In [9]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

In [10]:

```
df=sns.load_dataset('diamonds')
df.head()
```

Out[10]:

	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 [11]:

```
df.shape
```

Out[11]:

```
(53940, 10)
```

In [12]:

```
df.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  category
 2   color    53940 non-null  category
 3   clarity  53940 non-null  category
 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: category(3), float64(6), int64(1)
memory usage: 3.0 MB
```

In [13]:

```
df.describe()
```

Out[13]:

	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 [14]:

```
df.nunique()
```

Out[14]:

```
carat        273
cut            5
color          7
clarity        8
depth        184
table        127
price      11602
x            554
y            552
z            375
dtype: int64
```

In [15]:

```
df.isnull().sum()
```

Out[15]:

```
carat      0
cut        0
color      0
clarity    0
depth      0
table      0
price      0
x          0
y          0
z          0
dtype: int64
```

In [16]:

```
# identifying the input and output
y=df["price"]
X=df[["carat","cut","color","clarity","depth","table","x","y","z"]]
```

In [20]:

```
# split into train and test
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.20,random_state=0)
print(X_train.shape,y_train.shape)
print(X_test.shape,y_test.shape)
```

```
(43152, 9) (43152,)
(10788, 9) (10788,)
```

In [9]:

```
X_train.dtypes
```

Out[9]:

```
carat       float64
cut        category
color      category
clarity    category
depth       float64
table       float64
x           float64
y           float64
z           float64
dtype: object
```

In [10]:

```
# separating the datatypes
X_train_num=X_train.select_dtypes(include="number")
X_train_cat=X_train.select_dtypes(exclude="number")
```

X_train_num Transform

In [11]:

```
from sklearn.preprocessing import StandardScaler

#creating the object for StandardScaler class
scaler=StandardScaler()

#Transforming and fitting the train data

X_train_num_rescaled=pd.DataFrame(scaler.fit_transform(X_train_num),columns=X_train_num.columns,index=X_train_num.index)
X_train_num_rescaled.describe()
```

Out[11]:

	carat	depth	table	x	y	z
count	3.775800e+04	3.775800e+04	3.775800e+04	3.775800e+04	3.775800e+04	3.775800e+04
mean	6.764310e-17	2.404498e-16	4.340214e-16	2.547238e-17	6.161608e-16	-5.917926e-17
std	1.000013e+00	1.000013e+00	1.000013e+00	1.000013e+00	1.000013e+00	1.000013e+00
min	-1.260601e+00	-1.307535e+01	-6.472397e+00	-5.103314e+00	-4.965602e+00	-4.977120e+00
25%	-8.392639e-01	-4.569131e-01	-6.496383e-01	-9.097871e-01	-8.791912e-01	-8.851020e-01
50%	-2.072588e-01	1.008080e-01	-2.017338e-01	-2.834518e-02	-2.208391e-02	-2.732499e-02
75%	5.090137e-01	5.190987e-01	6.940753e-01	7.195449e-01	6.965010e-01	7.038947e-01
max	8.872548e+00	1.202209e+01	9.652166e+00	4.458995e+00	4.602795e+01	3.973978e+01

Applying OneHot Encoding on categorical columns

In [13]:

```
from sklearn.preprocessing import OneHotEncoder
encoder=OneHotEncoder(drop="first",sparse=False)
#columns nams are lose after onehot encoding
# the data frame is converted to a numpy array
X_train_cat_ohe=pd.DataFrame(encoder.fit_transform(X_train_cat),columns=encoder.get_feature_names(X_train_cat.columns),index=X_tr
X_train_cat_ohe.head()
```

Out[13]:

	cut_Good	cut_Ideal	cut_Premium	cut_Very Good	color_E	color_F	color_G	color_H	color_I	color_J	clarity_IF	clarity_SI1	clarity_SI2	cla
16259	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	
24005	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	
12211	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	
37918	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
181	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

concating the X_train_num_rescaled and X_train_cat_ohe

In [14]:

```
dff1=pd.concat([X_train_num_rescaled,X_train_cat_ohe],axis=1)
```

In [15]:

```
# transformin the test data
```

In [16]:

```
X_test_num=X_test.select_dtypes(include="number")
X_test_cat=X_test.select_dtypes(exclude="number")
```

In [17]:

```
# X_test_num Transform
#Transforming and fitting the train data

X_test_num_rescaled=pd.DataFrame(scaler.transform(X_test_num),columns=X_test_num.columns,index=X_test_num.index)
X_test_num_rescaled.describe()
```

Out[17]:

	carat	depth	table	x	y	z
count	16182.000000	16182.000000	16182.000000	16182.000000	16182.000000	16182.000000
mean	-0.003102	-0.013932	0.010137	-0.002015	-0.002834	-0.003272
std	0.995295	0.995772	1.002756	0.995840	0.962224	0.974254
min	-1.260601	-12.378200	-6.024493	-5.103314	-4.965602	-4.977120
25%	-0.839264	-0.526628	-0.649638	-0.909787	-0.879191	-0.885102
50%	-0.207259	0.031093	-0.201734	-0.037249	-0.022084	-0.013263
75%	0.509014	0.519099	0.694075	0.710641	0.687843	0.689833
max	7.018667	8.257478	16.818639	3.800140	3.562183	4.064693

In [18]:

```
# Applying OneHot Encoding on categorical columns
```

In [20]:

```
#columns nams are lose after onehot encoding
# the data frame is converted to a numpy array
X_test_cat_ohe=pd.DataFrame(encoder.transform(X_test_cat),columns=encoder.get_feature_names(X_test_cat.columns),index=X_test_cat.
X_test_cat_ohe.head()
```

Out[20]:

	cut_Good	cut_Ideal	cut_Premium	cut_Very Good	color_E	color_F	color_G	color_H	color_I	color_J	clarity_IF	clarity_SI1	clarity_SI2	cla
10176	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	
16083	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	
13420	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	
20407	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	
8909	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

In [21]:

```
# concating the X_test_num_rescaled and X_test_cat_ohe
```

In [22]:

```
dff2=pd.concat([X_test_num_rescaled,X_test_cat_ohe],axis=1)
dff2
```

Out[22]:

	carat	depth	table	x	y	z	cut_Good	cut_Ideal	cut_Premium	cut_Very Good	...	color_H	color_I	color_
10176	0.635415	0.170523	-1.097543	0.781869	0.791735	0.802328	0.0	1.0	0.0	0.0	...	1.0	0.0	0
16083	1.035685	0.588814	-0.649638	1.093490	1.034149	1.139814	0.0	1.0	0.0	0.0	...	1.0	0.0	0
13420	0.846083	-0.456913	0.246171	1.022262	0.921600	0.900762	0.0	0.0	1.0	0.0	...	0.0	1.0	0
20407	1.478088	-0.596343	-0.649638	1.511952	1.406428	1.350743	0.0	1.0	0.0	0.0	...	0.0	0.0	0
8909	0.214078	-0.038622	-0.201734	0.390117	0.410799	0.394533	0.0	0.0	0.0	1.0	...	0.0	0.0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
49313	-0.965665	0.031093	-1.545447	-1.114566	-1.061002	-1.067907	0.0	1.0	0.0	0.0	...	0.0	0.0	0
32991	-0.881398	-0.038622	0.694075	-0.972111	-0.983083	-0.969473	0.0	0.0	1.0	0.0	...	0.0	1.0	0
18841	1.499155	-0.944919	0.694075	1.369497	1.354482	1.210124	0.0	0.0	0.0	1.0	...	1.0	0.0	0
25490	2.910633	0.588814	-1.097543	2.259842	2.150986	2.278830	0.0	1.0	0.0	0.0	...	0.0	0.0	0
17489	2.636764	-3.384948	0.246171	2.429008	2.220247	1.730415	1.0	0.0	0.0	0.0	...	0.0	0.0	0

16182 rows × 23 columns

In [24]:

```
# Building the model
# LinearRegression
```

In [25]:

```
import time
```

In [29]:

```
start_time=time.time()
from sklearn.linear_model import LinearRegression
regressor= LinearRegression()
regressor.fit(dff1,y_train)
y_test_pred=regressor.predict(dff2)
print("training  is completed in {} seconds".format(time.time()-start_time))
from sklearn import metrics
metrics.mean_absolute_error(y_test,y_test_pred)
```

```
training  is completed in 0.02797389030456543 seconds
```

Out[29]:

```
739.9743204473408
```

In [30]:

```
# DecisionTreeRegressor
```

In [31]:

```
start_time=time.time()
from sklearn.tree import DecisionTreeRegressor
regressor= DecisionTreeRegressor()
regressor.fit(dff1,y_train)
y_test_pred=regressor.predict(dff2)
print("training  is completed in {} seconds".format(time.time()-start_time))
from sklearn import metrics
metrics.mean_absolute_error(y_test,y_test_pred)
```

```
training  is completed in 1.3454275131225586 seconds
```

Out[31]:

```
401.1144790507972
```

In [32]:

```
# KNeighborsRegressor
```

In [41]:

```
start_time=time.time()
from sklearn.neighbors import KNeighborsRegressor
regressor= KNeighborsRegressor()
regressor.fit(dff1,y_train)
y_test_pred=regressor.predict(dff2)
print("training  is completed in {} seconds".format(time.time()-start_time))
from sklearn import metrics
metrics.mean_absolute_error(y_test,y_test_pred)
```

```
training  is completed in 11.928352355957031 seconds
```

Out[41]:

```
427.98060808305524
```

In [34]:

```
# AdaBoostRegressor
```

In [35]:

```
from sklearn.ensemble import AdaBoostRegressor
regressor= AdaBoostRegressor()
regressor.fit(dff1,y_train)
y_test_pred=regressor.predict(dff2)
from sklearn import metrics
metrics.mean_absolute_error(y_test,y_test_pred)
```

Out[35]:

```
1078.3016971236402
```

In [36]:

```
# GradientBoostingRegressor
```

In [37]:

```
from sklearn.ensemble import GradientBoostingRegressor
regressor= GradientBoostingRegressor()
regressor.fit(dff1,y_train)
y_test_pred=regressor.predict(dff2)
from sklearn import metrics
metrics.mean_absolute_error(y_test,y_test_pred)
```

Out[37]:

```
447.1391716535539
```

In [38]:

```
# RandomForestRegressor
```

In [40]:

```
start_time=time.time()
from sklearn.ensemble import RandomForestRegressor
regressor= RandomForestRegressor()
regressor.fit(dff1,y_train)
y_test_pred=regressor.predict(dff2)
print("training  is completed in {} seconds".format(time.time()-start_time))
from sklearn import metrics
metrics.mean_absolute_error(y_test,y_test_pred)
```

```
training  is completed in 13.527787208557129 seconds
```

Out[40]:

```
300.7088142623197
```

In [ ]:

```

```