ACTUAL","PREDICTED"),dtype
=float)
OUTPUT.head()
```

```
   ACTUAL   PREDICTED
0   326.0  341.388199
1   326.0  203.164865
```

```
2   334.0  542.645392
3   335.0  455.116691
4   336.0  321.404602
```

SCATTER PLOT

```
plt.figure(figsize=(18,6))
plt.title('ACTUAL VS PREDICTED')
sns.scatterplot(data=OUTPUT,x="ACTUAL",y="PREDICTED");
```