Problem statement:

Create a classification model to predict whether price range ofmobile based on certain specifications

Context:

An entrepreneur has started his own mobile company. He wants to givetough fight to big companies like Apple, Samsung etc.He does not know how to estimate price of mobiles his company creates. In thiscompetitive mobile phone market, one cannot simply assume things. To solve thisproblem, he collects sales data of mobile phones of various companies.He wants to find out some relation between features of a mobile phone (e.g., RAM,Internal Memory etc) and its selling price. But he is not so good at MachineLearning.So, he needs your help to solve this problem.In this problem you do not have to predict actual price but a price range indicating how high the price is

Details of features:

The columns are described as follows:

Dataset as 21 features and 2000 entries. The meanings of the features are given below.

- **battery_power: Total energy a battery can store in one time measured in mAh**

- **blue: Has bluetooth or not**

- **clock_speed: speed at which microprocessor executes instructions**
- **dual_sim: Has dual sim support or not**
- **fc: Front Camera mega pixels**
- **four_g: Has 4G or not**
- **int_memory: Internal Memory in Gigabytes**
- **m_dep: Mobile Depth in cm**
- **mobile_wt: Weight of mobile phone**
- **n_cores: Number of cores of processor**
- **pc: Primary Camera mega pixels**
- **px_height: Pixel Resolution Height**
- **px_width: Pixel Resolution Width**
- **ram: Random Access Memory in Mega Bytes**
- **sc_h: Screen Height of mobile in cm**
- **sc_w: Screen Width of mobile in cm**

- **talk_time: longest time that a single battery charge will last when you are**

- **three_g: Has 3G or not**

- **touch_screen: Has touch screen or not**

- **wifi: Has wifi or not**

- **price_range: This is the target variable with value of 0(low cost), 1(medium cost),2(high cost) and 3(very high cost)**

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

Remove warnings

```
import warnings
warnings.filterwarnings('ignore')
```

Import dataset

```
dataset=pd.read_csv("C:/Users/reddy/OneDrive/Desktop/mobile_price_range_data.
csv")
```

Describe dataset

```
dataset.describe()
```

```
       battery_power       blue  clock_speed     dual_sim           fc  \
count    2000.000000  2000.0000  2000.000000  2000.000000  2000.000000
mean     1238.518500     0.4950     1.522250     0.509500     4.309500
std       439.418206     0.5001     0.816004     0.500035     4.341444
min       501.000000     0.0000     0.500000     0.000000     0.000000
25%       851.750000     0.0000     0.700000     0.000000     1.000000
50%      1226.000000     0.0000     1.500000     1.000000     3.000000
75%      1615.250000     1.0000     2.200000     1.000000     7.000000
max      1998.000000     1.0000     3.000000     1.000000    19.000000

            four_g   int_memory        m_dep    mobile_wt      n_cores  ...
```

```
\
count  2000.000000  2000.000000  2000.000000  2000.000000  2000.000000  ...
mean      0.521500    32.046500     0.501750   140.249000     4.520500  ...
std       0.499662    18.145715     0.288416    35.399655     2.287837  ...
min       0.000000     2.000000     0.100000    80.000000     1.000000  ...
25%       0.000000    16.000000     0.200000   109.000000     3.000000  ...
50%       1.000000    32.000000     0.500000   141.000000     4.000000  ...
75%       1.000000    48.000000     0.800000   170.000000     7.000000  ...
max       1.000000    64.000000     1.000000   200.000000     8.000000  ...

         px_height     px_width          ram         sc_h         sc_w  \
count  2000.000000  2000.000000  2000.000000  2000.000000  2000.000000
mean    645.108000  1251.515500  2124.213000    12.306500     5.767000
std     443.780811   432.199447  1084.732044     4.213245     4.356398
min       0.000000   500.000000   256.000000     5.000000     0.000000
25%     282.750000   874.750000  1207.500000     9.000000     2.000000
50%     564.000000  1247.000000  2146.500000    12.000000     5.000000
75%     947.250000  1633.000000  3064.500000    16.000000     9.000000
max    1960.000000  1998.000000  3998.000000    19.000000    18.000000

         talk_time     three_g  touch_screen         wifi  price_range
count  2000.000000  2000.000000   2000.000000  2000.000000  2000.000000
mean     11.011000     0.761500      0.503000     0.507000     1.500000
std       5.463955     0.426273      0.500116     0.500076     1.118314
min       2.000000     0.000000      0.000000     0.000000     0.000000
25%       6.000000     1.000000      0.000000     0.000000     0.750000
50%      11.000000     1.000000      1.000000     1.000000     1.500000
75%      16.000000     1.000000      1.000000     1.000000     2.250000
max      20.000000     1.000000      1.000000     1.000000     3.000000

[8 rows x 21 columns]
```

Checking null values

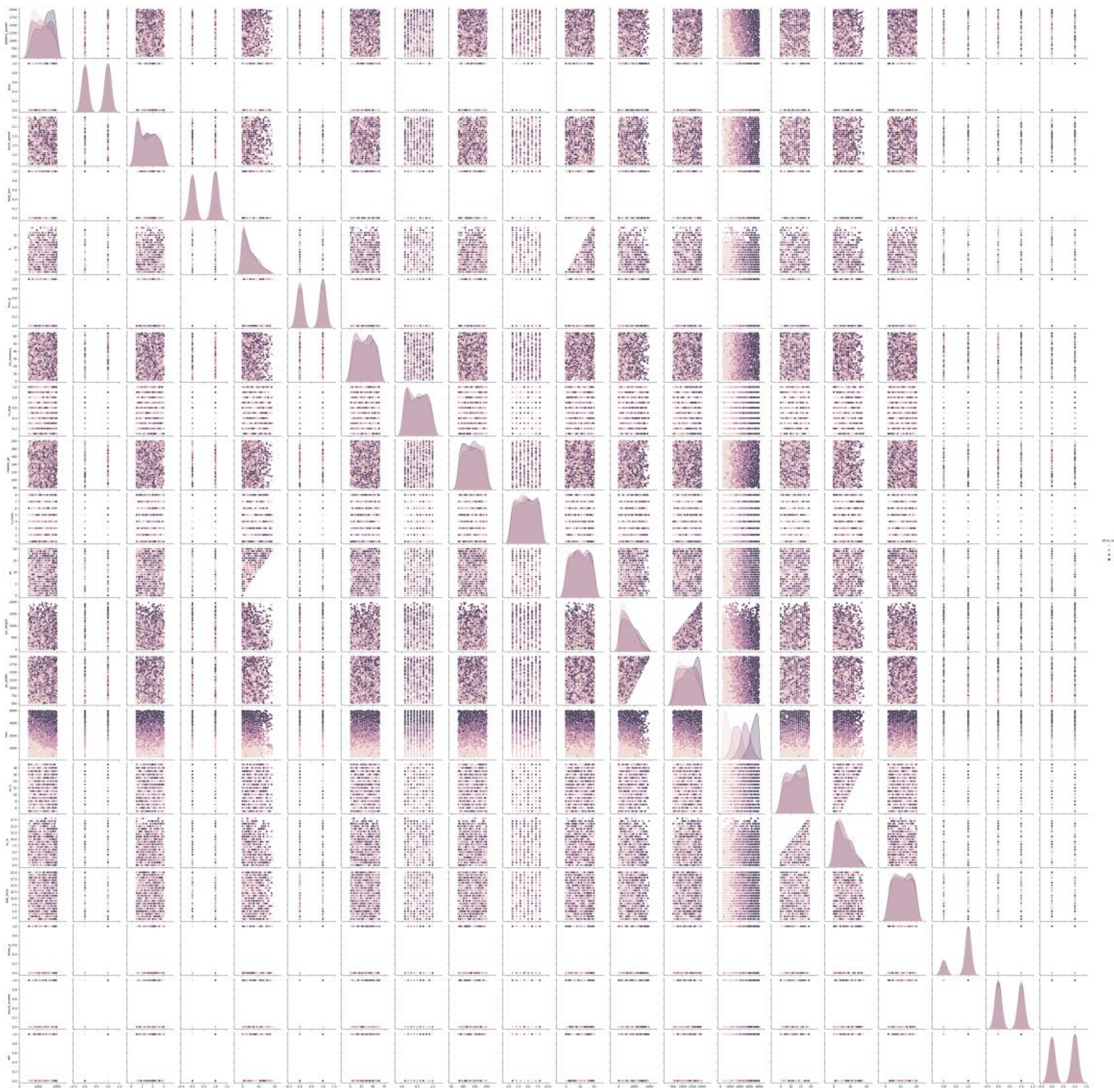
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   battery_power  2000 non-null   int64
 1   blue           2000 non-null   int64
 2   clock_speed    2000 non-null   float64
 3   dual_sim       2000 non-null   int64
 4   fc             2000 non-null   int64
 5   four_g         2000 non-null   int64
 6   int_memory     2000 non-null   int64
 7   m_dep          2000 non-null   float64
```

```
 8   mobile_wt      2000 non-null   int64
 9   n_cores        2000 non-null   int64
 10  pc             2000 non-null   int64
 11  px_height      2000 non-null   int64
 12  px_width       2000 non-null   int64
 13  ram            2000 non-null   int64
 14  sc_h           2000 non-null   int64
 15  sc_w           2000 non-null   int64
 16  talk_time      2000 non-null   int64
 17  three_g        2000 non-null   int64
 18  touch_screen   2000 non-null   int64
 19  wifi           2000 non-null   int64
 20  price_range    2000 non-null   int64
dtypes: float64(2), int64(19)
memory usage: 328.2 KB
```

```
sns.pairplot(dataset,hue="price_range")
```

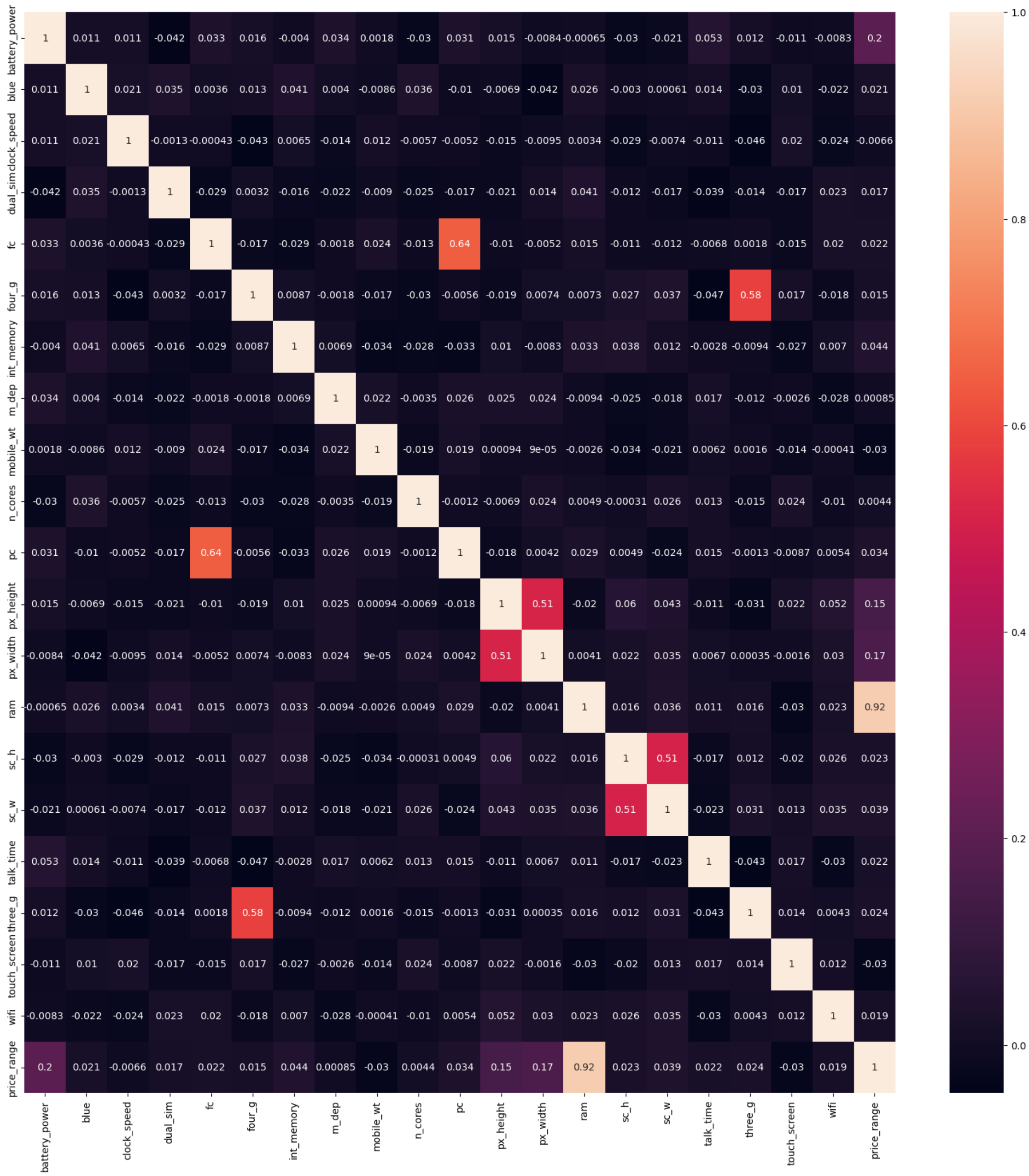
```
<seaborn.axisgrid.PairGrid at 0x1e86c026490>
```



Coorealation

```
plt.figure(figsize=(20,20))
sns.heatmap(dataset.corr(), annot=True)
```

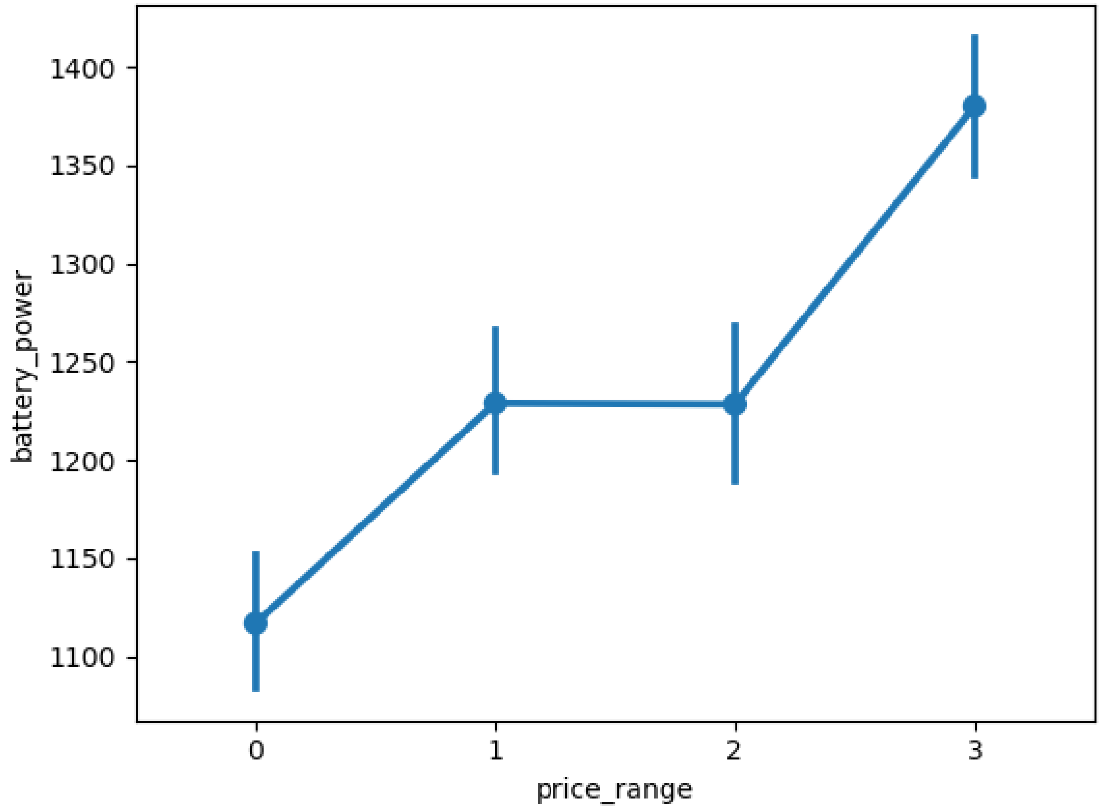
<AxesSubplot:>



battery_power vs price_range

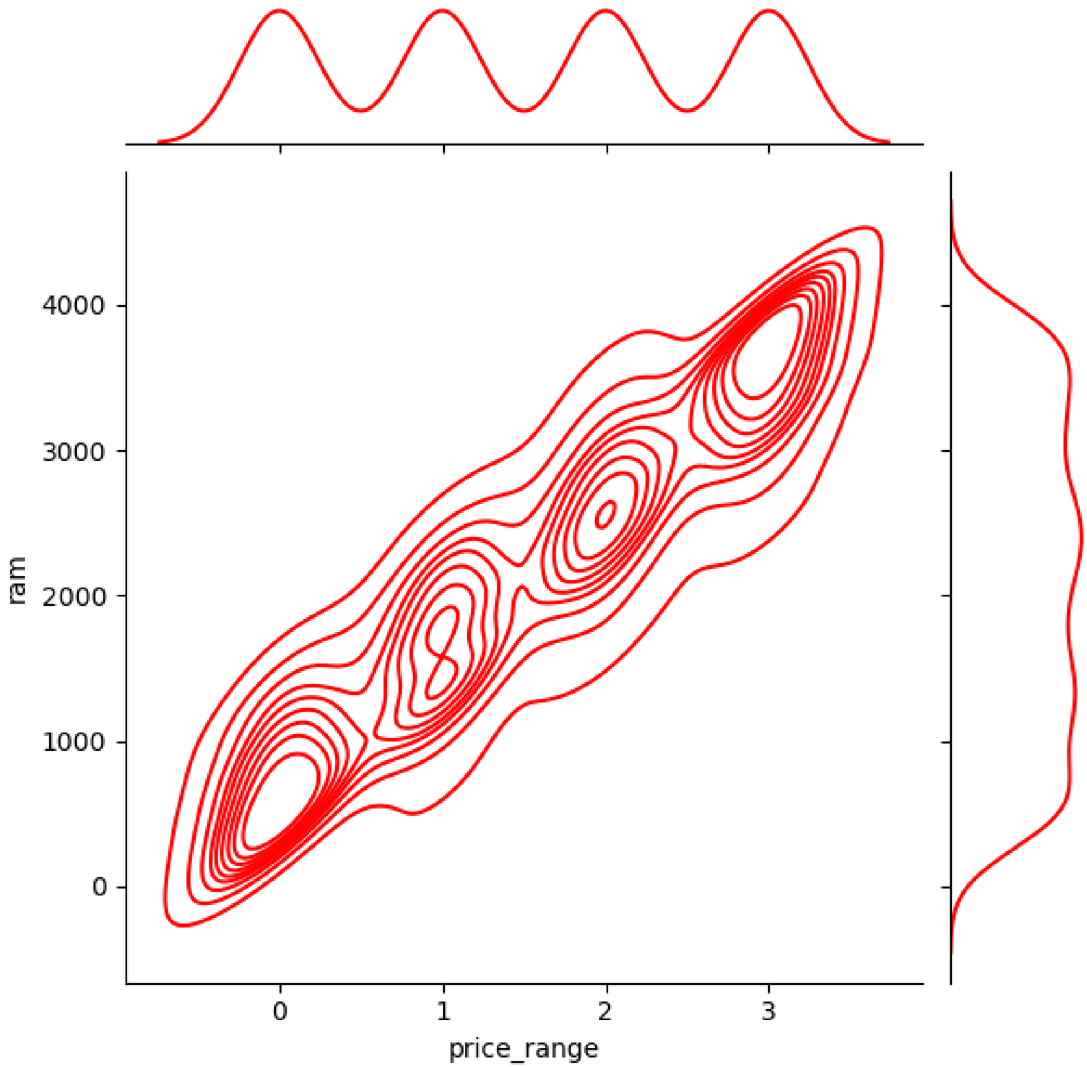
```
sns.pointplot(y="battery_power",x="price_range",data=dataset)
```

```
<AxesSubplot:xlabel='price_range', ylabel='battery_power'>
```



ram vs price_range

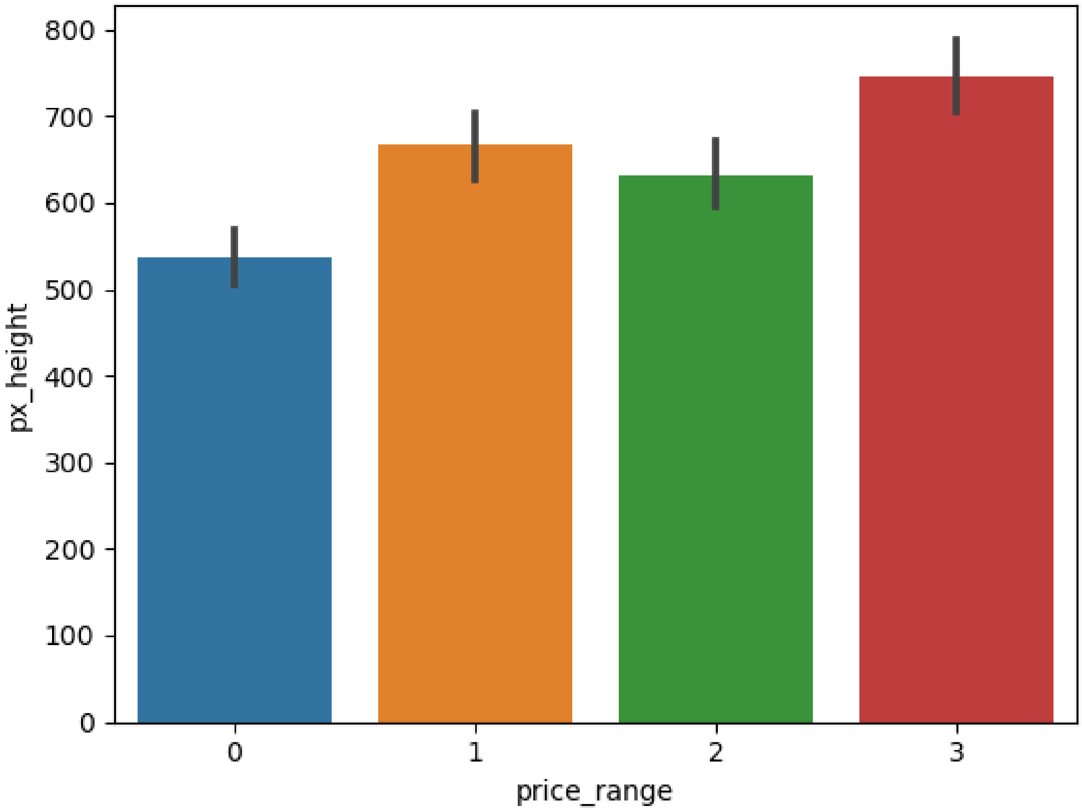
```
sns.jointplot(y="ram",x="price_range",data=dataset,color='red',kind='kde');
```



px_height vs price_range

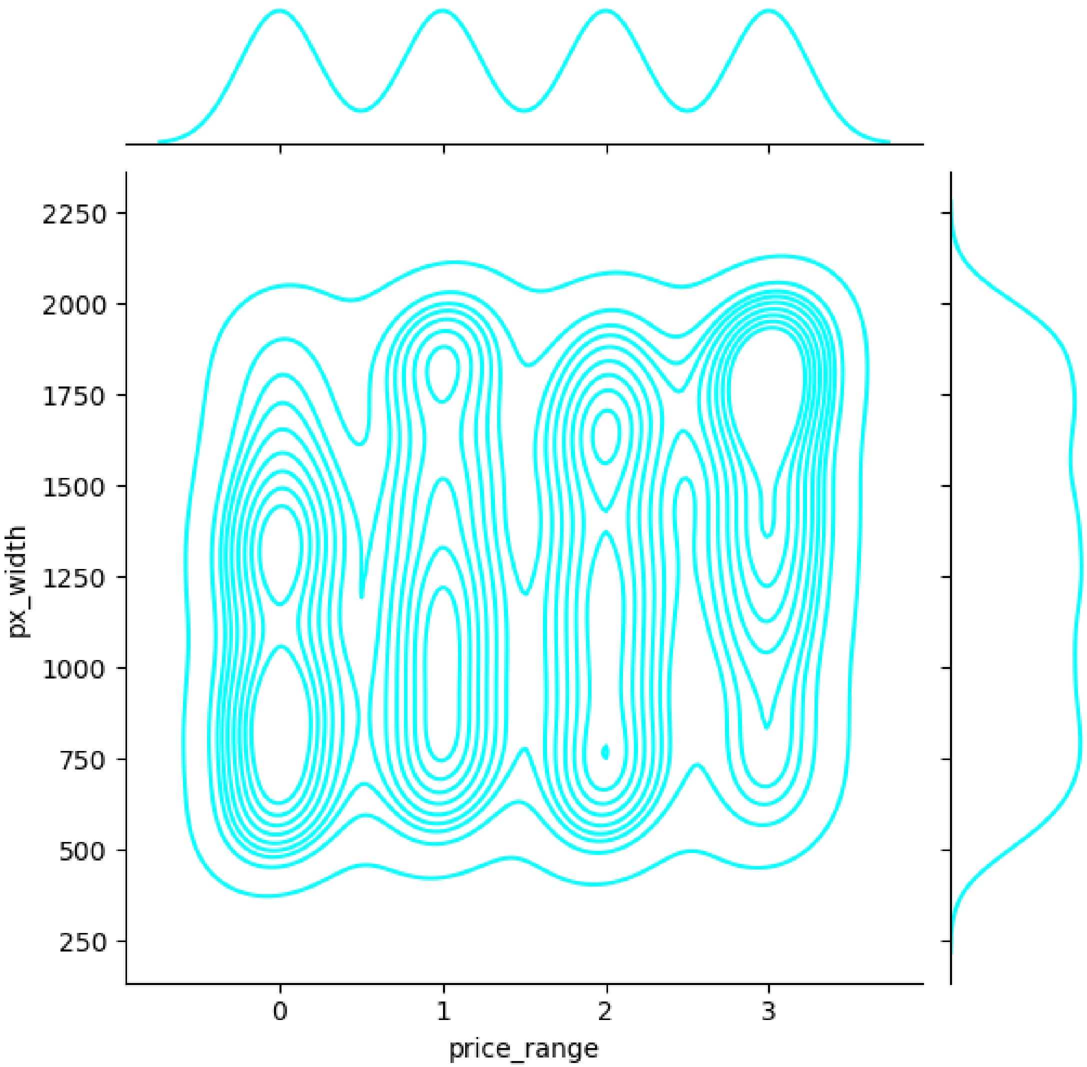
```
sns.barplot(y="px_height",x="price_range",data=dataset)
```

```
<AxesSubplot:xlabel='price_range', ylabel='px_height'>
```



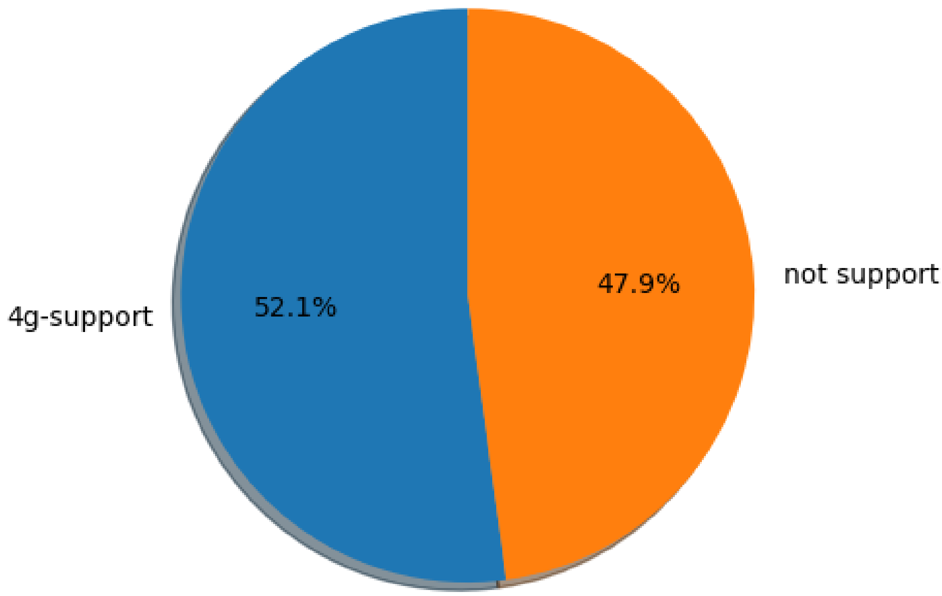
px_width vs price_range

```
sns.jointplot(y="px_width",x="price_range",data=dataset,color='cyan',kind='kde');
```



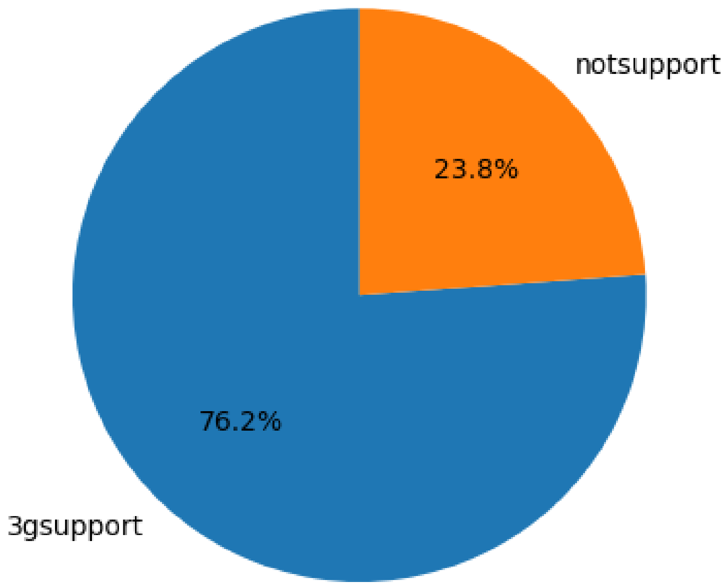
four_g vs price_range

```
labels4g=["4g-support","not support"]
values4g=dataset["four_g"].value_counts().values
f1,a1=plt.subplots()
a1.pie(values4g,labels=labels4g,shadow=True,startangle=90,autopct='%1.1f%%')
plt.show()
```



three_g vs price_range

```
labels4g=["3gsupport","notsupport"]
values4g=dataset["three_g"].value_counts()
a1,f1=plt.subplots()
f1.pie(values4g,labels=labels4g,autopct="%1.1f%%",startangle=90)
plt.show()
```




```
x=dataset.drop('price_range',axis=1)
y = dataset['price_range']
```

```
x
```

```
      battery_power  blue  clock_speed  dual_sim  fc  four_g  int_memory  \
0               842     0          2.2         0   1       0           7   
1              1021     1          0.5         1   0       1          53   
2               563     1          0.5         1   2       1          41   
3               615     1          2.5         0   0       0          10   
4              1821     1          1.2         0  13       1          44   
...             ...   ...          ...       ...  ..     ...         ...   
1995            794     1          0.5         1   0       1           2   
1996           1965     1          2.6         1   0       0          39   
1997           1911     0          0.9         1   1       1          36   
1998           1512     0          0.9         0   4       1          46   
1999            510     1          2.0         1   5       1          45   

      m_dep  mobile_wt  n_cores  pc  px_height  px_width   ram  sc_h  sc_w  \
0       0.6        188        2   2         20       756  2549     9     7   
1       0.7        136        3   6        905      1988  2631    17     3   
2       0.9        145        5   6       1263      1716  2603    11     2   
3       0.8        131        6   9       1216      1786  2769    16     8   
4       0.6        141        2  14       1208      1212  1411     8     2   
...     ...        ...      ...  ..        ...       ...   ...   ...   ...   
1995    0.8        106        6  14       1222      1890   668    13     4   
1996    0.2        187        4   3        915      1965  2032    11    10   
1997    0.7        108        8   3        868      1632  3057     9     1   
```

```
1998    0.1        145        5   5        336       670   869     18     10
1999    0.9        168        6  16        483       754  3919     19      4

      talk_time  three_g  touch_screen  wifi
0            19        0             0     1
1             7        1             1     0
2             9        1             1     0
3            11        1             0     0
4            15        1             1     0
...         ...      ...           ...   ...
1995         19        1             1     0
1996         16        1             1     1
1997          5        1             1     0
1998         19        1             1     1
1999          2        1             1     1

[2000 rows x 20 columns]
```

```
y
```

```
0       1
1       2
2       2
3       2
4       1
       ..
1995    0
1996    2
1997    3
1998    0
1999    3
Name: price_range, Length: 2000, dtype: int64
```

Split data into training and test data.

```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25)

y_train
```

```
52      3
715     1
650     2
1907    3
1125    2
       ..
1881    2
565     0
121     3
```

```
514     1
5       1
Name: price_range, Length: 1500, dtype: int64
```

```
y_test
```

```
690     3
664     0
557     2
321     3
471     3
       ..
1657    1
370     3
1877    1
1695    2
710     2
Name: price_range, Length: 500, dtype: int64
```

Apply the following models on the training dataset and generate the predicted value for the test dataset

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
logmodel = LogisticRegression()
```

```
logmodel.fit(x_train,y_train)
```

```
LogisticRegression()
```

Predict the price range for test data

```
y1_predict=logmodel.predict(x_test)
```

```
y1_predict
```

```
array([2, 0, 2, 2, 3, 3, 3, 3, 2, 2, 1, 0, 2, 0, 3, 1, 0, 1, 3, 2, 0, 2,
       3, 1, 0, 0, 3, 3, 3, 3, 3, 3, 1, 3, 2, 3, 2, 0, 2, 2, 3, 3, 3, 2,
       0, 0, 2, 3, 2, 0, 0, 1, 1, 0, 1, 0, 2, 3, 2, 0, 2, 1, 3, 1, 2, 1,
       1, 0, 0, 1, 0, 0, 0, 0, 0, 3, 1, 3, 3, 1, 0, 1, 2, 1, 1, 0, 3, 1,
       3, 0, 3, 1, 1, 2, 3, 2, 0, 1, 3, 0, 1, 0, 3, 1, 1, 2, 1, 0, 3, 1,
       3, 1, 1, 0, 3, 1, 0, 2, 0, 2, 0, 2, 1, 3, 1, 1, 0, 3, 3, 1, 0, 1,
       0, 1, 3, 3, 3, 0, 0, 2, 1, 2, 3, 3, 0, 0, 0, 1, 0, 3, 0, 0, 1, 3,
       3, 3, 0, 0, 1, 0, 3, 0, 2, 3, 3, 2, 3, 1, 2, 3, 2, 0, 3, 1, 0, 1,
       3, 0, 3, 2, 3, 3, 2, 3, 3, 3, 2, 3, 0, 1, 2, 2, 3, 0, 0, 2, 2, 3,
       0, 3, 0, 2, 2, 2, 0, 1, 3, 3, 2, 2, 2, 2, 2, 1, 3, 3, 0, 3, 0, 0,
       2, 2, 1, 1, 0, 2, 1, 0, 3, 0, 0, 3, 2, 2, 1, 3, 0, 1, 2, 0, 0, 1,
```

```
       1, 3, 3, 2, 3, 3, 1, 1, 0, 2, 3, 3, 0, 2, 0, 1, 2, 0, 2, 3, 1, 2,
       3, 0, 1, 3, 3, 1, 0, 1, 0, 1, 2, 0, 1, 1, 2, 2, 3, 2, 0, 0, 3, 3,
       3, 2, 1, 2, 0, 3, 0, 2, 1, 0, 0, 3, 1, 3, 1, 1, 3, 2, 1, 3, 1, 1,
       1, 0, 3, 1, 0, 3, 0, 2, 1, 2, 3, 3, 2, 3, 3, 0, 0, 2, 1, 1, 0, 3,
       1, 1, 3, 3, 2, 1, 3, 0, 0, 0, 0, 1, 0, 1, 3, 0, 1, 2, 3, 3, 2, 0,
       0, 3, 3, 3, 3, 1, 3, 1, 1, 2, 2, 0, 2, 0, 1, 1, 1, 1, 2, 0, 3, 2,
       0, 0, 3, 0, 3, 3, 2, 2, 3, 1, 0, 1, 3, 3, 3, 2, 3, 0, 1, 2, 0, 0,
       3, 0, 0, 0, 2, 1, 1, 1, 1, 1, 2, 2, 1, 2, 1, 2, 0, 2, 1, 1, 1, 1,
       0, 1, 1, 2, 1, 1, 3, 2, 0, 3, 2, 0, 3, 3, 0, 0, 0, 1, 3, 0, 2, 1,
       1, 2, 0, 3, 3, 2, 1, 0, 2, 3, 1, 0, 0, 3, 1, 2, 1, 2, 3, 2, 0, 2,
       3, 1, 2, 0, 0, 2, 2, 3, 2, 3, 3, 2, 2, 1, 2, 2, 3, 2, 0, 1, 2, 3,
       2, 0, 1, 2, 3, 1, 3, 1, 2, 2, 0, 1, 2, 2, 1, 2], dtype=int64)
```

```
y_test
```

```
690     3
664     0
557     2
321     3
471     3
       ..
1657    1
370     3
1877    1
1695    2
710     2
Name: price_range, Length: 500, dtype: int64
```

accuracy

```
logmodel.score(x_test,y_test)
```

```
0.604
```

Confusion matrix

```
from sklearn.metrics import confusion_matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test,y1_predict))
```

```
Confusion Matrix:
[[98 37  0  0]
 [28 63 25  7]
 [ 0 20 58 45]
 [ 0  2 34 83]]
```

classification report

```
from sklearn.metrics import classification_report
print("Classification_report: ")
print(classification_report(y_test,y1_predict))
```

```
Classification_report:
              precision    recall  f1-score   support

           0       0.78      0.73      0.75       135
           1       0.52      0.51      0.51       123
           2       0.50      0.47      0.48       123
           3       0.61      0.70      0.65       119

    accuracy                           0.60       500
   macro avg       0.60      0.60      0.60       500
weighted avg       0.61      0.60      0.60       500
```

```
test1 = pd.DataFrame()

test1['price_org'] = y_test

test1['logistic_pred'] = y1_predict

test1
```

```
      price_org  logistic_pred
690           3              2
664           0              0
557           2              2
321           3              2
471           3              3
...         ...            ...
1657          1              1
370           3              2
1877          1              2
1695          2              1
710           2              2

[500 rows x 2 columns]
```

KNN Classification

```
from sklearn.neighbors import KNeighborsClassifier
km=KNeighborsClassifier(n_neighbors=10)
km.fit(x_train,y_train)
```

```
KNeighborsClassifier(n_neighbors=10)
```

Predict the price range for test data

```
y2_predict=km.predict(x_test)
```

```
y2_predict
```

```
array([3, 0, 2, 3, 3, 2, 3, 3, 3, 1, 0, 0, 2, 0, 2, 1, 0, 2, 3, 3, 0, 2,
       1, 1, 0, 0, 2, 3, 2, 2, 2, 2, 1, 3, 2, 3, 1, 0, 3, 0, 1, 3, 3, 3,
       1, 0, 2, 3, 1, 1, 1, 1, 2, 0, 1, 1, 2, 2, 2, 0, 1, 0, 3, 0, 1, 1,
       1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 2, 3, 3, 0, 0, 2, 1, 1, 0, 0, 3, 1,
       2, 0, 2, 0, 1, 3, 3, 3, 0, 1, 3, 0, 1, 0, 3, 1, 0, 2, 1, 0, 3, 2,
       3, 2, 0, 0, 3, 0, 0, 3, 0, 1, 1, 3, 0, 2, 0, 0, 0, 2, 2, 0, 0, 0,
       0, 1, 1, 2, 2, 1, 1, 3, 0, 2, 3, 2, 0, 0, 1, 1, 1, 3, 0, 0, 0, 3,
       3, 3, 1, 0, 0, 0, 3, 0, 1, 2, 2, 2, 3, 1, 2, 3, 2, 0, 3, 2, 0, 2,
       2, 0, 2, 2, 3, 2, 2, 3, 2, 2, 2, 3, 0, 1, 3, 2, 3, 1, 0, 2, 2, 2,
       1, 3, 0, 2, 1, 2, 0, 0, 3, 3, 3, 2, 2, 3, 2, 1, 2, 3, 0, 3, 0, 1,
       2, 2, 1, 3, 0, 2, 1, 1, 3, 0, 1, 2, 2, 3, 1, 2, 0, 2, 3, 0, 0, 1,
       1, 3, 2, 3, 3, 2, 1, 0, 0, 3, 3, 3, 0, 2, 1, 1, 3, 0, 3, 2, 1, 2,
       3, 0, 0, 2, 3, 1, 1, 1, 0, 1, 3, 1, 1, 2, 2, 1, 3, 2, 0, 0, 2, 3,
       2, 2, 1, 2, 0, 3, 0, 2, 1, 0, 0, 2, 1, 2, 0, 1, 3, 1, 1, 2, 0, 0,
       1, 1, 3, 1, 0, 3, 0, 2, 1, 3, 3, 2, 2, 3, 2, 0, 0, 3, 1, 1, 1, 1,
       0, 0, 2, 1, 3, 0, 3, 0, 1, 0, 0, 0, 0, 1, 2, 0, 0, 2, 2, 3, 2, 1,
       1, 3, 3, 2, 2, 0, 3, 1, 2, 2, 1, 0, 1, 1, 0, 0, 1, 1, 3, 0, 3, 1,
       0, 0, 2, 0, 3, 3, 3, 1, 3, 2, 0, 1, 3, 3, 2, 2, 3, 0, 1, 2, 0, 0,
       3, 0, 0, 0, 3, 1, 0, 2, 0, 2, 2, 2, 1, 3, 1, 2, 1, 3, 1, 1, 0, 0,
       0, 1, 1, 1, 0, 1, 3, 2, 0, 2, 3, 1, 1, 2, 0, 0, 0, 0, 3, 0, 2, 1,
       1, 2, 1, 2, 2, 3, 2, 0, 1, 2, 0, 0, 1, 3, 2, 1, 0, 3, 2, 1, 0, 1,
       2, 1, 2, 0, 0, 2, 2, 3, 2, 3, 1, 1, 2, 1, 2, 2, 3, 3, 0, 0, 1, 2,
       2, 1, 0, 3, 3, 1, 3, 2, 3, 3, 1, 2, 3, 1, 2, 2], dtype=int64)
```

accurary

```
km.score(x_test,y_test)
```

```
0.936
```

Confusion matrix

```
from sklearn.metrics import confusion_matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test,y2_predict))
```

```
Confusion Matrix:
[[131   4   0   0]
 [  5 115   3   0]
 [  0   5 114   4]
 [  0   0  11 108]]
```

classification report

```
print("Classification_report: ")
print(classification_report(y_test,y2_predict))
```

```
Classification_report: 
              precision    recall  f1-score   support

           0       0.96      0.97      0.97       135
           1       0.93      0.93      0.93       123
           2       0.89      0.93      0.91       123
           3       0.96      0.91      0.94       119

    accuracy                           0.94       500
   macro avg       0.94      0.93      0.94       500
weighted avg       0.94      0.94      0.94       500
```

```
test2 = pd.DataFrame()

test2['price_org'] = y_test

test2['km_predict'] = y2_predict

test2
```

```
      price_org  km_predict
690           3           3
664           0           0
557           2           2
321           3           3
471           3           3
...         ...         ...
1657          1           2
370           3           3
1877          1           1
1695          2           2
710           2           2

[500 rows x 2 columns]
```

SVM Classifier with linear

```
from sklearn import svm

lin= svm.SVC(kernel='linear', C=1.0)

lin.fit(x_train,y_train)
```

```
SVC(kernel='linear')
```

Predict the price range for test data

```
y3_predict=lin.predict(x_test)
```

```
y3_predict
```

```
array([3, 0, 2, 3, 3, 2, 3, 3, 2, 2, 0, 0, 2, 0, 3, 1, 0, 2, 3, 3, 0, 2,
       1, 1, 0, 0, 3, 3, 2, 2, 2, 2, 1, 3, 2, 3, 1, 0, 3, 1, 1, 3, 3, 3,
       1, 0, 2, 3, 1, 0, 1, 2, 2, 0, 1, 1, 2, 2, 2, 0, 1, 0, 3, 0, 1, 1,
       1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 2, 3, 3, 0, 0, 2, 1, 1, 0, 0, 3, 1,
       3, 0, 2, 0, 1, 3, 3, 3, 0, 2, 3, 0, 1, 0, 3, 1, 0, 2, 1, 0, 3, 2,
       3, 2, 0, 0, 3, 0, 0, 3, 0, 1, 1, 3, 0, 2, 0, 0, 0, 2, 2, 0, 0, 1,
       0, 1, 1, 2, 3, 1, 1, 2, 0, 2, 3, 2, 0, 0, 1, 1, 1, 3, 0, 0, 0, 3,
       3, 3, 1, 0, 0, 0, 3, 0, 1, 3, 2, 2, 3, 1, 2, 3, 2, 0, 3, 2, 0, 2,
       2, 0, 2, 2, 3, 2, 2, 3, 2, 2, 3, 3, 0, 1, 3, 2, 3, 1, 0, 2, 2, 2,
       1, 3, 0, 2, 1, 2, 0, 0, 3, 3, 3, 2, 2, 3, 2, 1, 2, 3, 0, 3, 0, 1,
       2, 2, 2, 3, 0, 2, 1, 1, 3, 0, 1, 2, 2, 3, 1, 2, 0, 2, 3, 0, 0, 1,
       1, 3, 2, 3, 3, 2, 1, 0, 0, 3, 3, 3, 0, 2, 1, 1, 2, 0, 3, 3, 1, 2,
       3, 0, 0, 2, 3, 1, 1, 1, 0, 1, 3, 1, 1, 2, 2, 1, 2, 2, 0, 0, 2, 3,
       3, 2, 1, 1, 0, 3, 0, 2, 1, 0, 0, 2, 1, 2, 0, 1, 3, 2, 1, 2, 1, 0,
       1, 1, 3, 1, 0, 3, 0, 2, 1, 3, 3, 2, 2, 3, 2, 0, 0, 3, 1, 1, 1, 2,
       0, 0, 2, 1, 3, 0, 3, 0, 1, 0, 0, 1, 0, 1, 2, 0, 0, 2, 2, 3, 2, 1,
       1, 3, 3, 2, 3, 0, 3, 1, 3, 2, 1, 0, 1, 1, 0, 0, 1, 1, 2, 0, 3, 1,
       0, 0, 2, 0, 3, 3, 3, 1, 3, 2, 0, 1, 3, 3, 3, 2, 3, 0, 1, 2, 0, 0,
       3, 0, 0, 0, 3, 2, 0, 2, 0, 2, 2, 2, 1, 3, 1, 2, 1, 2, 1, 1, 1, 0,
       0, 1, 1, 1, 0, 1, 3, 2, 0, 2, 3, 1, 1, 2, 0, 0, 0, 0, 3, 0, 2, 1,
       1, 2, 1, 2, 2, 3, 2, 0, 1, 2, 0, 0, 1, 3, 2, 1, 0, 3, 2, 1, 0, 1,
       2, 1, 2, 0, 0, 2, 2, 3, 2, 3, 1, 1, 2, 1, 2, 2, 3, 3, 0, 0, 1, 2,
       2, 0, 0, 3, 3, 1, 3, 2, 3, 3, 1, 1, 3, 1, 2, 2], dtype=int64)
```

accurary

```
lin.score(x_test,y_test)
```

```
0.978
```

Confusion matrix

```
print("Confusion Matrix:")
print(confusion_matrix(y_test,y3_predict))
```

```
Confusion Matrix:
[[132   3   0   0]
 [  1 119   3   0]
 [  0   0 122   1]
 [  0   0   3 116]]
```

classification report

```
print("Classification_report: ")
print(classification_report(y_test,y3_predict))
```

```
Classification_report:
              precision    recall  f1-score   support

           0       0.99      0.98      0.99       135
           1       0.98      0.97      0.97       123
           2       0.95      0.99      0.97       123
           3       0.99      0.97      0.98       119

    accuracy                           0.98       500
   macro avg       0.98      0.98      0.98       500
weighted avg       0.98      0.98      0.98       500
```

```
test3=pd.DataFrame()

test3['price_org']=y_test

test3['svm_predict'] = y3_predict

test3
```

```
      price_org  svm_predict
690           3            3
664           0            0
557           2            2
321           3            3
471           3            3
...         ...          ...
1657          1            1
370           3            3
1877          1            1
1695          2            2
710           2            2

[500 rows x 2 columns]
```

SVM Classifier with rbf kernel

```
from sklearn.svm import SVC
rbfs = SVC(kernel='rbf', probability=True)

rbfs.fit(x_train, y_train)
```

```
SVC(probability=True)
```

Predict the price range for test data

```
y4_predict=rbfs.predict(x_test)
```

```
y4_predict
```

```
array([3, 0, 2, 3, 3, 2, 3, 3, 3, 2, 0, 0, 2, 0, 2, 1, 0, 2, 3, 3, 0, 2,
       1, 1, 0, 0, 3, 3, 2, 2, 2, 2, 1, 3, 2, 3, 1, 0, 3, 1, 1, 3, 3, 3,
       1, 0, 2, 3, 1, 0, 1, 2, 2, 0, 1, 1, 2, 2, 2, 0, 1, 0, 3, 0, 1, 1,
       1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 2, 3, 3, 0, 0, 2, 1, 1, 0, 0, 3, 1,
       3, 0, 2, 0, 1, 3, 3, 3, 0, 1, 3, 0, 1, 0, 3, 1, 0, 2, 1, 0, 3, 2,
       3, 2, 0, 0, 3, 0, 0, 3, 0, 1, 1, 3, 0, 2, 0, 0, 0, 2, 2, 0, 0, 1,
       0, 1, 1, 2, 3, 1, 1, 3, 0, 2, 3, 2, 0, 0, 1, 1, 1, 3, 0, 0, 0, 3,
       3, 3, 1, 0, 0, 0, 3, 0, 1, 3, 2, 2, 3, 1, 2, 3, 2, 0, 3, 1, 0, 2,
       2, 0, 2, 2, 3, 2, 3, 3, 2, 2, 3, 3, 0, 1, 3, 2, 3, 1, 0, 2, 2, 2,
       1, 3, 0, 2, 1, 3, 0, 0, 3, 3, 3, 2, 2, 3, 2, 1, 2, 3, 0, 3, 0, 1,
       2, 3, 1, 3, 0, 2, 1, 1, 3, 0, 1, 2, 2, 3, 1, 2, 0, 2, 3, 0, 0, 1,
       1, 3, 2, 3, 3, 2, 1, 0, 0, 3, 3, 3, 0, 2, 1, 1, 3, 0, 3, 3, 1, 2,
       3, 0, 0, 2, 3, 1, 1, 1, 0, 1, 3, 1, 1, 2, 2, 1, 2, 2, 0, 0, 2, 3,
       3, 2, 1, 1, 0, 3, 0, 2, 1, 0, 0, 2, 1, 2, 0, 1, 3, 1, 1, 2, 1, 0,
       1, 1, 3, 1, 0, 3, 0, 2, 1, 3, 3, 2, 2, 3, 2, 0, 0, 3, 1, 1, 1, 1,
       0, 0, 2, 1, 3, 0, 3, 0, 1, 0, 0, 1, 0, 1, 2, 0, 0, 2, 2, 3, 2, 1,
       1, 3, 3, 2, 2, 0, 3, 1, 3, 2, 1, 0, 1, 1, 0, 0, 1, 1, 3, 0, 3, 1,
       0, 0, 2, 0, 3, 3, 3, 1, 3, 2, 0, 1, 3, 3, 3, 2, 3, 0, 1, 2, 0, 0,
       3, 0, 0, 0, 3, 2, 0, 2, 0, 2, 3, 2, 1, 3, 1, 2, 1, 3, 1, 1, 1, 0,
       0, 1, 1, 1, 0, 1, 3, 2, 0, 2, 3, 1, 1, 2, 0, 0, 0, 0, 3, 0, 2, 1,
       1, 2, 1, 2, 2, 3, 2, 0, 1, 2, 0, 0, 1, 3, 2, 1, 0, 3, 2, 1, 0, 1,
       2, 1, 2, 0, 0, 2, 3, 3, 2, 3, 1, 2, 2, 1, 2, 2, 3, 3, 0, 0, 1, 2,
       2, 0, 0, 3, 3, 1, 3, 2, 3, 3, 1, 1, 3, 1, 2, 2], dtype=int64)
```

accuracy

```
rbfs.score(x_test,y_test)
```

```
0.954
```

Confusion matrix

```
print("Confusion Matrix:")
print(confusion_matrix(y_test,y4_predict))
```

```
Confusion Matrix:
[[132   3   0   0]
 [  1 119   3   0]
 [  0   4 110   9]
 [  0   0   3 116]]
```

classification report

```
print("Classification_report: ")
print(classification_report(y_test,y4_predict))
```

```
Classification_report:
              precision    recall  f1-score   support

           0       0.99      0.98      0.99       135
           1       0.94      0.97      0.96       123
           2       0.95      0.89      0.92       123
           3       0.93      0.97      0.95       119

    accuracy                           0.95       500
   macro avg       0.95      0.95      0.95       500
weighted avg       0.95      0.95      0.95       500
```

```
test4=pd.DataFrame()

test4['orginalprice']=y_test

test4['rbf_predict']=y4_predict

test4
```

```
      orginalprice  rbf_predict
690              3            3
664              0            0
557              2            2
321              3            3
471              3            3
...            ...          ...
1657             1            1
370              3            3
1877             1            1
1695             2            2
710              2            2

[500 rows x 2 columns]
```

==>Report the model with the best accuracy.

from the above four models

a)Logistic Regression---0.646=64%

b)KNN Classification---0.916=91%

c)SVM Classifier with linear---0.966=96%

d)SVM Classifier with rbf kernel---0.944=94%

Therefore SVM Classifier with linear model scoring high accuracy

so SVM Classifier with linear model is a best accuracy model