Predicting diamond prices using KNN regression

Import libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, r2_score
```

Reading the data

In [2]:
```
df = pd.read_csv("diamonds.csv")
print(df.shape)
df
```

```
(53940, 11)
```

Out[2]:

	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
...	...	...	...	...	...	...	...	...	...	...	...
53935	53936	0.72	Ideal	D	SI1	60.8	57.0	2757	5.75	5.76	3.50
53936	53937	0.72	Good	D	SI1	63.1	55.0	2757	5.69	5.75	3.61
53937	53938	0.70	Very Good	D	SI1	62.8	60.0	2757	5.66	5.68	3.56
53938	53939	0.86	Premium	H	SI2	61.0	58.0	2757	6.15	6.12	3.74
53939	53940	0.75	Ideal	D	SI2	62.2	55.0	2757	5.83	5.87	3.64

53940 rows × 11 columns

Data Preprocessing

Dropping irrelevant columns

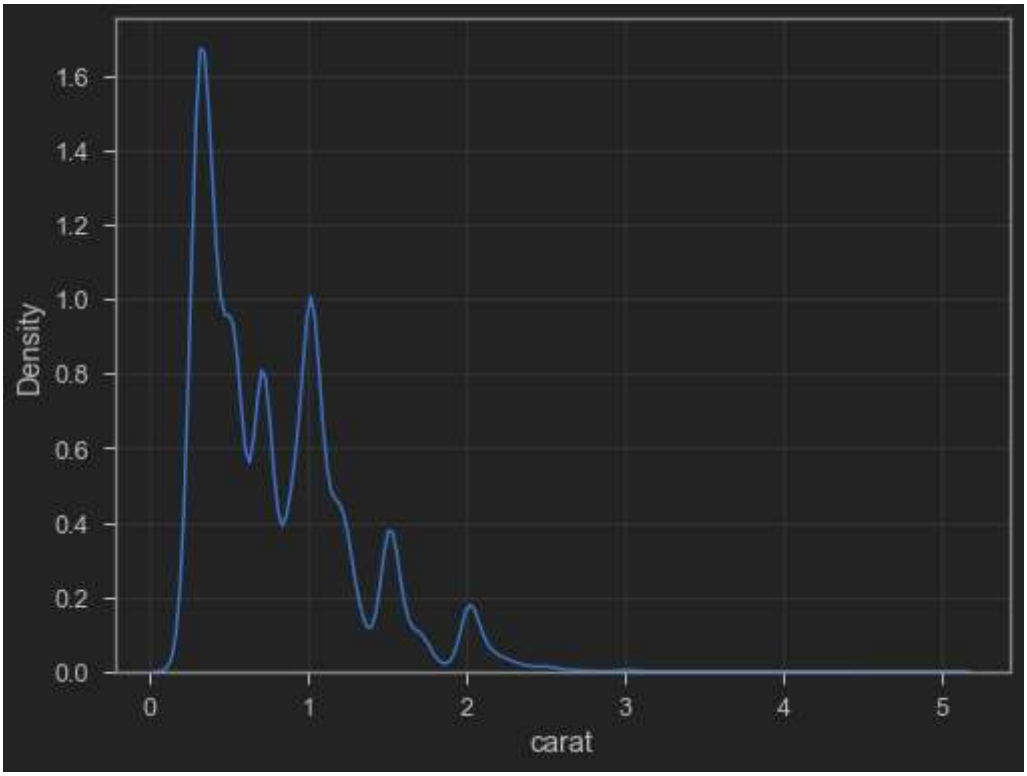
In [3]:
```
df = df.iloc[:, 1:]
df
```

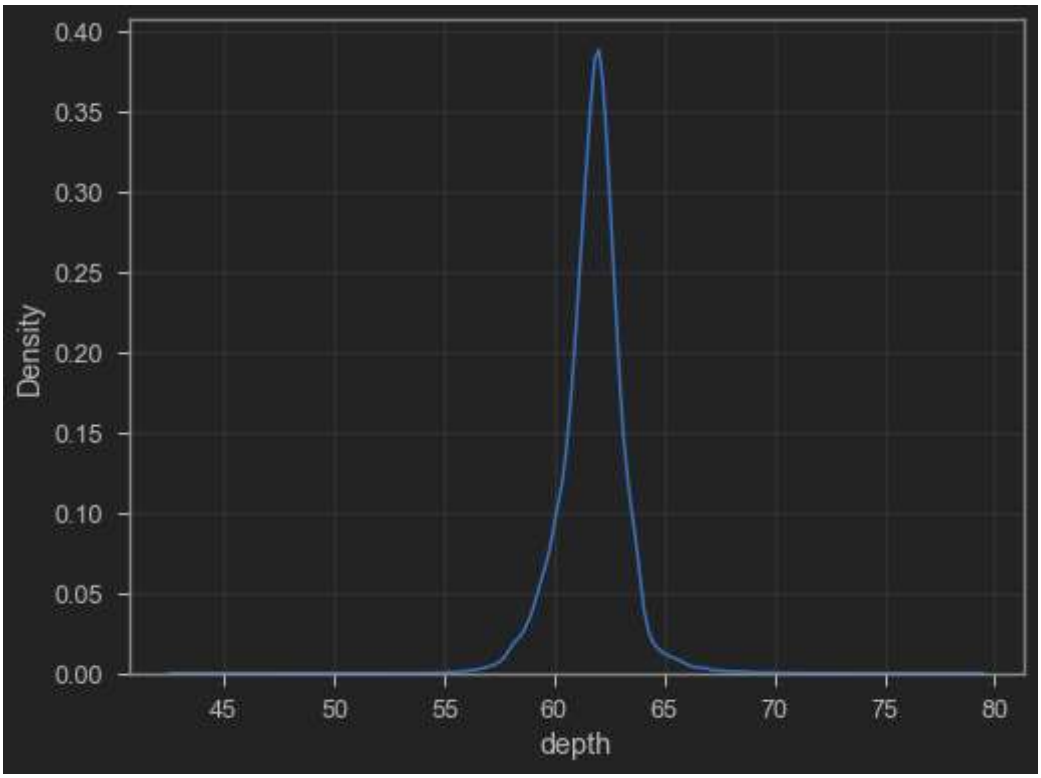
Out[3]:

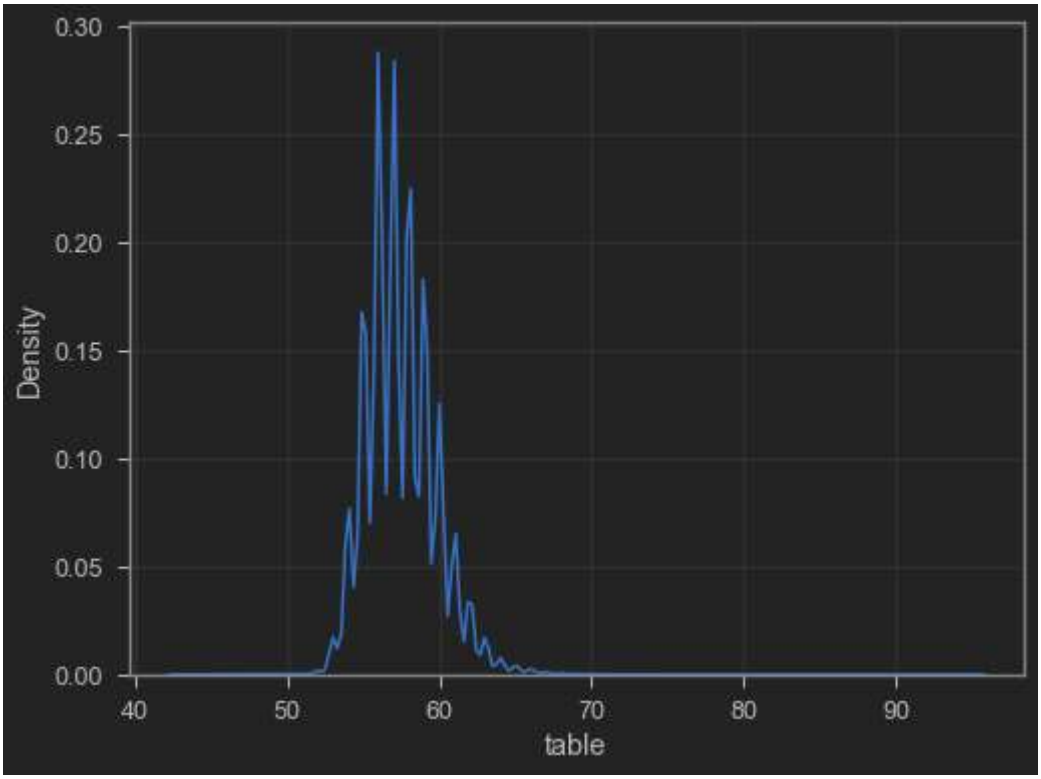
	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75
...	...	...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

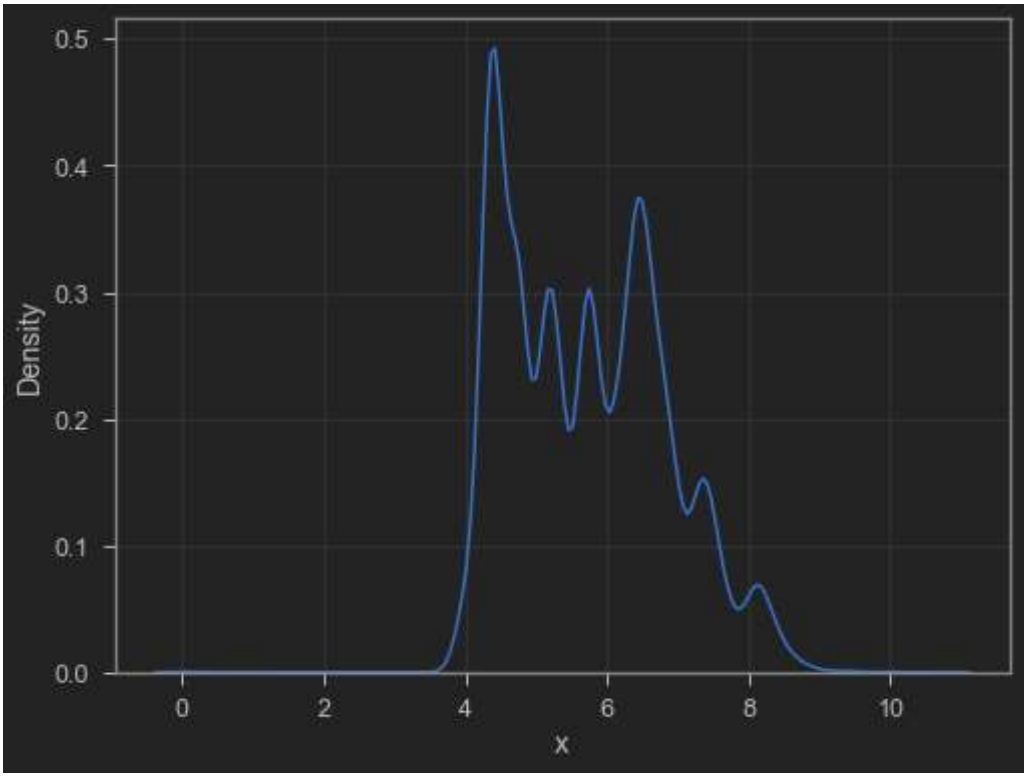
53940 rows × 10 columns

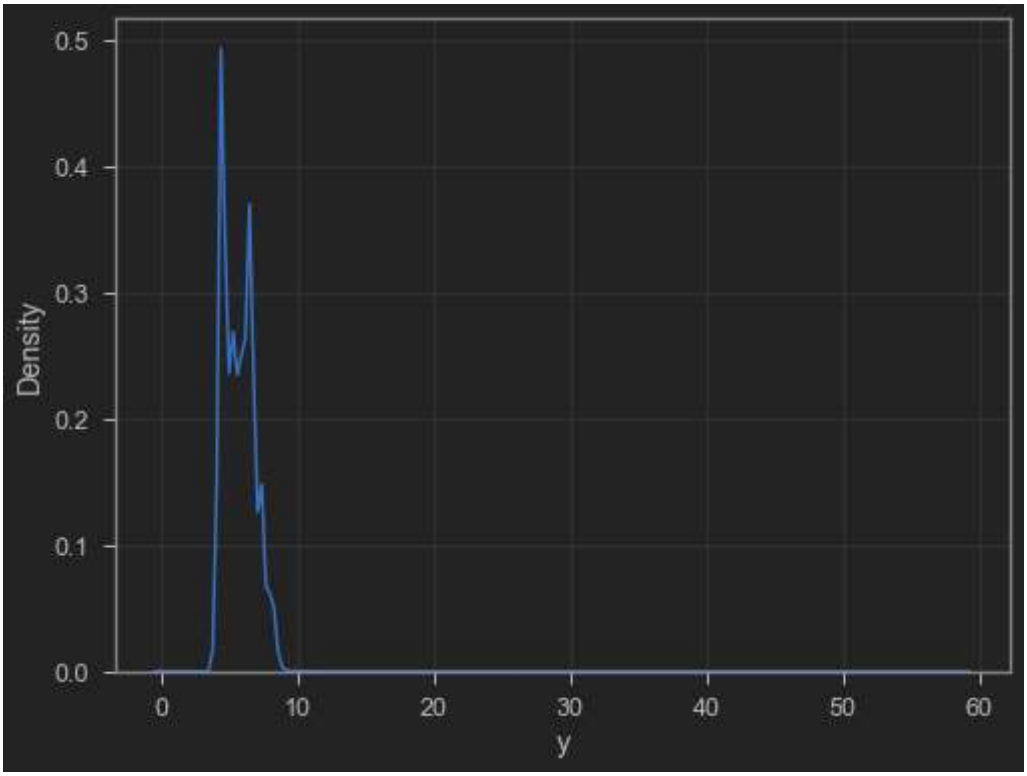
In [4]:
```
num_features = ['carat', 'depth', 'table', 'x', 'y', 'z']
for feature in num_features:
    plt.figure(figsize=(8, 6))
    sns.kdeplot(x=feature, data=df)
    plt.show()
```

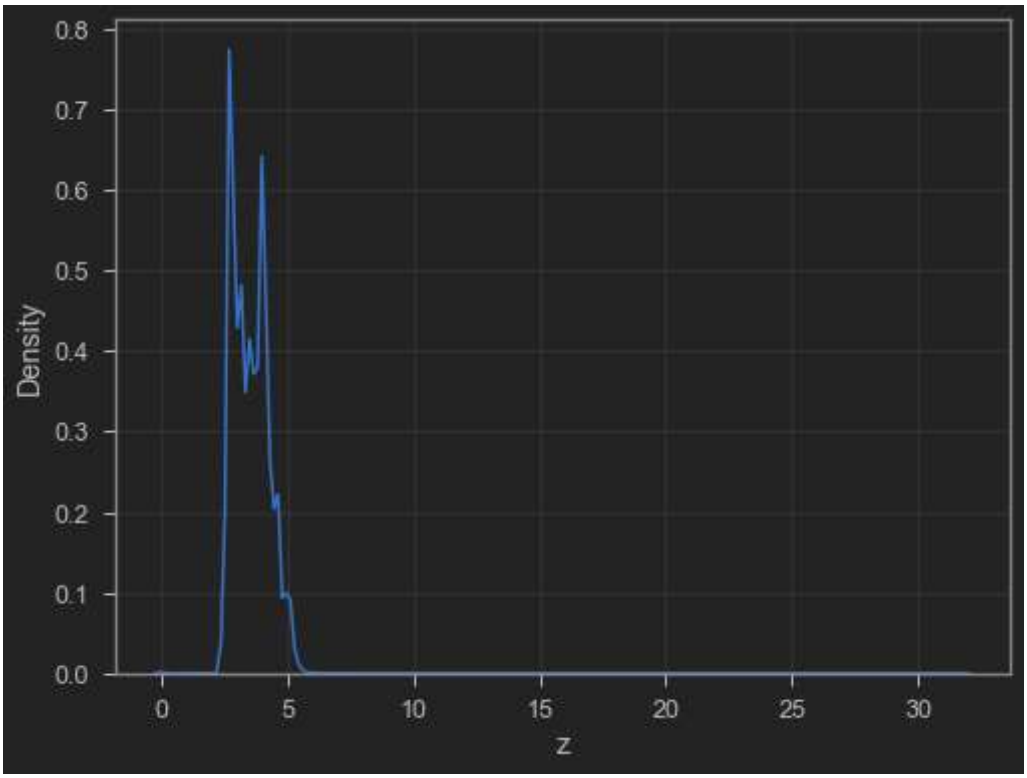

Density
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
5
carat


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

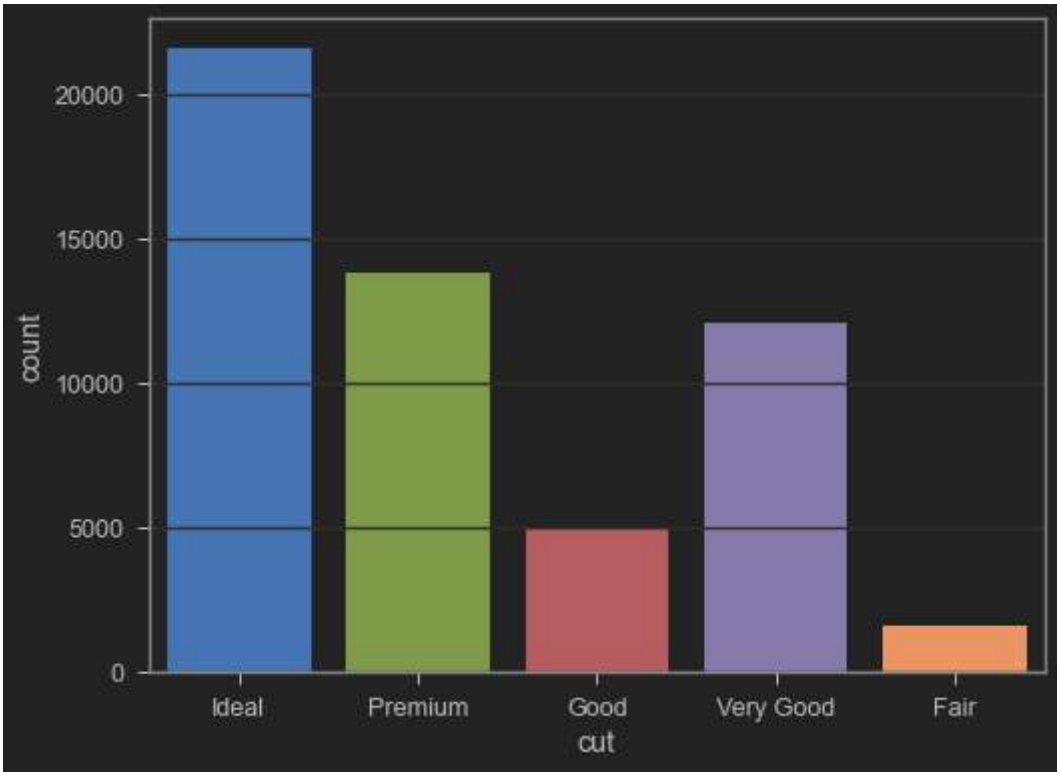

0.5
0.4
0.3
0.2
0.1
0.0
Density
0
2
4
6
8
10
x

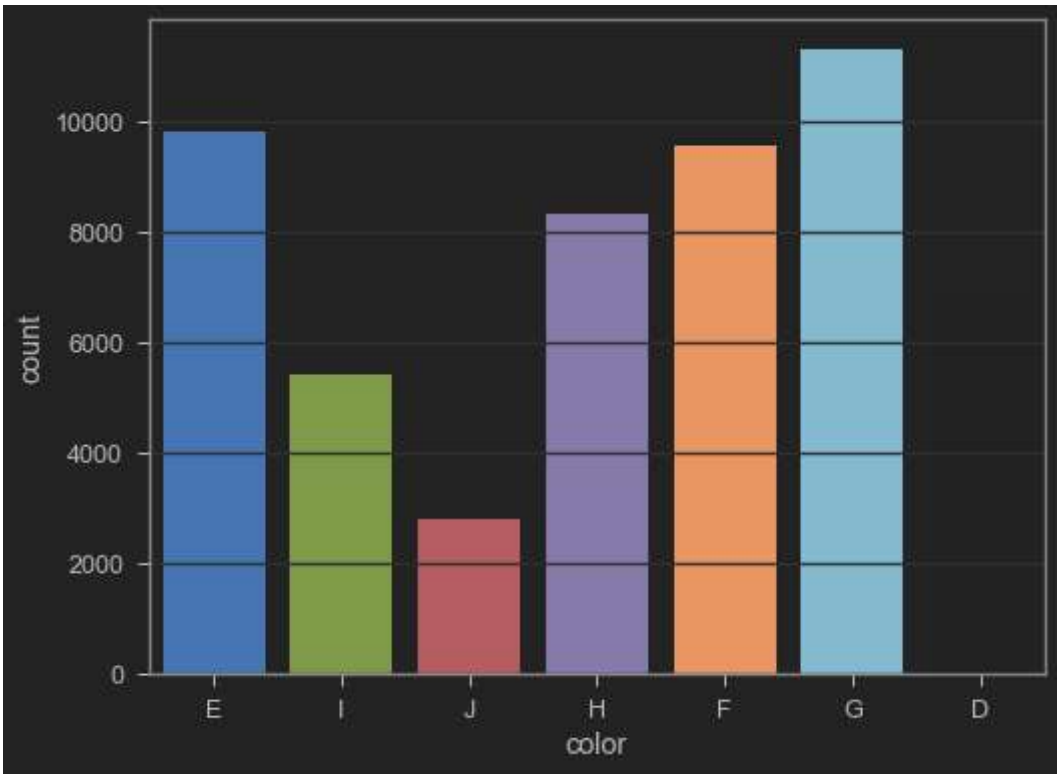


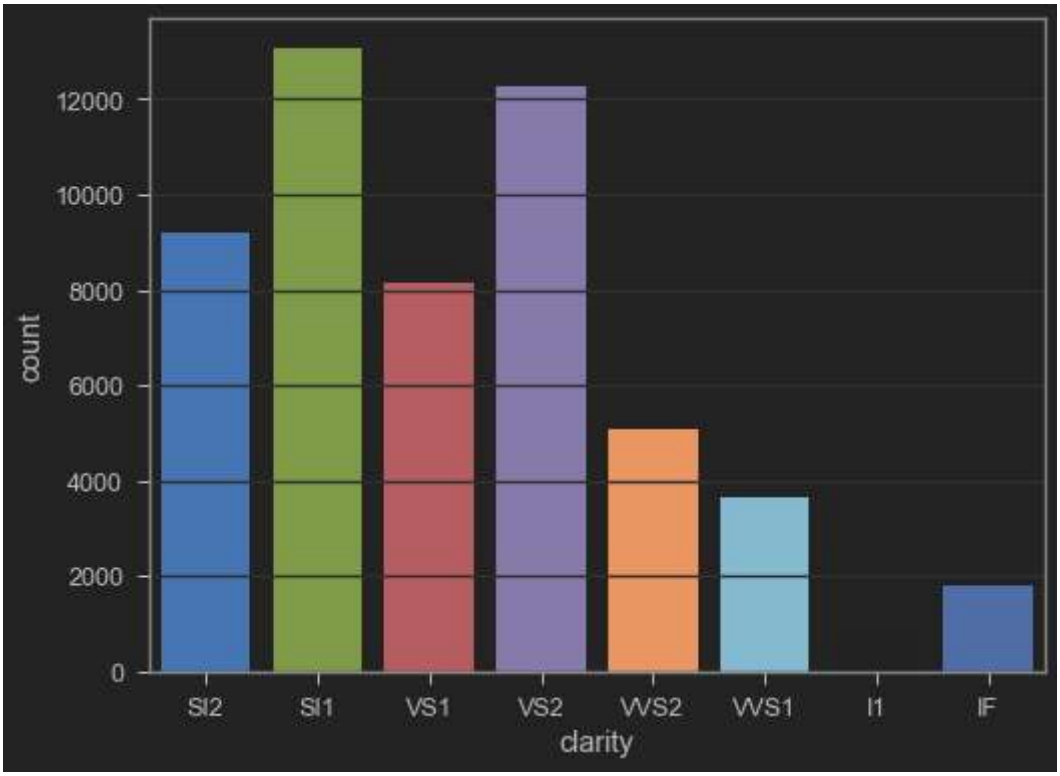


In [5]:

```
cat_features = ['cut', 'color', 'clarity']
for feature in cat_features:
    plt.figure(figsize=(8, 6))
    sns.countplot(x=feature, data=df)
    plt.show()
```

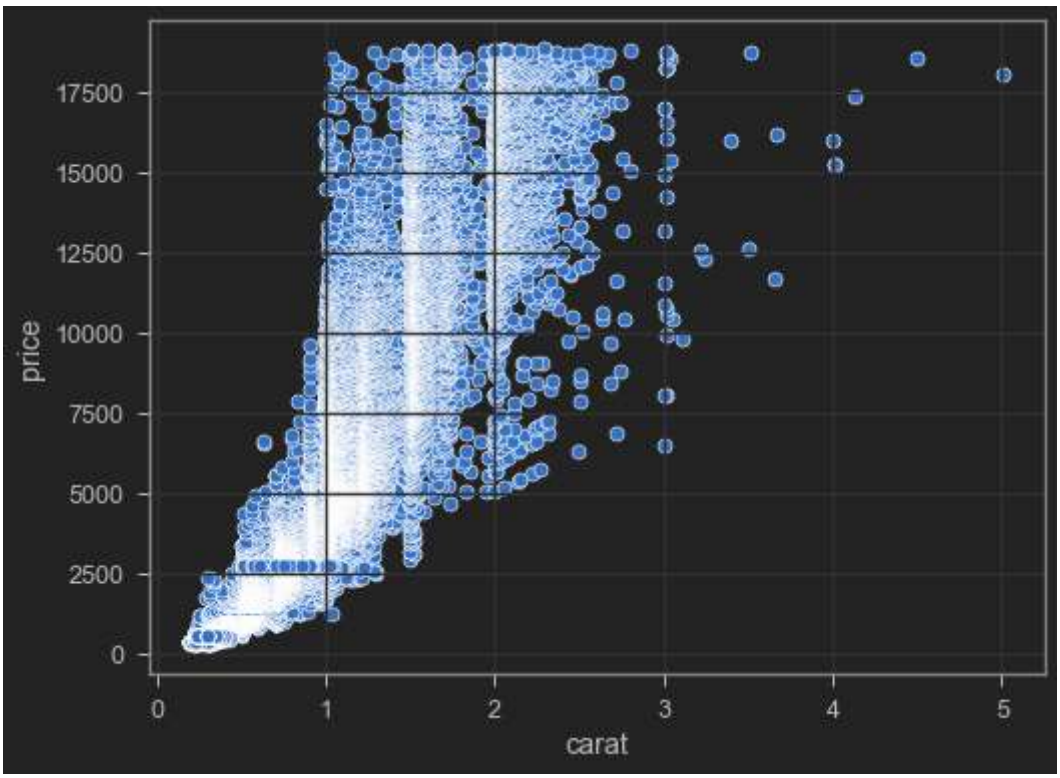

20000
15000
10000
5000
0
count
Ideal
Premium
Good
Very Good
Fair
cut

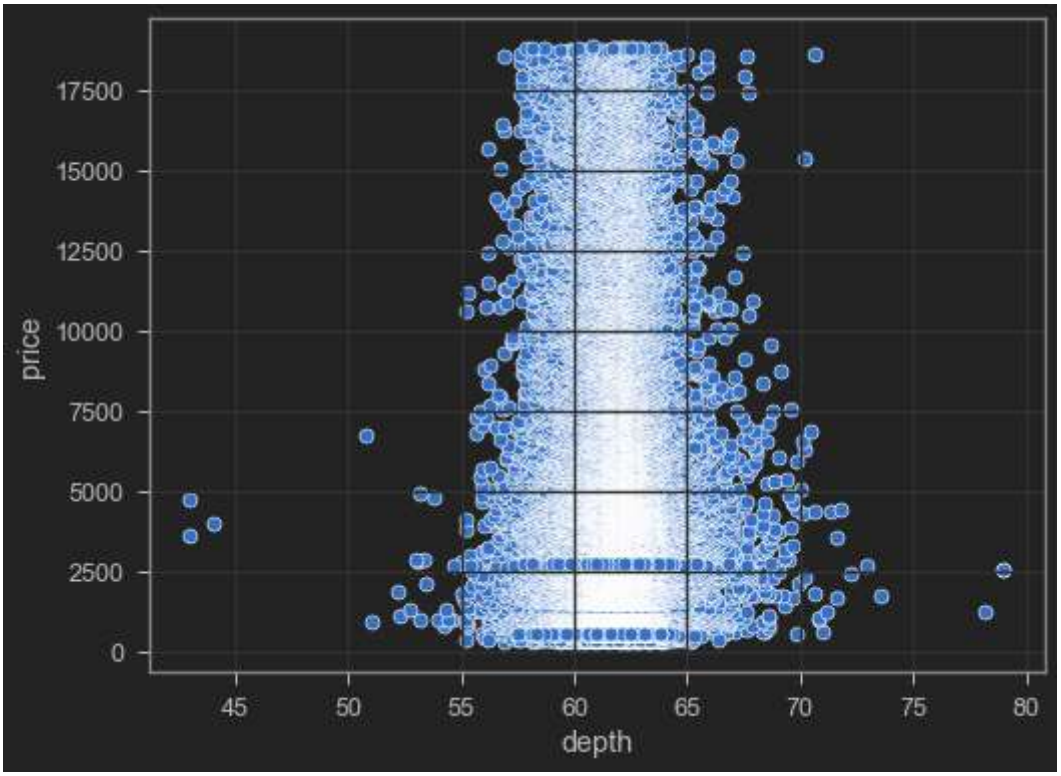

10000
8000
6000
4000
2000
0
count
E
I
J
H
F
G
D
color

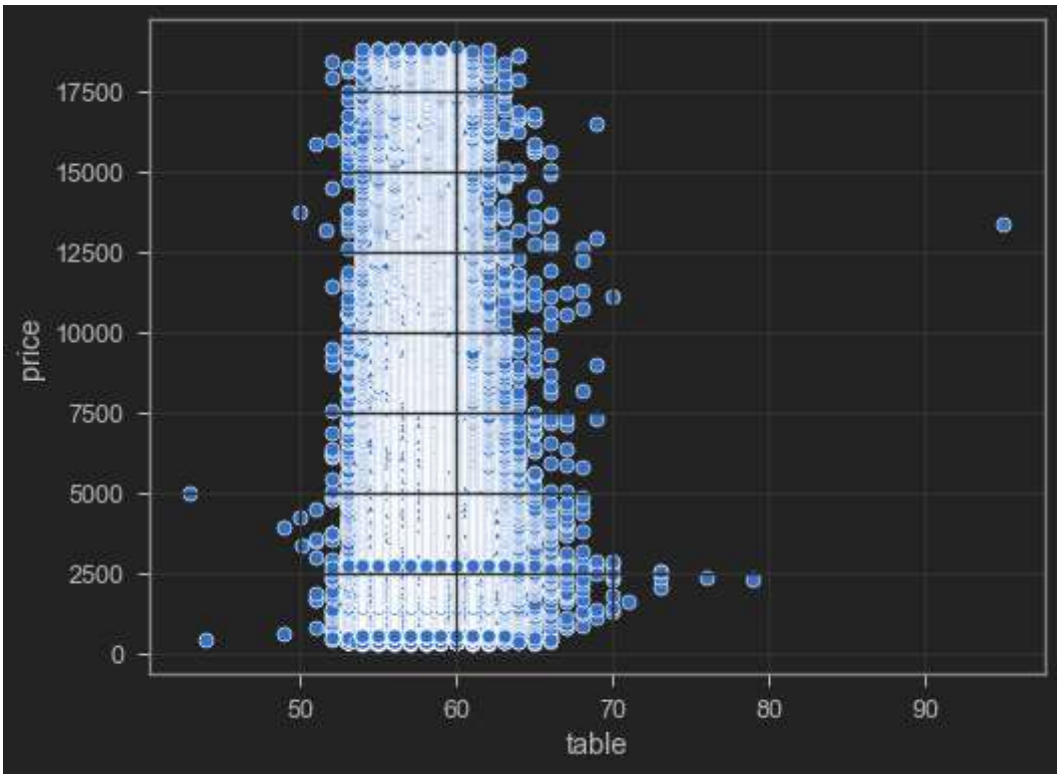


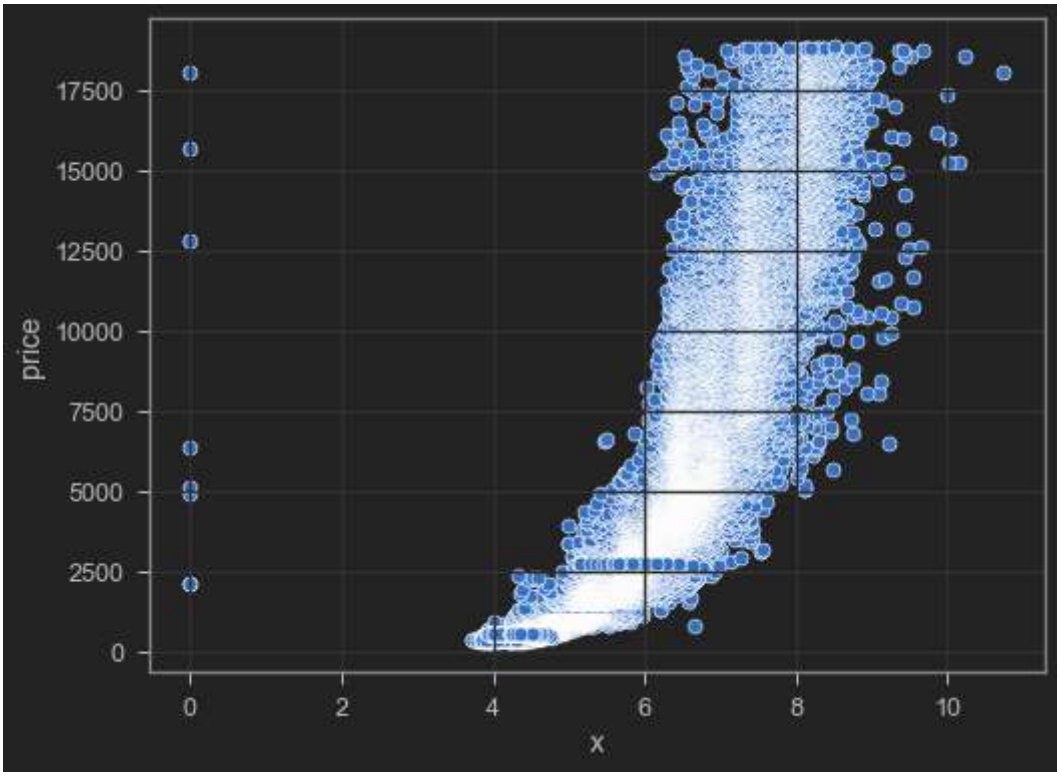
In [6]:

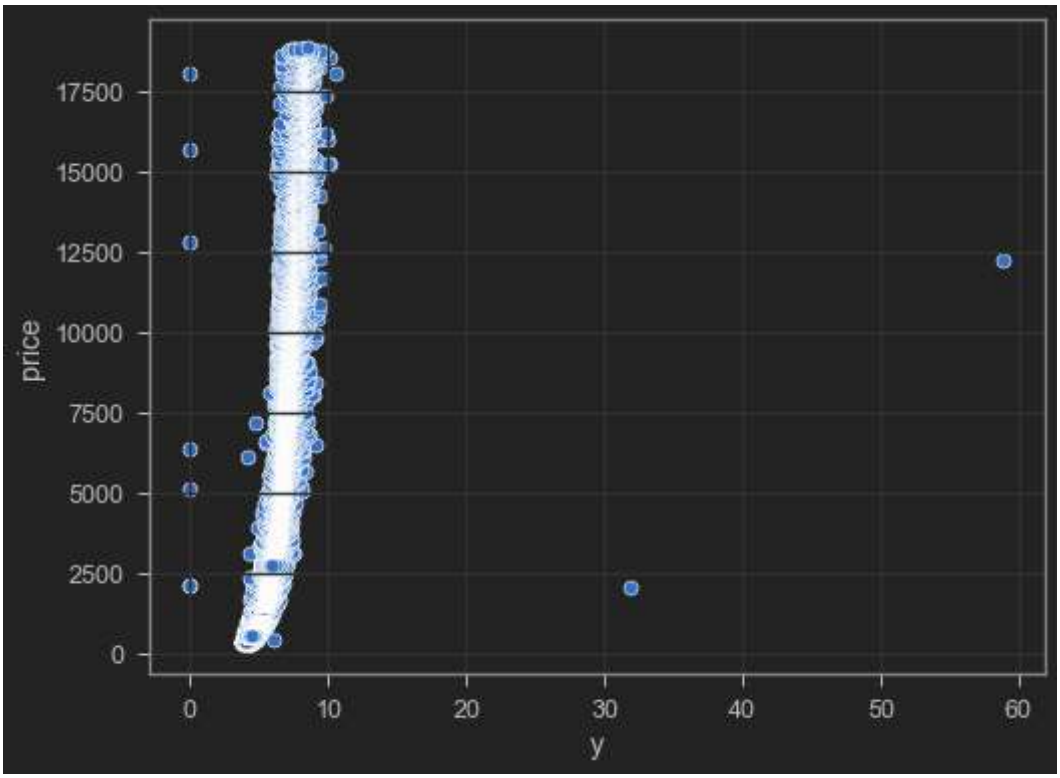
```
for feature in num_features:
    plt.figure(figsize=(8, 6))
    sns.scatterplot(x=feature, y='price', data=df)
    plt.show()
```

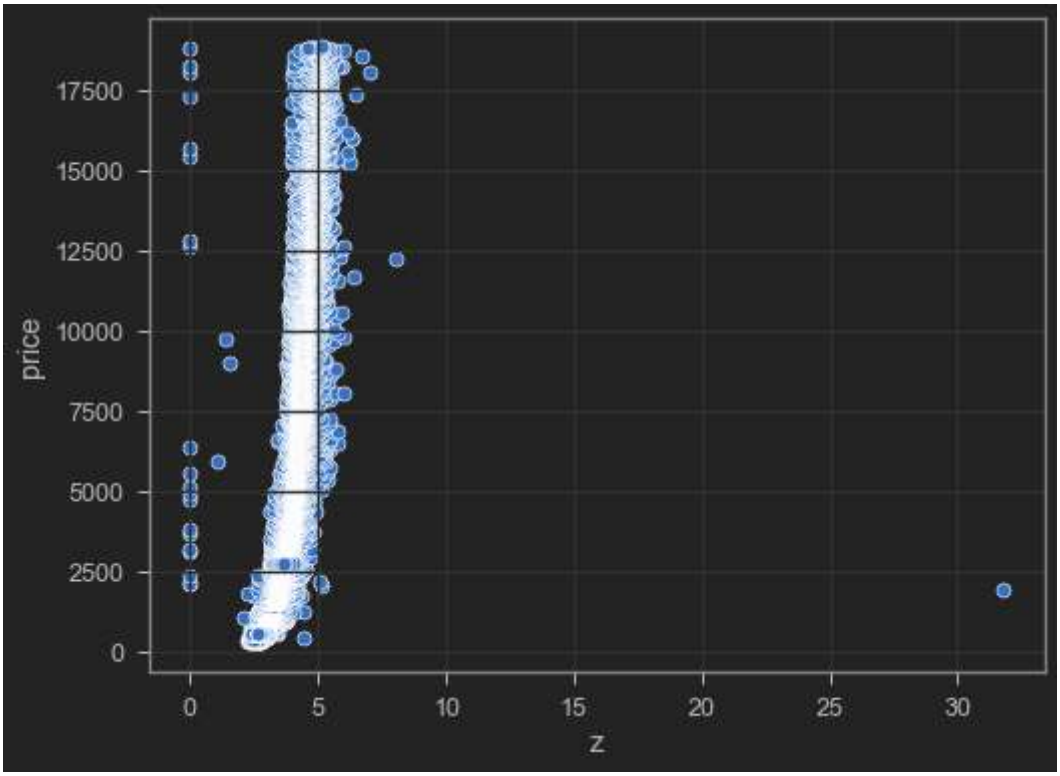



price
17500
15000
12500
10000
7500
5000
2500
0
45
50
55
60
65
70
75
80
depth


price
17500
15000
12500
10000
7500
5000
2500
0
50
60
70
80
90
table

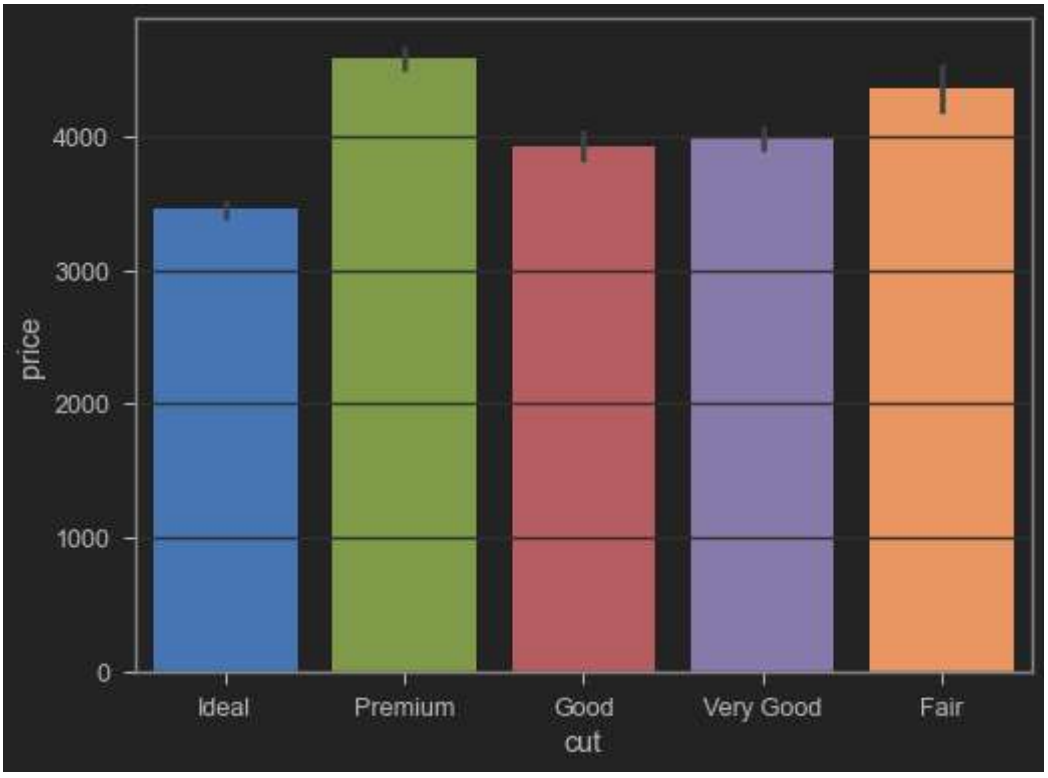

price
17500
15000
12500
10000
7500
5000
2500
0
0
2
4
6
8
10
x

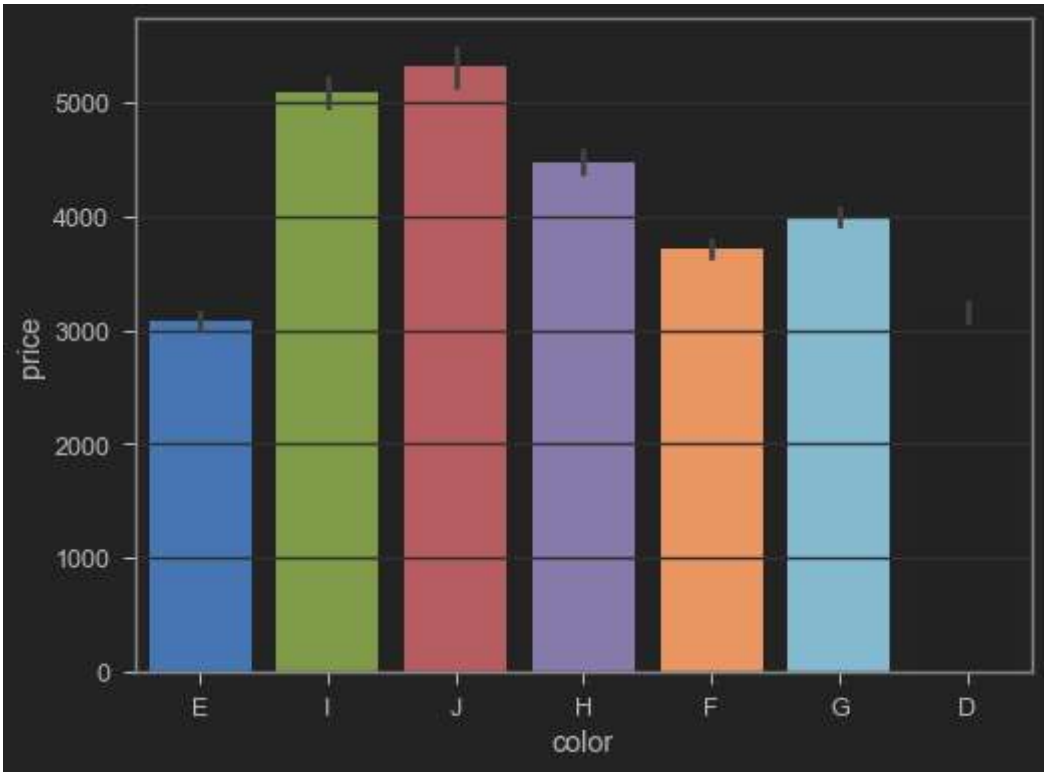

price
17500
15000
12500
10000
7500
5000
2500
0
0
10
20
30
40
50
60
y

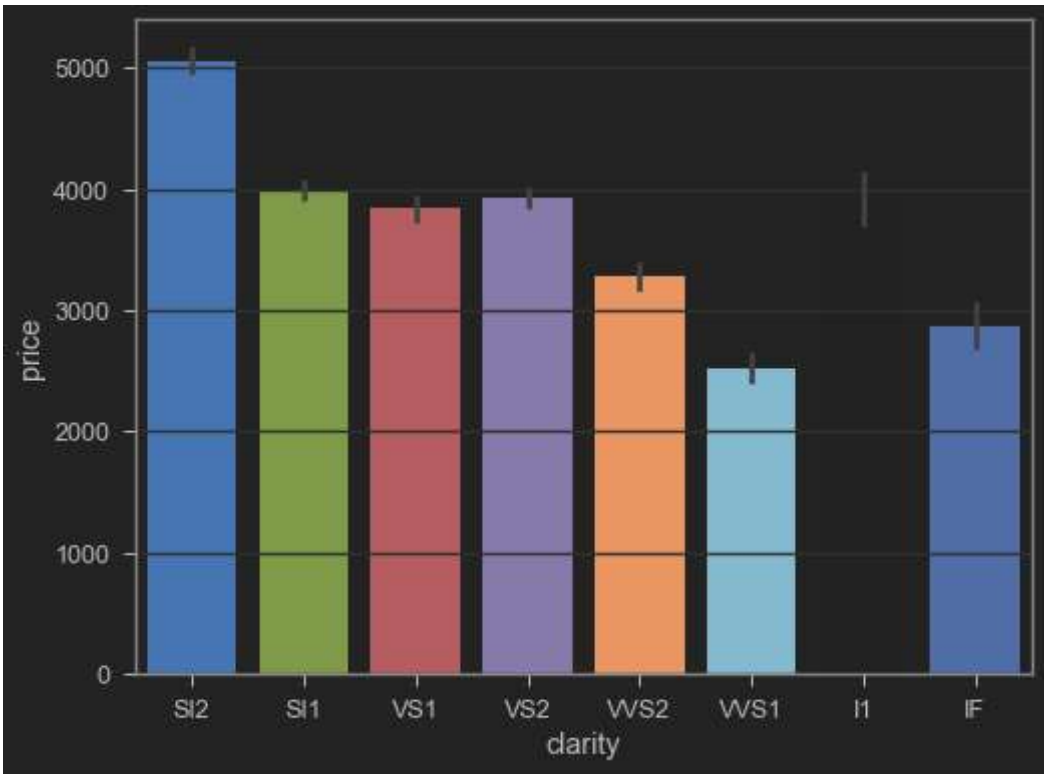


In [7]:

```
for feature in cat_features:
    plt.figure(figsize=(8, 6))
    sns.barplot(x=feature, y='price', data=df)
    plt.show()
```







One-Hot encoding for categorical columns

In [8]:
```
df = pd.get_dummies(df)
df
```

Out[8]:

	carat	depth	table	price	x	y	z	cut_Fair	cut_Good	cut_Ideal	...	color_I	color_J
0	0.23	61.5	55.0	326	3.95	3.98	2.43	0	0	1	...	0	0

	carat	depth	table	price	x	y	z	cut_Fair	cut_Good	cut_Ideal	...	color_I	color_J
1	0.21	59.8	61.0	326	3.89	3.84	2.31	0	0	0	...	0	0
2	0.23	56.9	65.0	327	4.05	4.07	2.31	0	1	0	...	0	0
3	0.29	62.4	58.0	334	4.20	4.23	2.63	0	0	0	...	1	0
4	0.31	63.3	58.0	335	4.34	4.35	2.75	0	1	0	...	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...
53935	0.72	60.8	57.0	2757	5.75	5.76	3.50	0	0	1	...	0	0
53936	0.72	63.1	55.0	2757	5.69	5.75	3.61	0	1	0	...	0	0
53937	0.70	62.8	60.0	2757	5.66	5.68	3.56	0	0	0	...	0	0
53938	0.86	61.0	58.0	2757	6.15	6.12	3.74	0	0	0	...	0	0
53939	0.75	62.2	55.0	2757	5.83	5.87	3.64	0	0	1	...	0	0

53940 rows × 27 columns

Data Splitting

In [9]:
```
X = df.drop('price', axis=1)
y = df['price']
```

In [10]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.33, random_state=42)
```

Modeling using KNN Regressor

In [11]:
```
knn = KNeighborsRegressor(n_neighbors=10)


knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)


score = knn.score(X_test, y_test)
score
```

Out[11]: 0.9388175223554891

Model Evaluation

In [12]:

```
print("Root Mean Squared error: ",
np.sqrt(mean_squared_error(y_test, y_pred)))
print("Correlation score: ", r2_score(y_test, y_pred))
```

```
Root Mean Squared error:  977.3582034889167
Correlation score:  0.9388175223554891
```

Root Mean Squared error: 977.3582034889167

Correlation score: 0.9388175223554891