Importing Libraries

- Introduction of Pythons libraries which is used in the given dataset
- Pandas : used as basic Level where it is manipulate and analyse the given dataset
- Numpy : used as basically numerical data for mathematical calculation
- Matplotlib : it is used for visualising the data
- Seaborn : it is advanced visulating libraries
- Scikit : Basically it used for modelling and evaluation the data

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.linear_model import LinearRegression,Ridge, Lasso
from sklearn.metrics import accuracy_score

from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder

from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from xgboost import XGBRegressor
```

Importing a dataset

In [2]:

```
df = pd.read_csv('diamonds.csv')
```

Get the attribute Information

In [3]:
```
df.head(3)
```

Out[3]:

	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

In [4]:
```
df.tail(4)
```

Out[4]:

	Unnamed: 0	carat	cut	color	clarity	depth	table	price	x	y	z
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

Shape of data

In [5]:
```
print('The total size of a diamond dataset :',df.shape)
```

```
The total size of a diamond dataset : (53940, 11)
```

In [6]:
```
(df.columns.tolist())
```

Out[6]:
```
['Unnamed: 0',
 'carat',
 'cut',
 'color',
 'clarity',
 'depth',
 'table',
 'price',
 'x',
 'y',
 'z']
```

List of Columns

- Unnamed: 0 -> it is useless
- Carat -> Carat is the unit of measurement for the physical weight of diamonds.
- Cut -> cut refers to how well-proportioned the dimensions of a diamond to create sparkle and brilliance
- Color -> color is a crucial aspect of your diamond's appearance
- Clarity -> a measure of the purity and rarity of the stone, graded by the visibility of these characteristics under 10-power magnification.
- Depth -> the distance in millimeters from its culet (bottom tip) to its table (flat top surface).
- Table -> table is the facet which can be seen when the stone is viewed face up
- Price -> price of a diamond
- x -> coordinates of x -axis
- y -> coordinates of y-axis
- z -> coordinates of z-axis

In [7]:
```
# drop useless columns
df.drop(df[['Unnamed: 0', 'x', 'y', 'z']], axis=1, inplace=True)
```

In [8]:
```
print('After removing useless columns :')

df.head(4)
```

```
After removing useless columns :
```

Out[8]:

	carat	cut	color	clarity	depth	table	price
0	0.23	Ideal	E	SI2	61.5	55.0	326
1	0.21	Premium	E	SI1	59.8	61.0	326
2	0.23	Good	E	VS1	56.9	65.0	327
3	0.29	Premium	I	VS2	62.4	58.0	334

In [9]:
```
df.info() # information of dataset
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53940 entries, 0 to 53939
Data columns (total 7 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  
dtypes: float64(3), int64(1), object(3)
memory usage: 2.9+ MB
```

In [10]:
```
print("Datatype of all columns:")
df.dtypes
```

```
Datatype of all columns:
```

Out[10]:
```
carat      float64
cut         object
color       object
clarity     object
depth      float64
table      float64
price        int64
dtype: object
```

Check it is null or not & any duplicated

In [11]:
```
df.isnull().sum()
```

Out[11]:
```
carat      0
cut        0
color      0
clarity    0
depth      0
table      0
price      0
dtype: int64
```

In [12]:
```
df.duplicated().sum()
```

Out[12]:
```
803
```

In [13]:
```
df.drop_duplicates(inplace = True)
```

In [14]:
```
print('After remove duplicated value:',df.duplicated().sum())
```

```
After remove duplicated value: 0
```

Summary of Diamond Dataset

In [15]:
```
df.describe()
```

Out[15]:

	carat	depth	table	price
count	53137.000000	53137.000000	53137.000000	53137.000000
mean	0.802930	61.745185	57.471263	3967.827258
std	0.473626	1.436319	2.237208	3998.021972
min	0.200000	43.000000	43.000000	326.000000
25%	0.400000	61.000000	56.000000	967.000000
50%	0.710000	61.800000	57.000000	2451.000000
75%	1.050000	62.500000	59.000000	5376.000000
max	5.010000	79.000000	95.000000	18823.000000

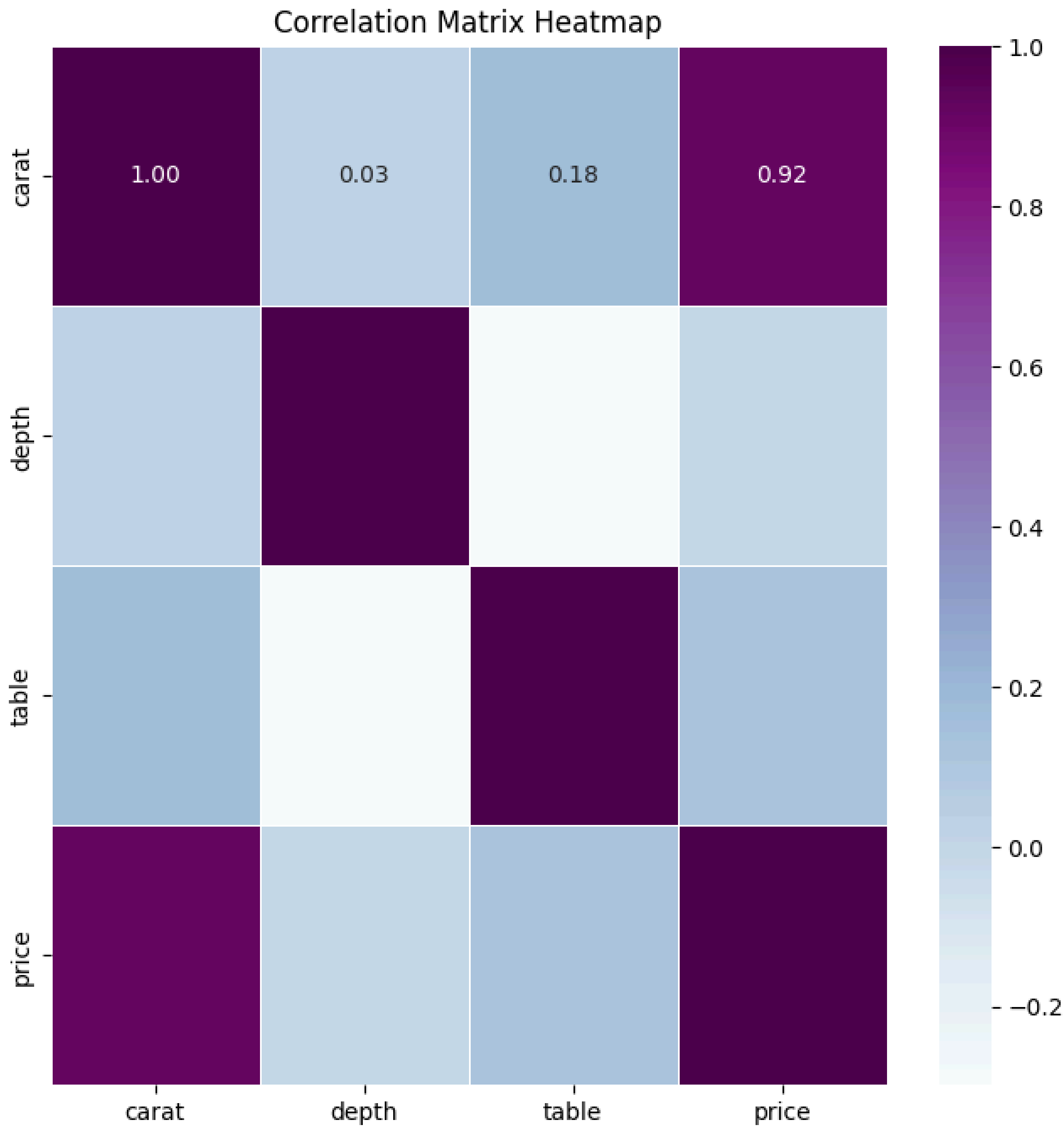
In [16]:
```
df.sample(10) # print random 10 samples of a dataset
```

Out[16]:

	carat	cut	color	clarity	depth	table	price
50789	0.70	Very Good	G	SI1	62.8	56.0	2304
50257	0.71	Premium	D	SI2	59.7	60.0	2235
35251	0.30	Very Good	G	IF	62.8	57.0	895
36845	0.37	Premium	F	VS2	60.6	58.0	957
41994	0.43	Premium	G	VVS1	61.3	57.0	1264
50179	0.64	Very Good	E	VS2	60.5	58.1	2222
31718	0.38	Good	G	VS2	58.8	62.0	771
15935	1.29	Ideal	I	SI1	61.9	56.0	6372
48798	0.71	Ideal	H	SI2	61.8	55.0	2024
52331	0.78	Ideal	G	SI1	61.0	57.0	2496

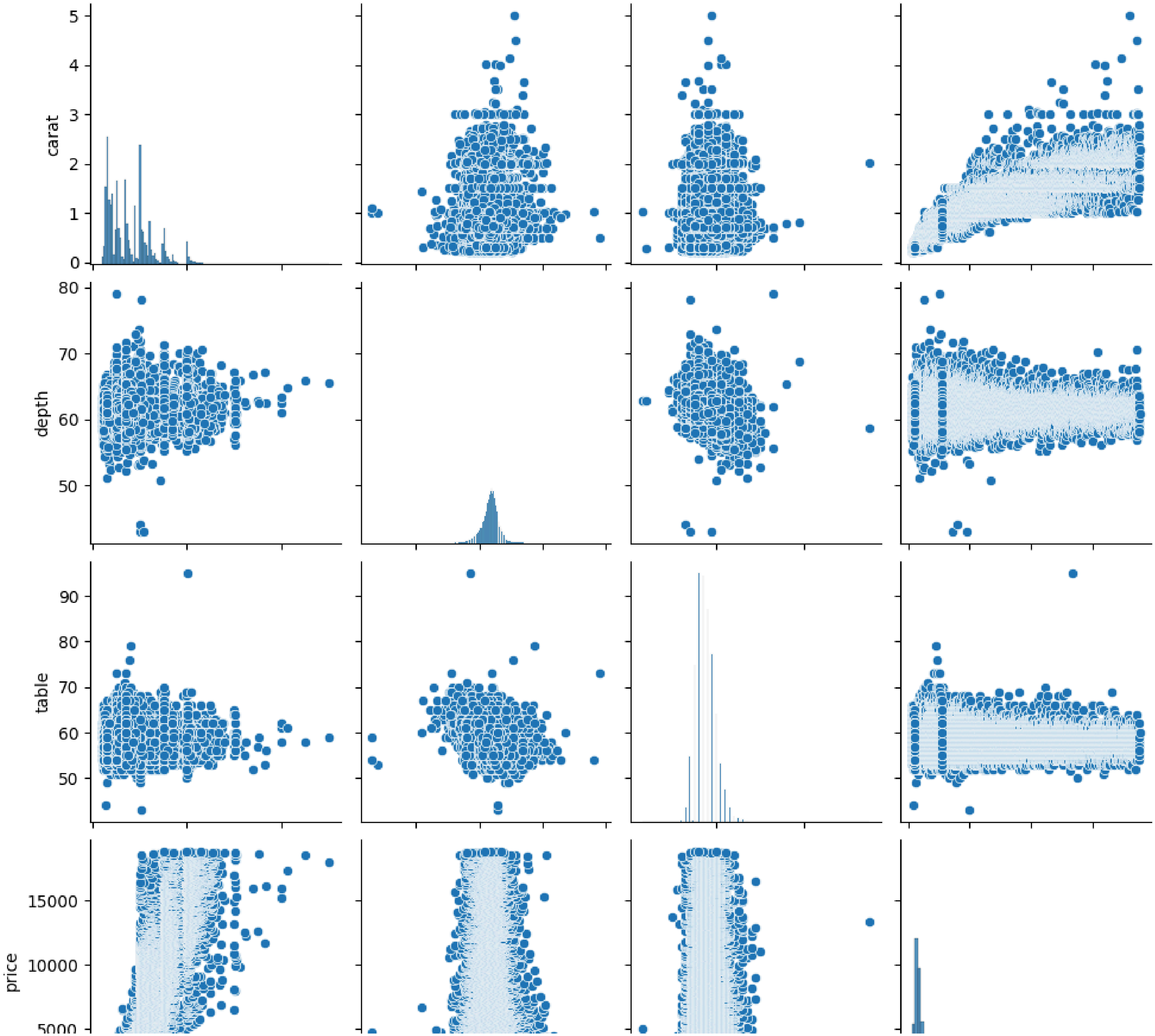
In [17]:

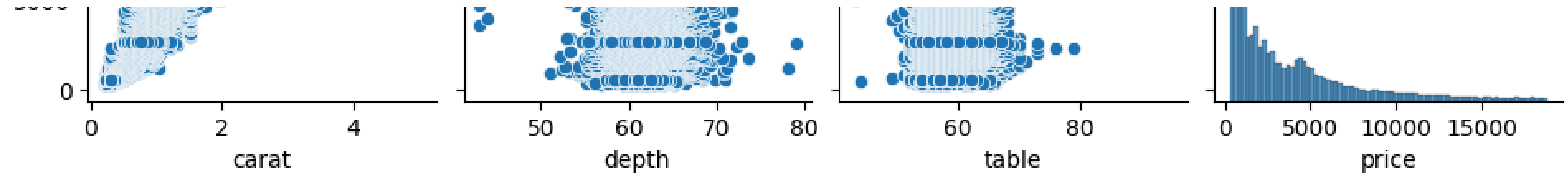
```
numeric_columns = df[['carat', 'depth','table','price']]
correlation_matrix = numeric_columns.corr()

plt.figure(figsize=(8,8))
sns.heatmap(correlation_matrix, annot=True, cmap='BuPu', fmt=".2f",
            linewidths = 0.5)
plt.title("Correlation Matrix Heatmap")
plt.show()
```


Correlation Matrix Heatmap
carat
depth
table
price
1.00
0.03
0.18
0.92
carat
depth
table
price
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

In [18]:
```
sns.pairplot(data = df)
```

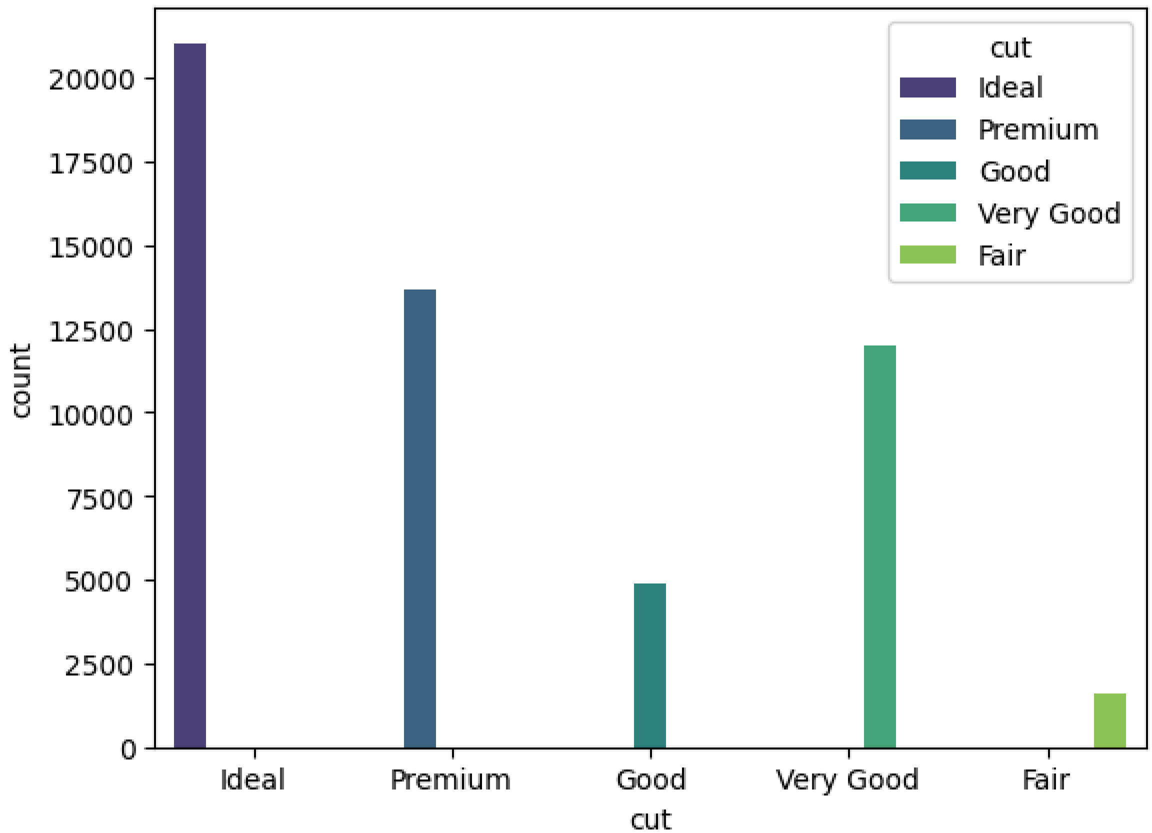
Out[18]:
```
<seaborn.axisgrid.PairGrid at 0x2ce119f60b0>
```


carat
depth
table
price
5
4
3
2
1
0
80
70
60
50
90
80
70
60
50
15000
10000
5000



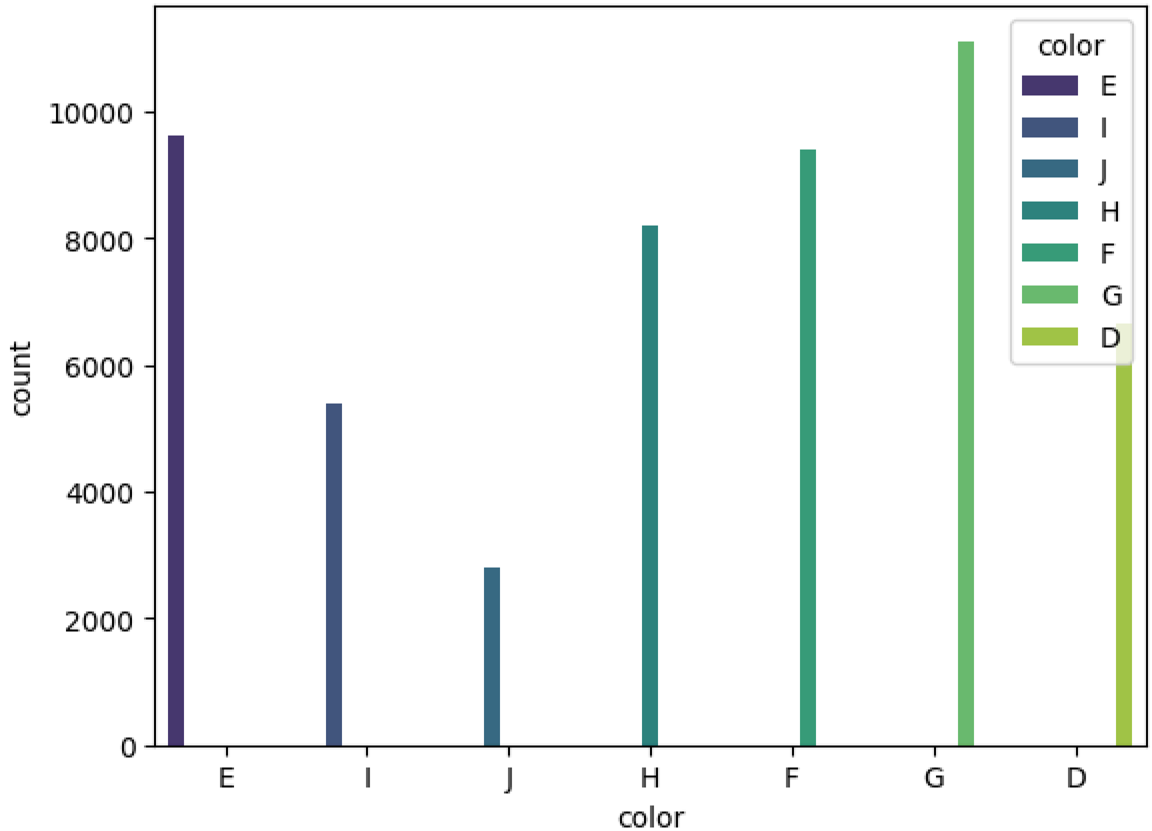
In [19]:
```
df['cut'].value_counts()

sns.countplot(x=df.cut, hue=df.cut, palette='viridis')
```

Out[19]: <Axes: xlabel='cut', ylabel='count'>



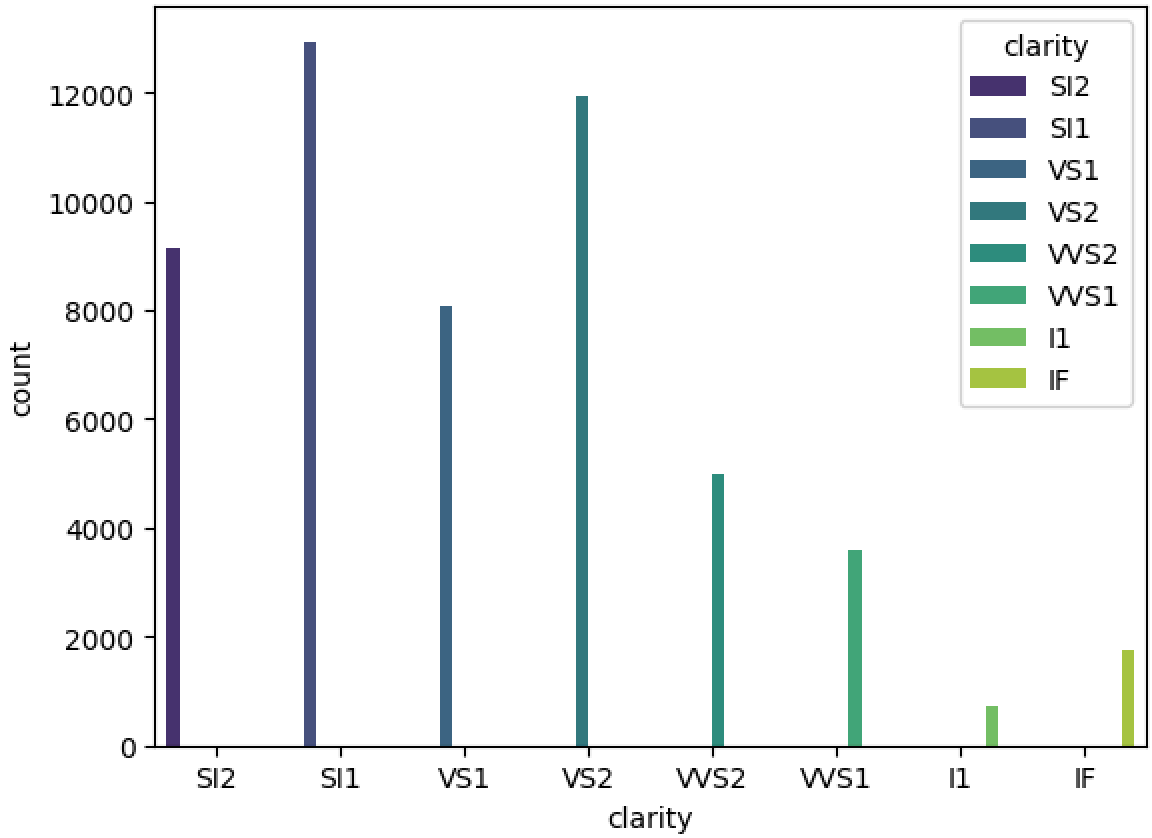
In [20]:
```
df['color'].value_counts()

sns.countplot(x=df.color, hue=df.color, palette='viridis')
```

Out[20]: <Axes: xlabel='color', ylabel='count'>



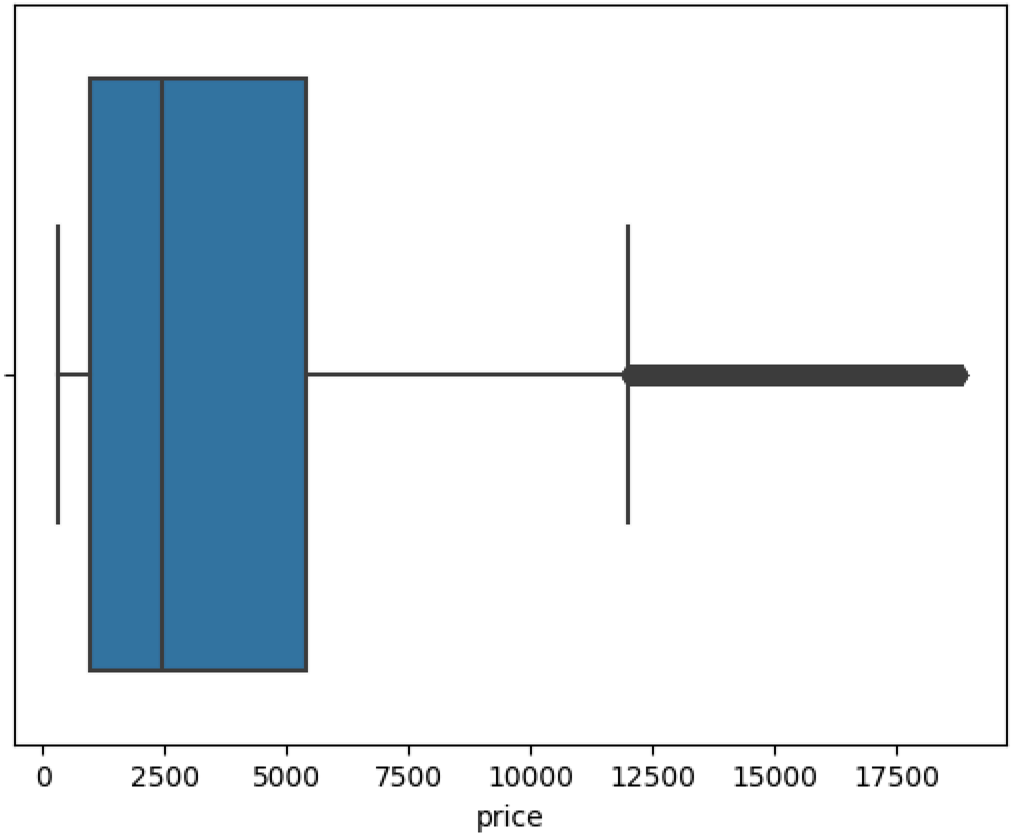
In [21]:
```
df['clarity'].value_counts()


sns.countplot(x=df.clarity, hue=df.clarity, palette='viridis')
```

Out[21]: <Axes: xlabel='clarity', ylabel='count'>



In [22]:
```
sns.boxplot(x=df['price'])
```

Out[22]: <Axes: xlabel='price'>

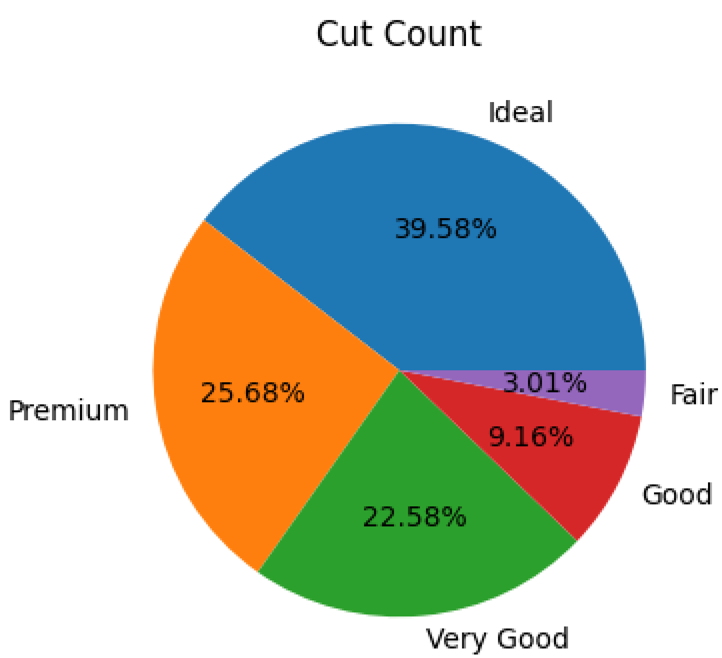


In [23]:
```
df.nunique()
```

Out[23]:
```
carat        273
cut            5
color          7
clarity        8
depth        184
table        127
price      11602
dtype: int64
```

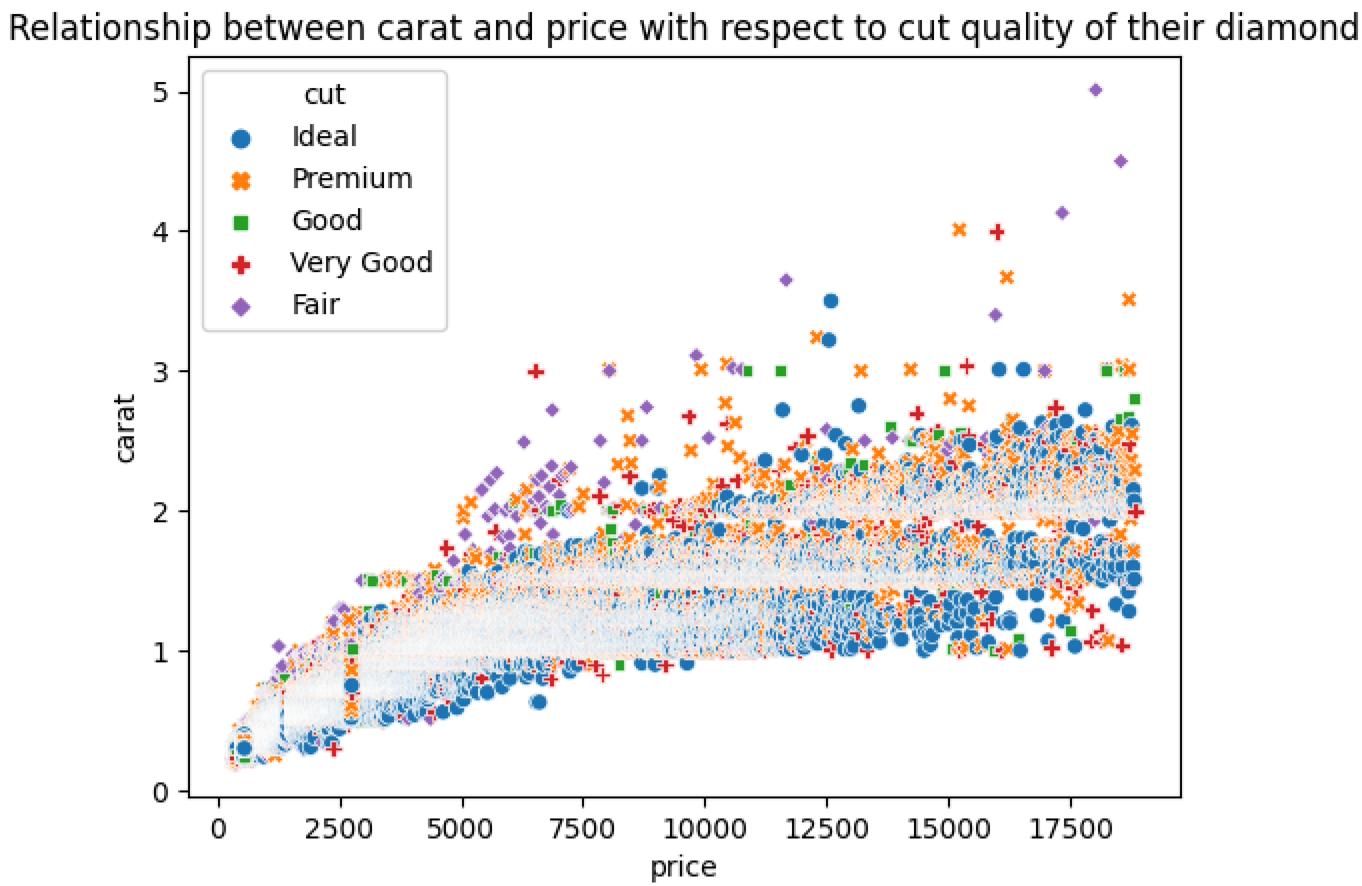
In [24]:
```
plt.figure(figsize=(5,4))
plt.pie(df.cut.value_counts().values, labels = df.cut.value_counts().index,
       autopct='%1.2f%%')
plt.title("Cut Count")
```

Out[24]:
```
Text(0.5, 1.0, 'Cut Count')
```



In [25]:
```
sns.scatterplot(df, y=df.carat, x=df.price , hue=df.cut, style=df.cut)
plt.title("Relationship between carat and price with respect to cut quality of their diamond")
```

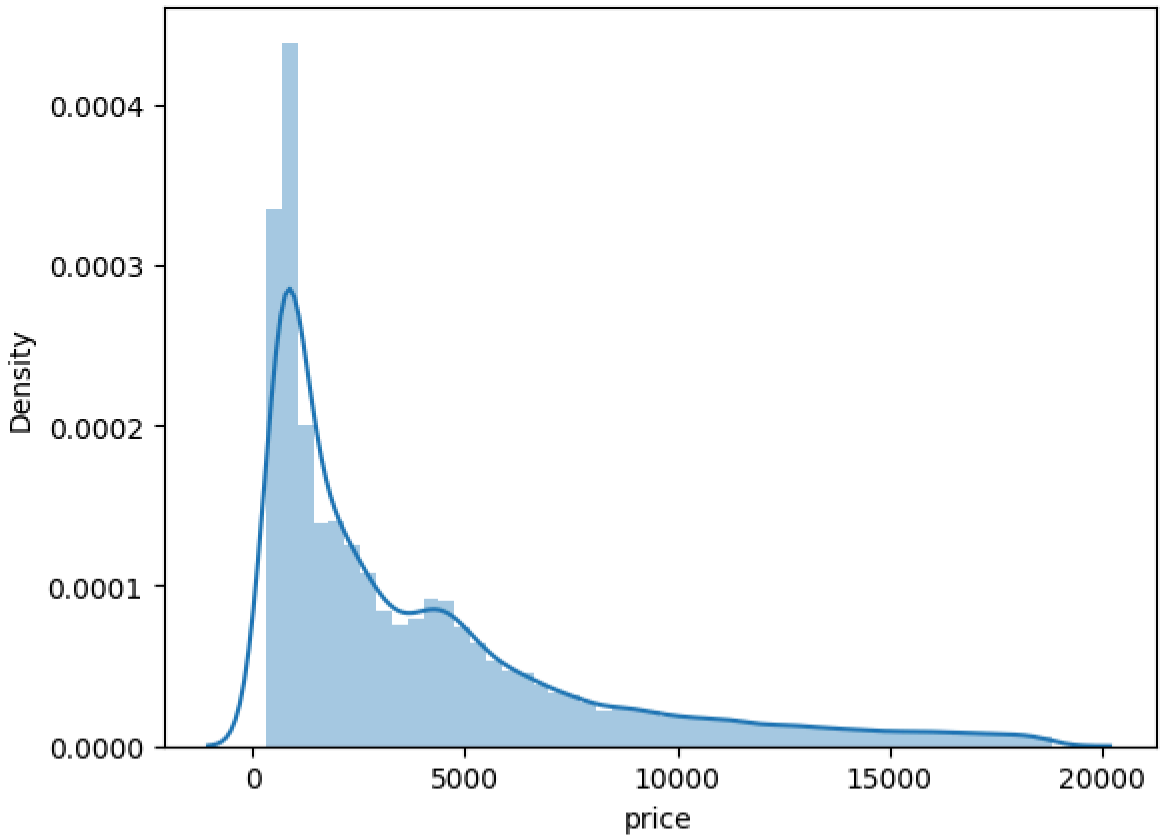
Out[25]:
```
Text(0.5, 1.0, 'Relationship between carat and price with respect to cut quality of their diamond')
```



In [26]:
```
sns.distplot(df.price)
```

```
C:\Users\loves\AppData\Local\Temp\ipykernel_19368\2239777731.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751 (https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751)

  sns.distplot(df.price)
```

Out[26]:
```
<Axes: xlabel='price', ylabel='Density'>
```



model testing

In [27]:
```
x = df.drop(columns= ['price'])
y = df.price
```

In [28]:
```
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.3, random_state=2392)
```

In [29]:
```
x_train.shape, y_train.shape
```

Out[29]:
```
((37195, 6), (37195,))
```

In [30]:
```
x_test.shape, y_test.shape
```

Out[30]:
```
((15942, 6), (15942,))
```

In [31]:
```
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import r2_score
```

Linear Regression

In [32]:
```
step1 = ColumnTransformer(transformers= [
    ('col_tnf',OneHotEncoder(drop='first'),['cut','color','clarity'])
],remainder='passthrough')

step2 = LinearRegression()

pipe = Pipeline([
    ('step1',step1),
    ('step2',step2)
])
```

In [33]:
```
pipe.fit(x_train,y_train)
```

Out[33]:

Pipeline

```
Pipeline(steps=[('step1',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('col_tnf',
                                                  OneHotEncoder(drop='first'),
                                                  ['cut', 'color',
                                                   'clarity'])])),
                ('step2', LinearRegression())])
```

(https://scikit-learn.org/1.4/modules/generated/sklear

step1: ColumnTransformer

(https://scikit-learn.org/1.4/modules/generated/sklearn.co

```
ColumnTransformer(remainder='passthrough',
                  transformers=[('col_tnf', OneHotEncoder(drop='first'),
                                 ['cut', 'color', 'clarity'])])
```

col_tnf

['cut', 'color', 'clarity']

remainder

['carat', 'depth', 'table']

OneHotEncoder

(https://scikit-learn.org/1.4/modules/generated/sklearn.preprocessing.OneHotEncoder.html)

OneHotEncoder(drop='first')

passthrough

passthrough

LinearRegression

(https://scikit-learn.org/1.4/modules/generated/sklearn.linear_model.LinearRegression.ht

LinearRegression()

In [34]:
```
y_pred = pipe.predict(x_test)
```

In [35]:
```
print('R2 Value:',r2_score(y_pred,y_test))
```

```
R2 Value: 0.9070815014367546
```

Ridge

In [36]:
```
step1 = ColumnTransformer(transformers= [
    ('col_tnf',OneHotEncoder(drop='first'),['cut','color','clarity'])
],remainder='passthrough')

step2 = Ridge(alpha=10)

pipe = Pipeline([
    ('step1',step1),
    ('step2',step2)
])
```

In [37]:
```
pipe.fit(x_train,y_train)
```

Out[37]:
```
Pipeline
(https://scikit-learn.org/1.4/modules/generated/sklearn.pipeline.Pipeline.html)
step1: ColumnTransformer
(https://scikit-learn.org/1.4/modules/generated/sklearn.compose.ColumnTransformer.html
col_tnf                          remainder
OneHotEncoder                    passthrough
(https://scikit-learn.org/1.4/modules/generated/sklearn.preprocessing.OneHotEncoder.html)
OneHotEncoder(drop='first')      passthrough
Ridge
(https://scikit-learn.org/1.4/modules/generated/sklearn.linear_model.Ridge.html)
Ridge(alpha=10)
```

In [38]:
```
y_pred=pipe.predict(x_test)

print('R2 Score:',r2_score(y_test,y_pred))
```

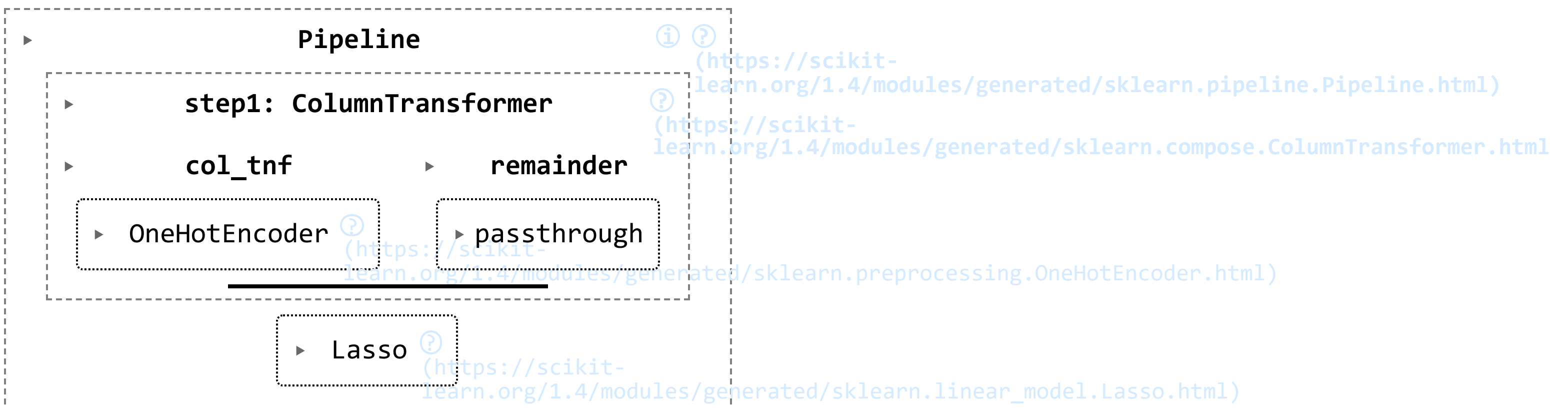
```
R2 Score: 0.915985487540254
```

Lasso

In [39]:
```
step1 = ColumnTransformer(transformers= [
    ('col_tnf',OneHotEncoder(drop='first'),['cut','color','clarity'])
],remainder='passthrough')

step2 = Lasso(alpha=0.001)

pipe = Pipeline([
    ('step1',step1),
    ('step2',step2)
])
```

In [40]:
```
pipe.fit(x_train,y_train)
```

Out[40]:



In [41]:
```
y_pred=pipe.predict(x_test)

print('R2 Score:',r2_score(y_test,y_pred))
```

```
R2 Score: 0.9161771606883635
```

KNN

In [42]:
```
step1 = ColumnTransformer(transformers= [
    ('col_tnf',OneHotEncoder(drop='first'),['cut','color','clarity'])
],remainder='passthrough')

step2 = KNeighborsRegressor(n_neighbors = 4)

pipe = Pipeline([
    ('step1',step1),
    ('step2',step2)
])
```

In [43]:
```
pipe.fit(x_train,y_train)
```

Out[43]:

Pipeline

```
Pipeline(steps=[('step1',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('col_tnf',
                                                  OneHotEncoder(drop='first'),
                                                  ['cut', 'color',
                                                   'clarity'])])),
                ('step2', KNeighborsRegressor(n_neighbors=4))])
```

(https://scikit-learn.org/1.4/modules/generated/sklear

step1: ColumnTransformer

```
ColumnTransformer(remainder='passthrough',
                  transformers=[('col_tnf', OneHotEncoder(drop='first'),
                                 ['cut', 'color', 'clarity'])])
```

(https://scikit-learn.org/1.4/modules/generated/sklearn.co

col_tnf

['cut', 'color', 'clarity']

OneHotEncoder

OneHotEncoder(drop='first')

(https://scikit-learn.org/1.4/modules/generated/sklearn.preprocessing.OneHotEncoder.html)

remainder

['carat', 'depth', 'table']

passthrough

passthrough

KNeighborsRegressor

KNeighborsRegressor(n_neighbors=4)

(https://scikit-learn.org/1.4/modules/generated/sklearn.neighbors.KNeighborsRegre

In [44]:
```
y_pred=pipe.predict(x_test)

print('R2 Score:',r2_score(y_test,y_pred))
```

R2 Score: 0.7702531600271065

Decision Trees

In [45]:
```
step1 = ColumnTransformer(transformers= [
    ('col_tnf',OneHotEncoder(sparse_output = False,drop='first'),['cut','color','clarity'])
],remainder='passthrough')

step2 = DecisionTreeRegressor(max_depth = 8)

pipe = Pipeline([
    ('step1',step1),
    ('step2',step2)
])
```

In [46]:
```
pipe.fit(x_train,y_train)
```

Out[46]:

```
Pipeline
(https://scikit-learn.org/1.4/modules/generated/sklearn.pipeline.Pipeline.html)
step1: ColumnTransformer
(https://scikit-learn.org/1.4/modules/generated/sklearn.compose.ColumnTransformer.html)
col_tnf                 remainder
OneHotEncoder           passthrough
(https://scikit-learn.org/1.4/modules/generated/sklearn.preprocessing.OneHotEncoder.html)
DecisionTreeRegressor
(https://scikit-learn.org/1.4/modules/generated/sklearn.tree.DecisionTreeRegressor.html)
```

In [47]:
```
y_pred = pipe.predict(x_test)
print('R2 Score:',r2_score(y_test, y_test))
```

```
R2 Score: 1.0
```

Random Forest Regressor

In [48]:
```
step1 = ColumnTransformer(transformers= [
    ('col_tnf',OneHotEncoder(drop='first'),['cut','color','clarity'])
],remainder='passthrough')

step2 = RandomForestRegressor(n_estimators=100,
                              random_state=3,
                              max_samples=0.5,
                              max_features=0.75,
                              max_depth=15)

pipe = Pipeline([
    ('step1',step1),
    ('step2',step2)
])

pipe.fit(x_train, y_train)

y_pred = pipe.predict(x_test)

print('R2 score',r2_score(y_test,y_pred))
```

```
R2 score 0.9736099636587119
```

Support Vector Machine(SVM)

In [49]:
```
step1 = ColumnTransformer(transformers= [
    ('col_tnf',OneHotEncoder(drop='first'),['cut','color','clarity'])
],remainder='passthrough')

step2 = SVR(kernel='rbf',C=10000,epsilon=0.1)

pipe = Pipeline([
    ('step1',step1),
    ('step2',step2)
])

pipe.fit(x_train, y_train)

y_pred = pipe.predict(x_test)

print('R2 score',r2_score(y_test,y_pred))
```

```
R2 score 0.8529194034617444
```

Xg Boost

In [50]:
```
step1 = ColumnTransformer(transformers= [
    ('col_tnf',OneHotEncoder(drop='first'),['cut','color','clarity'])
],remainder='passthrough')

step2 = XGBRegressor(n_estimators=45, max_depth=5,learning_rate=0.5)

pipe = Pipeline([
    ('step1',step1),
    ('step2',step2)
])

pipe.fit(x_train, y_train)

y_pred = pipe.predict(x_test)

print('R2 score',r2_score(y_test,y_pred))
```

```
R2 score 0.975004971630552
```

In [ ]:

In [ ]: