In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import RFE
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.feature_selection import SelectFromModel
```

In [2]:

```
#load the data
data=pd.read_csv("daimonds(predict price).csv")
```

In [3]:

```
#realize the data
data.shape
```

Out[3]:

```
(53940, 12)
```

In [4]:

```
#glance the data
#第一col是多餘的(要刪掉)，第二col為編號，要預測為price
data.head()
```

Out[4]:

	Unnamed: 0	Unnamed: 0.1	carat	cut	color	clarity	depth	table	price	x	y	z
0	0	1	0.23	NaN	E	SI2	61.5	55.0	326.0	3.95	3.98	2.43
1	1	2	0.21	Premium	E	SI1	59.8	NaN	326.0	3.89	3.84	2.31
2	2	3	0.23	Good	E	VS1	56.9	65.0	327.0	4.05	4.07	2.31
3	3	4	0.29	Premium	I	NaN	62.4	58.0	334.0	4.20	NaN	2.63
4	4	5	0.31	Good	J	SI2	63.3	58.0	335.0	4.34	4.35	2.75

In [5]:

```
#刪除第一個col和編號
data=data.drop(["Unnamed: 0","Unnamed: 0.1"],axis=1)
data
```

Out[5]:

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	NaN	E	SI2	61.5	55.0	326.0	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	NaN	326.0	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327.0	4.05	4.07	2.31
3	0.29	Premium	I	NaN	62.4	58.0	334.0	4.20	NaN	2.63
4	0.31	Good	J	SI2	63.3	58.0	335.0	4.34	4.35	2.75
...	...	...	...	...	...	...	...	...	...	...
53935	0.72	Ideal	D	SI1	60.8	57.0	2757.0	5.75	5.76	3.50
53936	0.72	Good	D	SI1	63.1	55.0	2757.0	5.69	5.75	3.61
53937	0.70	Very Good	D	SI1	62.8	60.0	2757.0	5.66	5.68	3.56
53938	0.86	Premium	H	SI2	61.0	58.0	2757.0	6.15	6.12	3.74

In [6]:

```
#checking null data
#資料集看起來夠大，可以刪除空值
print(data.isnull().sum())
```

```
carat       993
cut         989
color       992
clarity     995
depth       990
table       993
price       992
x           991
y           997
z           992
dtype: int64
```

In [7]:

```
data_drop=data.dropna()
data_drop
```

Out[7]:

	carat	cut	color	clarity	depth	table	price	x	y	z
2	0.23	Good	E	VS1	56.9	65.0	327.0	4.05	4.07	2.31
4	0.31	Good	J	SI2	63.3	58.0	335.0	4.34	4.35	2.75
5	0.24	Very Good	J	VVS2	62.8	57.0	336.0	3.94	3.96	2.48
6	0.24	Very Good	I	VVS1	62.3	57.0	336.0	3.95	3.98	2.47
7	0.26	Very Good	H	SI1	61.9	55.0	337.0	4.07	4.11	2.53
...	...	...	...	...	...	...	...	...	...	...
53934	0.72	Premium	D	SI1	62.7	59.0	2757.0	5.69	5.73	3.58
53935	0.72	Ideal	D	SI1	60.8	57.0	2757.0	5.75	5.76	3.50
53936	0.72	Good	D	SI1	63.1	55.0	2757.0	5.69	5.75	3.61
53937	0.70	Very Good	D	SI1	62.8	60.0	2757.0	5.66	5.68	3.56
53938	0.86	Premium	H	SI2	61.0	58.0	2757.0	6.15	6.12	3.74

44773 rows × 10 columns

In [8]:

```
#check null data
#not null data
print(data_drop.isnull().sum())
```

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

In [9]:

```
#有feature是文字，將其轉換成數字
#依照鑽石的等級分類表
#先觀察每一個feature(cut,color,clarity)的value_counts
```

In [10]:

```
#cut
data_drop["cut"].value_counts()
```

Out[10]:

```
Ideal        17852
Premium      11502
Very Good     9978
Good          4096
Fair          1345
Name: cut, dtype: int64
```

In [11]:

```
#color
data_drop["color"].value_counts()
```

Out[11]:

```
G    9405
E    8076
F    7973
H    6919
D    5593
I    4496
J    2311
Name: color, dtype: int64
```

In [12]:

```
#clarity
data_drop["clarity"].value_counts()
```

Out[12]:

```
SI1     10854
VS2     10191
SI2      7626
VS1      6800
VVS2     4192
VVS1     3009
IF       1495
I1        606
Name: clarity, dtype: int64
```

In [13]:

```
#將所有的轉換成數值
cut_mapping = {
    'Fair':1.0,
    'Good':2.0,
    'Very Good':3.0,
    'Premium':4.0,
    'Ideal':5.0
}
data_drop['cut']=data_drop['cut'].map(cut_mapping)
color_mapping = {
    'J':1.0,
    'I':2.0,
    'H':3.0,
    'G':4.0,
    'F':5.0,
    'E':6.0,
    'D':7.0
}
data_drop['color']=data_drop['color'].map(color_mapping)
clarity_mapping = {
    'I1':1.0,
    'SI2':2.0,
    'SI1':3.0,
    'VS2':4.0,
    'VS1':5.0,
    'VVS2':6.0,
    'VVS1':7.0,
    'IF':8.0
}
data_drop['clarity']=data_drop['clarity'].map(clarity_mapping)
data_drop
```

```
<ipython-input-13-7b12959132d0>:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  data_drop['cut']=data_drop['cut'].map(cut_mapping)
<ipython-input-13-7b12959132d0>:19: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  data_drop['color']=data_drop['color'].map(color_mapping)
<ipython-input-13-7b12959132d0>:30: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
```

```
ng-a-view-versus-a-copy)
  data_drop['clarity']=data_drop['clarity'].map(clarity_mapping)
```

Out[13]:

	carat	cut	color	clarity	depth	table	price	x	y	z
2	0.23	2.0	6.0	5.0	56.9	65.0	327.0	4.05	4.07	2.31
4	0.31	2.0	1.0	2.0	63.3	58.0	335.0	4.34	4.35	2.75
5	0.24	3.0	1.0	6.0	62.8	57.0	336.0	3.94	3.96	2.48
6	0.24	3.0	2.0	7.0	62.3	57.0	336.0	3.95	3.98	2.47
7	0.26	3.0	3.0	3.0	61.9	55.0	337.0	4.07	4.11	2.53
...	...	...	...	...	...	...	...	...	...	...
53934	0.72	4.0	7.0	3.0	62.7	59.0	2757.0	5.69	5.73	3.58
53935	0.72	5.0	7.0	3.0	60.8	57.0	2757.0	5.75	5.76	3.50
53936	0.72	2.0	7.0	3.0	63.1	55.0	2757.0	5.69	5.75	3.61
53937	0.70	3.0	7.0	3.0	62.8	60.0	2757.0	5.66	5.68	3.56
53938	0.86	4.0	3.0	2.0	61.0	58.0	2757.0	6.15	6.12	3.74

44773 rows × 10 columns

In [14]:

```
##區分出預測值和feature
y=data_drop["price"]
x=data_drop.drop(["price"],axis=1)
```

In [15]:

```
columns = x.columns
```

In [16]:

```
#將x標準化
from sklearn import preprocessing
x = preprocessing.scale(x)
```

In [17]:

```
x_pd = pd.DataFrame(x,columns=columns)
x_pd
```

Out[17]:

	carat	cut	color	clarity	depth	table	x	y	z
0	-1.201147	-1.702967	0.940515	0.576862	-3.371648	3.370086	-1.500724	-1.452156	-1.737904
1	-1.031920	-1.702967	-2.004794	-1.246023	1.079742	0.241853	-1.241903	-1.207954	-1.115649
2	-1.179994	-0.807976	-2.004794	1.184491	0.731977	-0.205038	-1.598897	-1.548093	-1.497487
3	-1.179994	-0.807976	-1.415732	1.792120	0.384212	-0.205038	-1.589972	-1.530650	-1.511629
4	-1.137687	-0.807976	-0.826670	-0.638395	0.106000	-1.098819	-1.482874	-1.417270	-1.426776
...	...	...	...	...	...	...	...	...	...
44768	-0.164628	0.087014	1.529576	-0.638395	0.662424	0.688743	-0.037046	-0.004387	0.058151
44769	-0.164628	0.982005	1.529576	-0.638395	-0.659082	-0.205038	0.016503	0.021778	-0.054986
44770	-0.164628	-1.702967	1.529576	-0.638395	0.940636	-1.098819	-0.037046	0.013056	0.100578
44771	-0.206935	-0.807976	1.529576	-0.638395	0.731977	1.135633	-0.063821	-0.047994	0.029867

In [18]:

```
y.index=x_pd.index
y.index
```

Out[18]:

```
RangeIndex(start=0, stop=44773, step=1)
```

In [19]:

```
data=pd.concat([y,x_pd],axis=1)
data
```

Out[19]:

	price	carat	cut	color	clarity	depth	table	x	y	
0	327.0	-1.201147	-1.702967	0.940515	0.576862	-3.371648	3.370086	-1.500724	-1.452156	-1.737
1	335.0	-1.031920	-1.702967	-2.004794	-1.246023	1.079742	0.241853	-1.241903	-1.207954	-1.115
2	336.0	-1.179994	-0.807976	-2.004794	1.184491	0.731977	-0.205038	-1.598897	-1.548093	-1.497
3	336.0	-1.179994	-0.807976	-1.415732	1.792120	0.384212	-0.205038	-1.589972	-1.530650	-1.511
4	337.0	-1.137687	-0.807976	-0.826670	-0.638395	0.106000	-1.098819	-1.482874	-1.417270	-1.426
...	...	...	...	...	...	...	...	...	...	
44768	2757.0	-0.164628	0.087014	1.529576	-0.638395	0.662424	0.688743	-0.037046	-0.004387	0.058
44769	2757.0	-0.164628	0.982005	1.529576	-0.638395	-0.659082	-0.205038	0.016503	0.021778	-0.054
44770	2757.0	-0.164628	-1.702967	1.529576	-0.638395	0.940636	-1.098819	-0.037046	0.013056	0.100
44771	2757.0	-0.206935	-0.807976	1.529576	-0.638395	0.731977	1.135633	-0.063821	-0.047994	0.029

In [20]:

```
y=data["price"]
x=data.drop(["price"],axis=1)
```

In [21]:

```
#將資料區分為訓練集和測試集
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3,random_state=42)
```

In [22]:

```
x_train
```

Out[22]:

	carat	cut	color	clarity	depth	table	x	y	z
1281	-0.143474	-2.597958	0.351453	0.576862	-2.189248	3.816976	0.168226	0.126436	-0.111555
12071	0.618050	-0.807976	-0.826670	0.576862	-1.076400	1.582524	0.828666	0.841599	0.680407
1163	0.554590	0.087014	-1.415732	-1.246023	-0.172212	-0.205038	0.748342	0.658447	0.680407
27057	-1.010766	0.982005	0.351453	1.184491	-0.589529	-0.651928	-1.152654	-1.112018	-1.172217
5633	-0.841539	-0.807976	0.351453	-1.246023	0.245106	-0.205038	-0.920607	-0.867816	-0.861090
...	...	...	...	...	...	...	...	...	...
11284	1.168040	0.087014	-2.004794	-0.638395	-0.589529	0.688743	1.194585	1.207902	1.104672
44732	-0.206935	0.982005	0.351453	0.576862	0.523318	-0.651928	-0.001347	-0.030551	0.044009
38158	-0.503083	0.087014	1.529576	-0.638395	-0.659082	1.135633	-0.394041	-0.370690	-0.450967
860	-0.122321	-1.702967	0.940515	-0.638395	0.731977	1.582524	0.007578	0.021778	0.100578

In [23]:

```
y_train
```

Out[23]:

```
1281     3002.0
12071    5845.0
1163     2968.0
27057     800.0
5633      579.0
          ...
11284    5546.0
44732    2751.0
38158    1723.0
860      2906.0
15795    7797.0
Name: price, Length: 31341, dtype: float64
```

In [24]:

```
#先觀察都沒做feature selection前的mode準確度
model = LinearRegression()
model.fit(x_train,y_train)
```

Out[24]:

```
LinearRegression()
```

In [25]:

```
#觀察迴歸模型的準確度
score = model.score(x_test, y_test)
print('Score: ', score)
print('Accuracy: ' + str(score*100) + '%')
```

```
Score:  0.907752783949035
Accuracy: 90.77527839490351%
```

In [26]:

```
#feature selection
#使用filter法中的皮爾森相關係數方法作為篩選
#設在相關係數前n大的數字
def feature_selection_n(n):
    featuresCorr = data.corr()
    targetCorr=abs(featuresCorr["price"])
    targetCorr = targetCorr.drop("price")
    result =pd.Series.argsort(-targetCorr)
    targetCorr=targetCorr[result]
    selectedFeatures=targetCorr[0:n]
    #print(f"Number of selected features: {len(selectedFeatures)} \n\nHighly relative featu
    return selectedFeatures
```

In [27]:

```
selectedFeatures=feature_selection_n(4)
selectedFeatures
```

Out[27]:

```
carat    0.920902
x        0.883254
y        0.860264
z        0.857077
Name: price, dtype: float64
```

In [28]:

```
x_train=x_train[selectedFeatures.index]
data_new=pd.concat([y_train,x_train],axis=1)
data_new
```

Out[28]:

	price	carat	x	y	z
1281	3002.0	-0.143474	0.168226	0.126436	-0.111555
12071	5845.0	0.618050	0.828666	0.841599	0.680407
1163	2968.0	0.554590	0.748342	0.658447	0.680407
27057	800.0	-1.010766	-1.152654	-1.112018	-1.172217
5633	579.0	-0.841539	-0.920607	-0.867816	-0.861090
...	...	...	...	...	...
11284	5546.0	1.168040	1.194585	1.207902	1.104672
44732	2751.0	-0.206935	-0.001347	-0.030551	0.044009
38158	1723.0	-0.503083	-0.394041	-0.370690	-0.450967
860	2906.0	-0.122321	0.007578	0.021778	0.100578

In [29]:

```
data_new.to_csv('daimonds_HW1_train.csv')
```

In [30]:

```
#觀察select後的準確度
```

In [31]:

```
x=data_new.drop(["price"],axis=1)
y=data_new["price"]
```

In [32]:

```
model = LinearRegression()
model.fit(x_train,y_train)
```

Out[32]:

```
LinearRegression()
```

In [33]:

```
x_test=x_test[selectedFeatures.index]
data_test=pd.concat([y_test,x_test],axis=1)
data_test
```

Out[33]:

	price	carat	x	y	z
8304	4694.0	0.448823	0.605544	0.545068	0.623838
34038	1184.0	-0.820385	-0.867058	-0.832930	-0.875232
10458	5262.0	0.448823	0.587694	0.658447	0.623838
25228	726.0	-0.926152	-1.054480	-0.981195	-1.087364
6561	4303.0	0.702664	0.828666	0.780549	0.864255
...	...	...	...	...	...
42843	2392.0	0.491129	0.623394	0.588675	0.694549
15400	7528.0	0.639204	0.766192	0.780549	0.779402
13538	6488.0	0.300748	0.435972	0.466574	0.510701
39448	1865.0	-0.206935	-0.135219	-0.170095	0.128862
35025	1286.0	-0.630004	-0.661786	-0.702107	-0.620673

13432 rows × 5 columns

In [34]:

```
#這邊發現無論怎麼調整相關係數的值，只要少任何一個feature，accuracy就會降低，故匯出保留所有feature
score = model.score(x_test, y_test)
print('Score: ', score)
print('Accuracy: ' + str(score*100) + '%')
```

```
Score:  0.8536277743784209
Accuracy: 85.36277743784208%
```

In [35]:

```
data_test.to_csv('daimonds_HW1_test.csv')
```

In [ ]:

```

```