Problem statement: Create a classification model to predict whether price range of mobile based on certain specifications

In [ ]:
```
SUBMITTED BY : SYED FAIZAN UDDIN
```

In [83]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
import scipy.stats as stats
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
```

In [84]:
```
df = pd.read_csv('mobile_price_range_data.csv')
```

In [85]:
```
df.head(20)
```

Out[85]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	px_height	px_width	ram	sc_h
0	842	0	2.2	0	1	0	7	0.6	188	2	...	20	756	2549	9
1	1021	1	0.5	1	0	1	53	0.7	136	3	...	905	1988	2631	17
2	563	1	0.5	1	2	1	41	0.9	145	5	...	1263	1716	2603	11
3	615	1	2.5	0	0	0	10	0.8	131	6	...	1216	1786	2769	16
4	1821	1	1.2	0	13	1	44	0.6	141	2	...	1208	1212	1411	8
5	1859	0	0.5	1	3	0	22	0.7	164	1	...	1004	1654	1067	17
6	1821	0	1.7	0	4	1	10	0.8	139	8	...	381	1018	3220	13
7	1954	0	0.5	1	0	0	24	0.8	187	4	...	512	1149	700	16
8	1445	1	0.5	0	0	0	53	0.7	174	7	...	386	836	1099	17
9	509	1	0.6	1	2	1	9	0.1	93	5	...	1137	1224	513	19
10	769	1	2.9	1	0	0	9	0.1	182	5	...	248	874	3946	5
11	1520	1	2.2	0	5	1	33	0.5	177	8	...	151	1005	3826	14
12	1815	0	2.8	0	2	0	33	0.6	159	4	...	607	748	1482	18
13	803	1	2.1	0	7	0	17	1.0	198	4	...	344	1440	2680	7
14	1866	0	0.5	0	13	1	52	0.7	185	1	...	356	563	373	14
15	775	0	1.0	0	3	0	46	0.7	159	2	...	862	1864	568	17
16	838	0	0.5	0	1	1	13	0.1	196	8	...	984	1850	3554	10
17	595	0	0.9	1	7	1	23	0.1	121	3	...	441	810	3752	10
18	1131	1	0.5	1	11	0	49	0.6	101	5	...	658	878	1835	19
19	682	1	0.5	0	4	0	19	1.0	121	4	...	902	1064	2337	11

20 rows × 21 columns

handle null values

In [86]:
```
df.isnull().sum()
```

Out[86]:
```
battery_power    0
blue             0
clock_speed      0
dual_sim         0
fc               0
four_g           0
int_memory       0
m_dep            0
mobile_wt        0
n_cores          0
pc               0
px_height        0
px_width         0
ram              0
sc_h             0
sc_w             0
talk_time        0
three_g          0
touch_screen     0
wifi             0
price_range      0
dtype: int64
```

In [5]:
```
df.isnull().sum().sum()
```

Out[5]:
```
0
```

In [6]:
```
df.shape
```

Out[6]:
```
(2000, 21)
```

In [7]:
```
df.columns
```

Out[7]:
```
Index(['battery_power', 'blue', 'clock_speed', 'dual_sim', 'fc', 'four_g',
       'int_memory', 'm_dep', 'mobile_wt', 'n_cores', 'pc', 'px_height',
       'px_width', 'ram', 'sc_h', 'sc_w', 'talk_time', 'three_g',
       'touch_screen', 'wifi', 'price_range'],
      dtype='object')
```

In [8]:
```
df.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
```

In [9]:
```
df.describe()
```

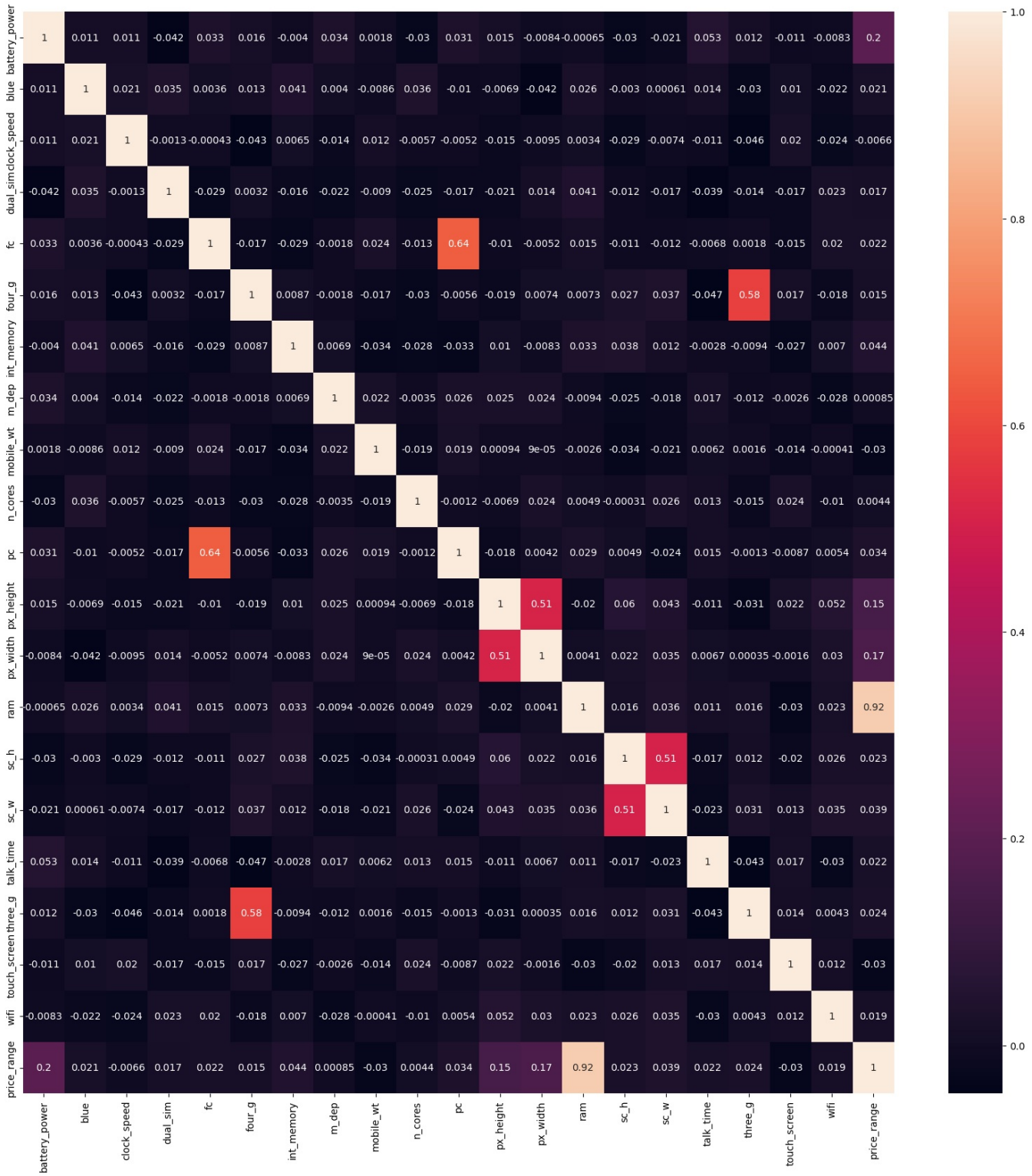
Out[9]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	.
count	2000.000000	2000.0000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	.
mean	1238.518500	0.4950	1.522250	0.509500	4.309500	0.521500	32.046500	0.501750	140.249000	4.520500	.
std	439.418206	0.5001	0.816004	0.500035	4.341444	0.499662	18.145715	0.288416	35.399655	2.287837	.
min	501.000000	0.0000	0.500000	0.000000	0.000000	0.000000	2.000000	0.100000	80.000000	1.000000	.
25%	851.750000	0.0000	0.700000	0.000000	1.000000	0.000000	16.000000	0.200000	109.000000	3.000000	.
50%	1226.000000	0.0000	1.500000	1.000000	3.000000	1.000000	32.000000	0.500000	141.000000	4.000000	.
75%	1615.250000	1.0000	2.200000	1.000000	7.000000	1.000000	48.000000	0.800000	170.000000	7.000000	.
max	1998.000000	1.0000	3.000000	1.000000	19.000000	1.000000	64.000000	1.000000	200.000000	8.000000	.

8 rows × 21 columns

In [10]:
```
plt.figure(figsize=(20,20))
sns.heatmap(df.corr(), annot=True)
```

Out[10]: <AxesSubplot:>



In [11]:
```
df['price_range'].value_counts()
```

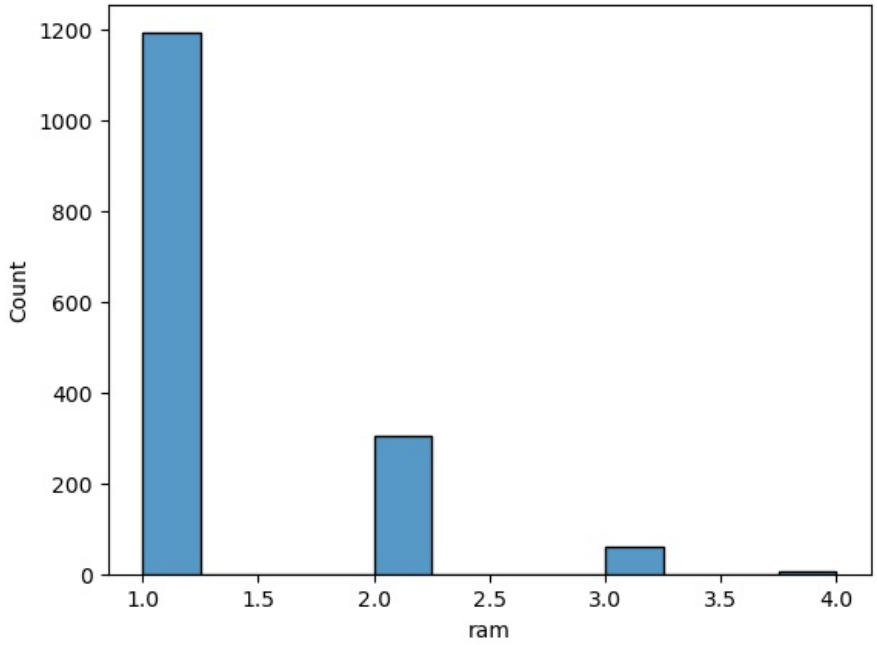
Out[11]:
```
1    500
2    500
3    500
0    500
Name: price_range, dtype: int64
```

In [12]:
```
df['ram'].value_counts()
```

Out[12]:
```
1464    4
3142    4
2610    4
2227    4
1229    4
       ..
2312    1
2167    1
3508    1
297     1
3919    1
Name: ram, Length: 1562, dtype: int64
```

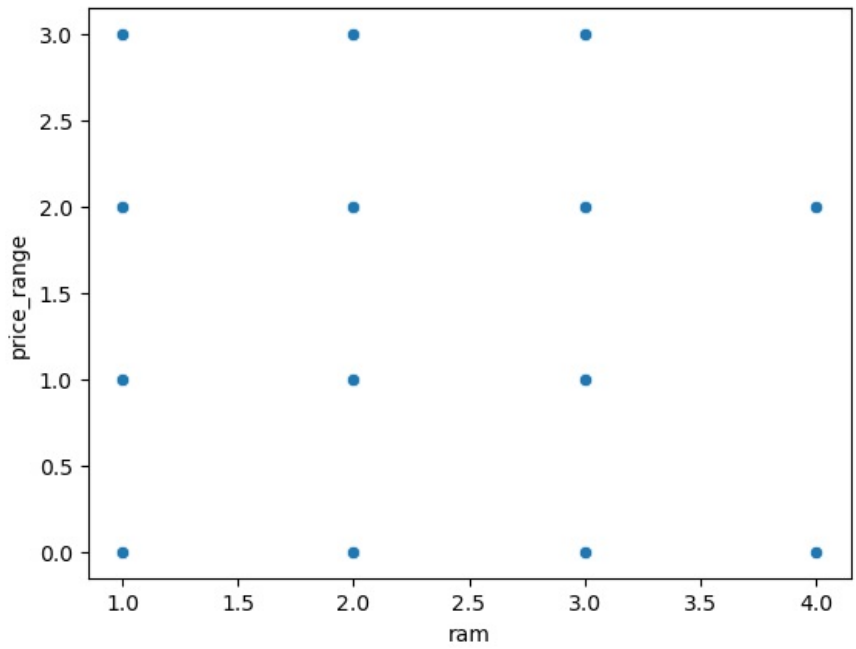
In [13]:
```
sns.histplot(x=df['ram'].value_counts())
```

Out[13]:
```
<AxesSubplot:xlabel='ram', ylabel='Count'>
```



In [14]:
```
sns.scatterplot(x=df['ram'].value_counts(),y=df['price_range'])
```

Out[14]:
```
<AxesSubplot:xlabel='ram', ylabel='price_range'>
```

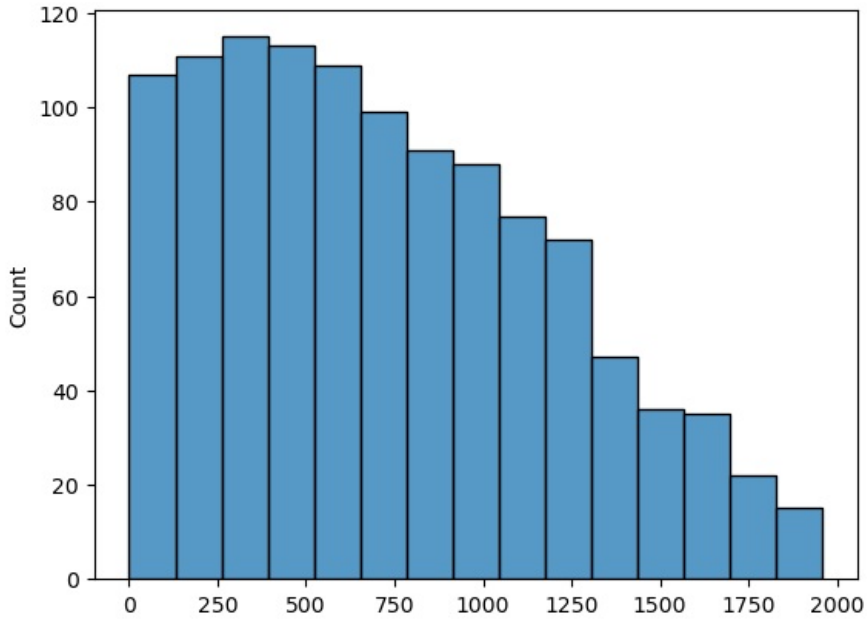


In [15]:
```
df['px_height'].value_counts().head(10)
```

Out[15]:
```
347    7
179    6
371    6
275    6
674    5
286    5
42     5
211    5
649    5
398    5
Name: px_height, dtype: int64
```

In [16]:
```
sns.histplot(df['px_height'].value_counts().keys(), bins=15)
```

Out[16]:
```
<AxesSubplot:ylabel='Count'>
```

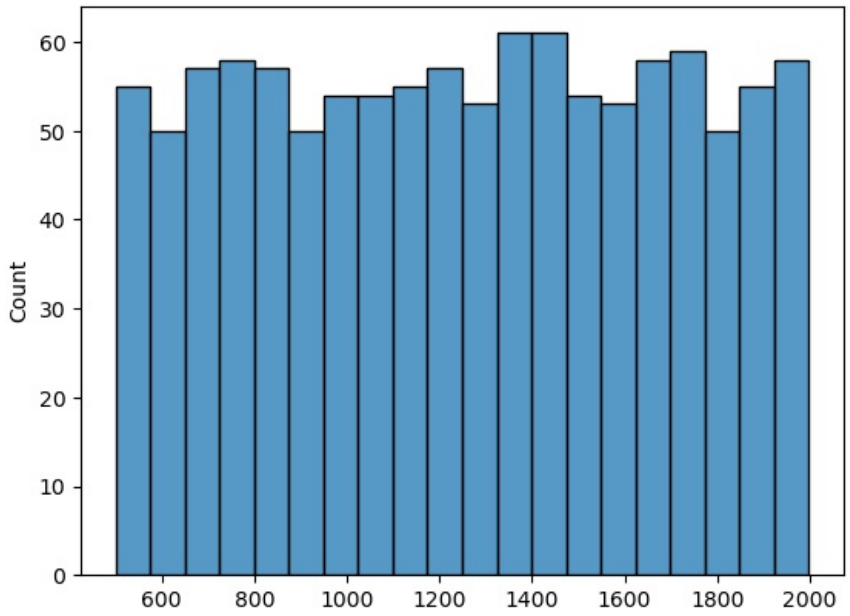


In [17]:
```
df['px_width'].value_counts()
```

Out[17]:
```
874     7
1247    7
1383    6
1463    6
1469    6
       ..
1125    1
1367    1
1569    1
1481    1
1632    1
Name: px_width, Length: 1109, dtype: int64
```

In [18]:
```
sns.histplot(df['px_width'].value_counts().keys(), bins=20)
```

Out[18]:
```
<AxesSubplot:ylabel='Count'>
```

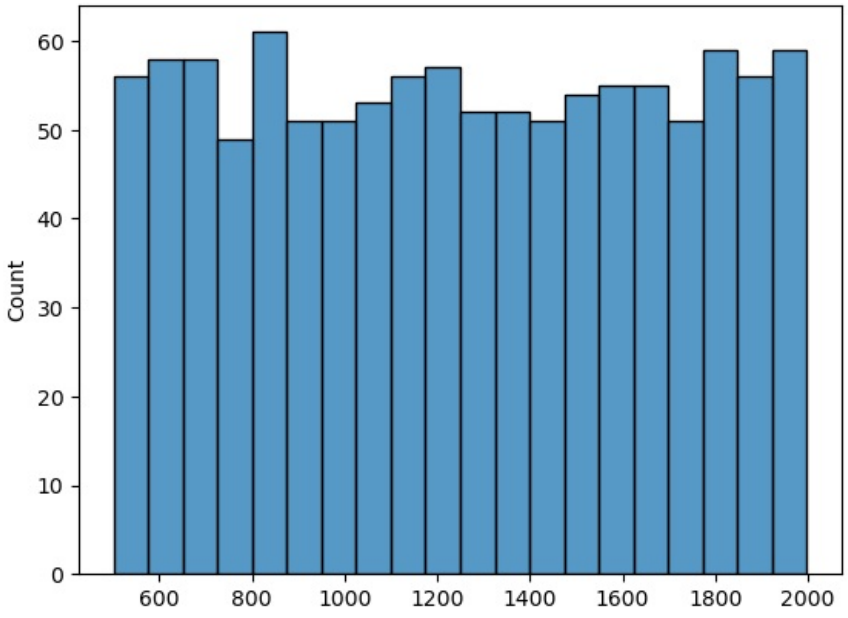


In [19]:
```
df['battery_power'].value_counts()
```

Out[19]:
```
1872    6
618     6
1589    6
1715    5
1807    5
       ..
660     1
1452    1
1005    1
1372    1
858     1
Name: battery_power, Length: 1094, dtype: int64
```

In [20]:
```
sns.histplot(df['battery_power'].value_counts().keys(), bins=20)
```

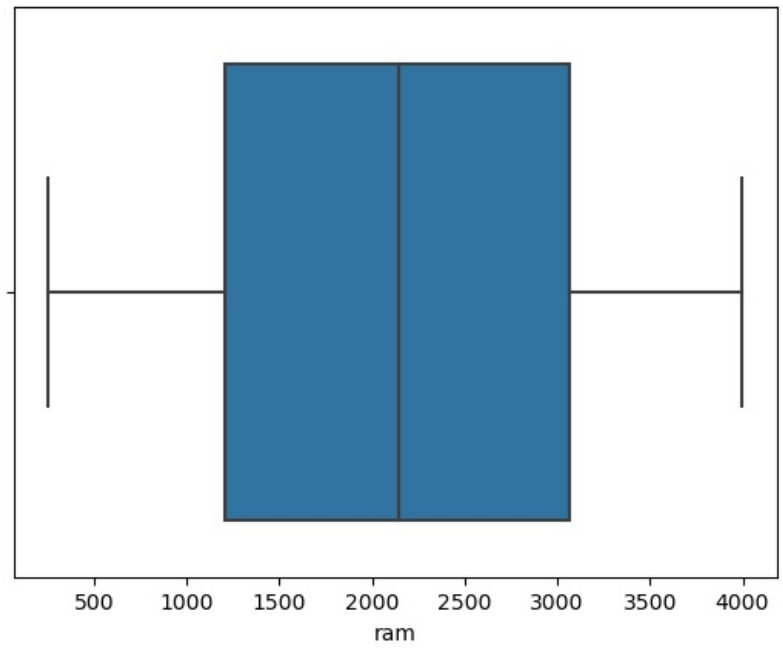
Out[20]:
```
<AxesSubplot:ylabel='Count'>
```



In [21]:
```
sns.boxplot(df['ram'])
```

```
C:\Users\shaik\python\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable a
s a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arg
uments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

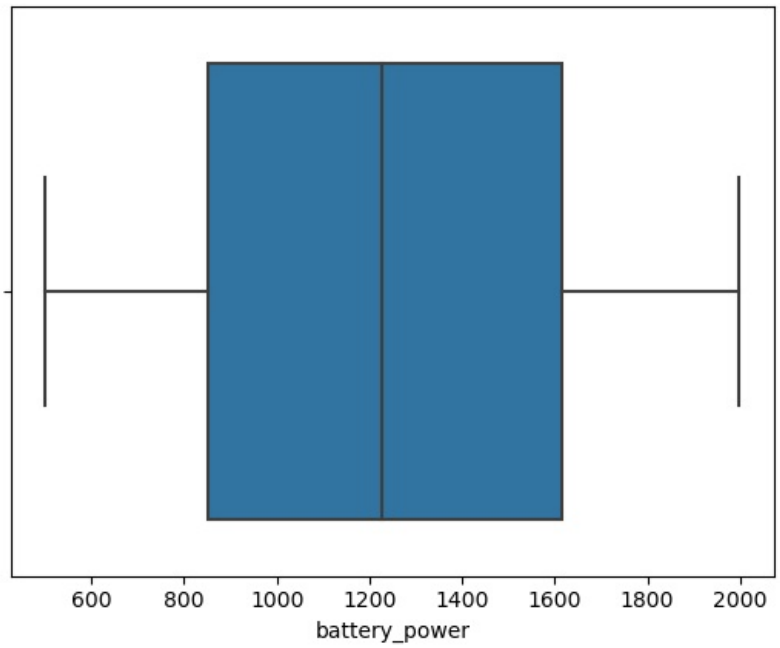
Out[21]: <AxesSubplot:xlabel='ram'>



In [22]:
```
sns.boxplot(df['battery_power'])
```

```
C:\Users\shaik\python\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable a
s a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arg
uments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

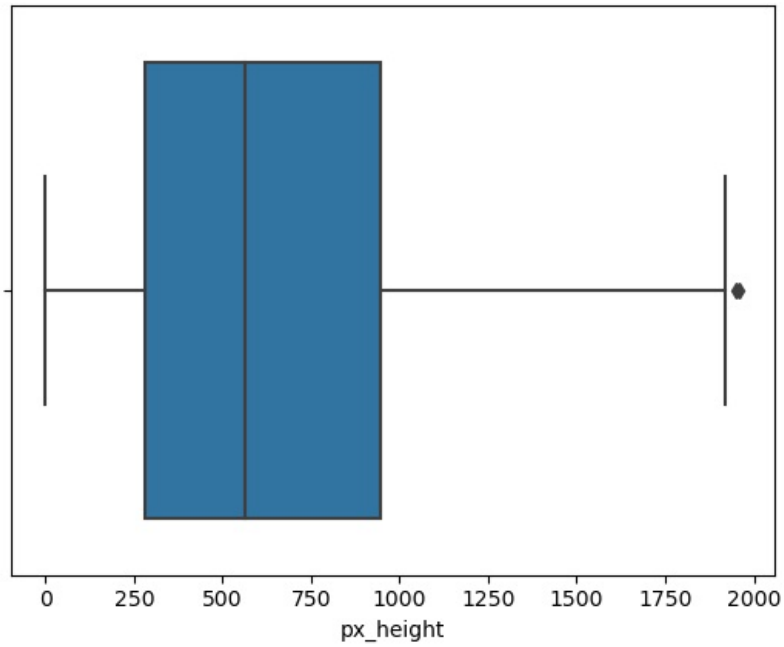
Out[22]: <AxesSubplot:xlabel='battery_power'>



In [23]:
```
sns.boxplot(df['px_height'])
```

```
C:\Users\shaik\python\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable a
s a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arg
uments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

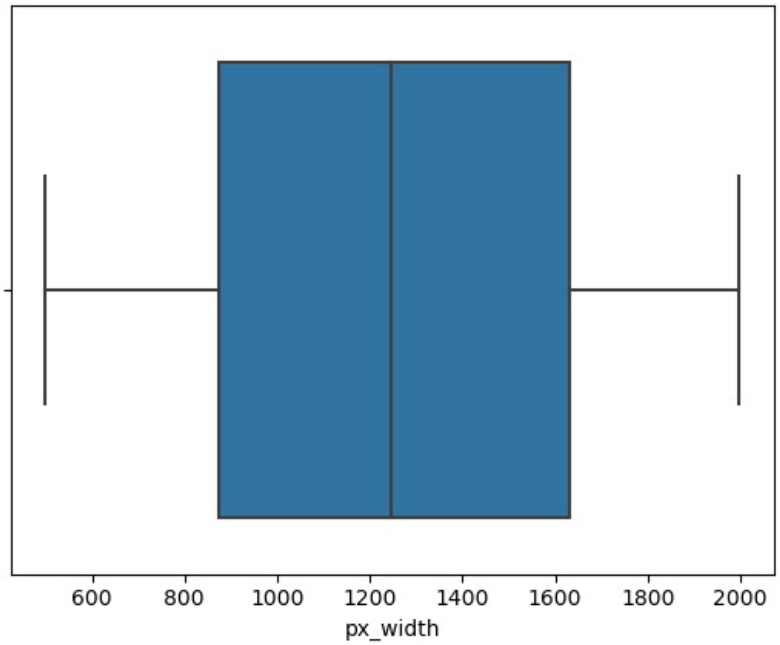
Out[23]: <AxesSubplot:xlabel='px_height'>



In [24]:
```
sns.boxplot(df['px_width'])
```

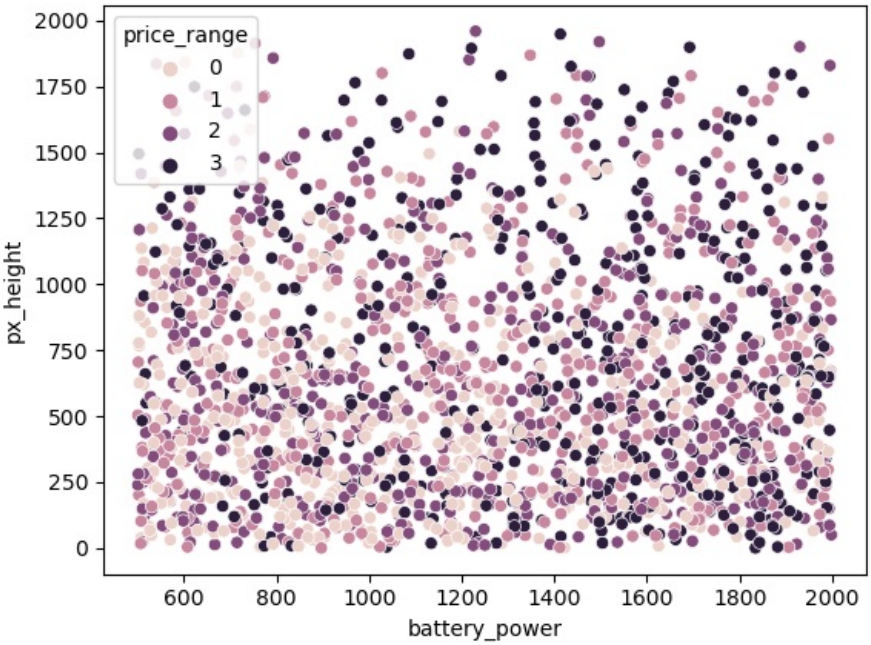
```
C:\Users\shaik\python\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable a
s a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arg
uments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[24]: <AxesSubplot:xlabel='px_width'>



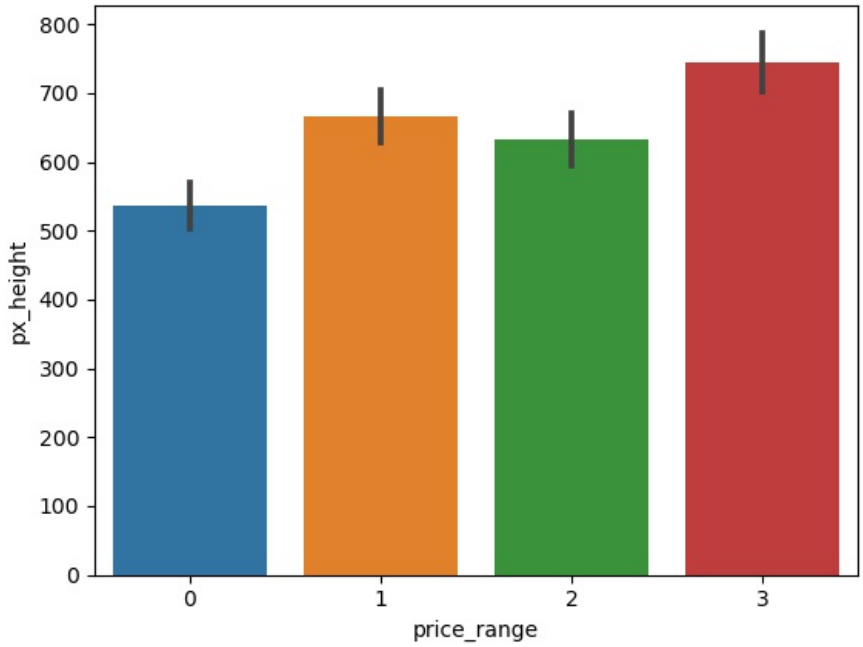
In [25]:
```
sns.scatterplot(x=df['battery_power'],y=df['px_height'],hue=df['price_range'])
```

Out[25]:
```
<AxesSubplot:xlabel='battery_power', ylabel='px_height'>
```



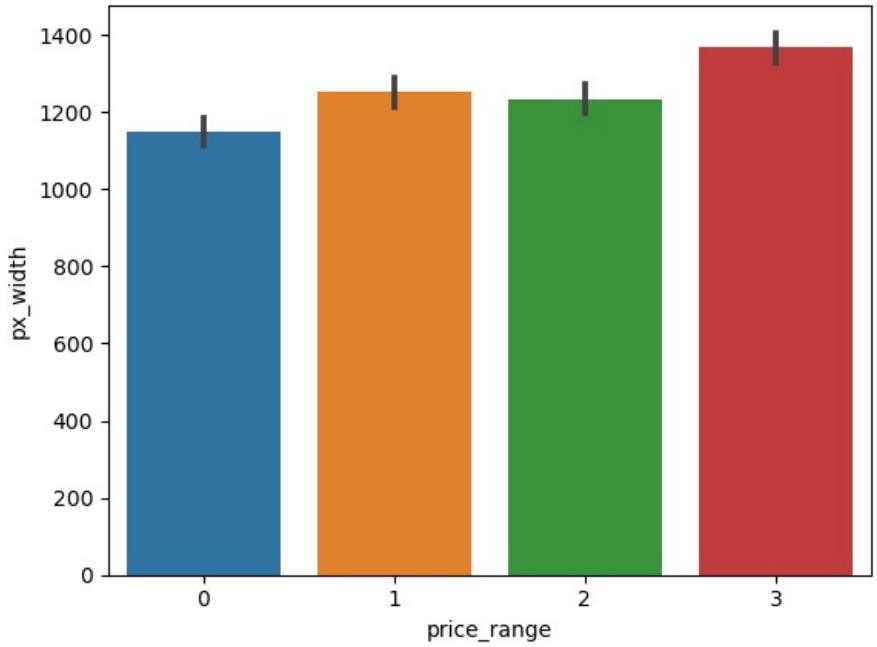
In [26]:
```
sns.barplot(x=df['price_range'], y = df['px_height'])
```

Out[26]:
```
<AxesSubplot:xlabel='price_range', ylabel='px_height'>
```



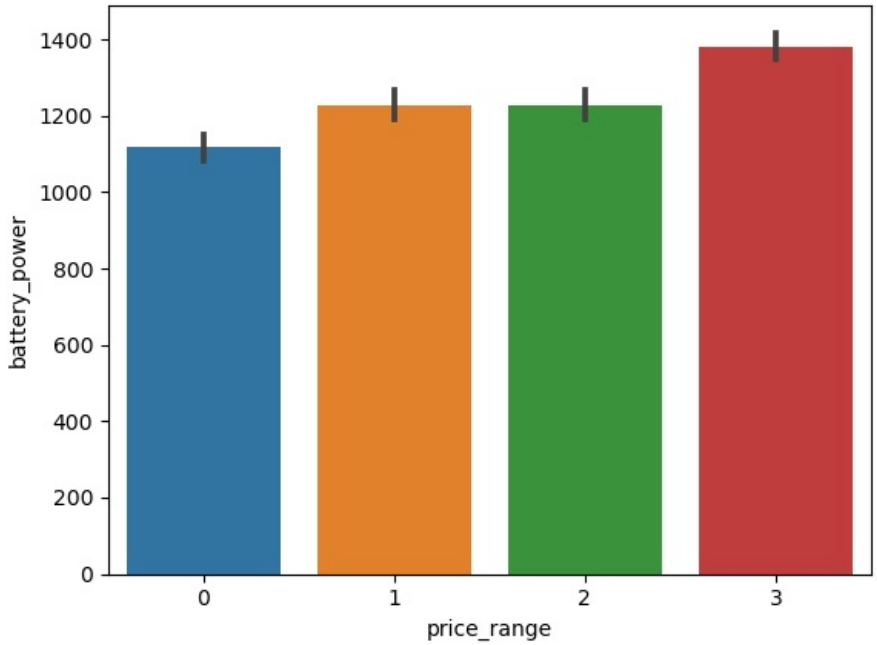
In [27]:
```
sns.barplot(x=df['price_range'], y = df['px_width'])
```

Out[27]:
```
<AxesSubplot:xlabel='price_range', ylabel='px_width'>
```



In [28]:
```
sns.barplot(x=df['price_range'], y = df['battery_power'])
```

Out[28]:
```
<AxesSubplot:xlabel='price_range', ylabel='battery_power'>
```



In [29]:
```
df.groupby(["price_range"]).count()
```

Out[29]:

price_range	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	pc	px_height	px_width	r
0	500	500	500	500	500	500	500	500	500	500	500	500	500	5
1	500	500	500	500	500	500	500	500	500	500	500	500	500	5
2	500	500	500	500	500	500	500	500	500	500	500	500	500	5
3	500	500	500	500	500	500	500	500	500	500	500	500	500	5

In [30]:
```
df1 = df.drop([1960], axis=0)
```

In [31]:
```
df.head()
```

Out[31]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	px_height	px_width	ram	sc_h	s
0	842	0	2.2	0	1	0	7	0.6	188	2	...	20	756	2549	9	
1	1021	1	0.5	1	0	1	53	0.7	136	3	...	905	1988	2631	17	
2	563	1	0.5	1	2	1	41	0.9	145	5	...	1263	1716	2603	11	
3	615	1	2.5	0	0	0	10	0.8	131	6	...	1216	1786	2769	16	
4	1821	1	1.2	0	13	1	44	0.6	141	2	...	1208	1212	1411	8	

5 rows × 21 columns

In [32]:
```
print(df.shape)
print(df1.shape)
```

```
(2000, 21)
(1999, 21)
```

In [33]:
```
x = df1[['battery_power','px_height','px_width','ram']]
y = df1['price_range']
```

In [34]:
```
x
```

Out[34]:

	battery_power	px_height	px_width	ram
0	842	20	756	2549
1	1021	905	1988	2631
2	563	1263	1716	2603
3	615	1216	1786	2769
4	1821	1208	1212	1411
...	...	...	...	...
1995	794	1222	1890	668
1996	1965	915	1965	2032
1997	1911	868	1632	3057
1998	1512	336	670	869
1999	510	483	754	3919

1999 rows × 4 columns

Splitting data between training and testing set

In [ ]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.25,random_state=42)
```

In [36]:
```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(1499, 4)
(500, 4)
(1499,)
(500,)
```

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

a) Logistic Regression

In [39]:
```
m1 = LogisticRegression()
m1.fit(x_train,y_train)
y_pred = m1.predict(x_test)
print("Training Score:",m1.score(x_train,y_train))
print("Testing Score:",m1.score(x_test,y_test))
```

```
Training Score: 0.9606404269513009
Testing Score: 0.962
C:\Users\shaik\python\lib\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

In [40]:
```
print("Confusion Matrix:")
print(confusion_matrix(y_test,y_pred))
```

```
Confusion Matrix:
[[114   3   0   0]
 [  0 126   0   0]
 [  0   5 121   5]
 [  0   0   6 120]]
```

In [41]:
```
print("Classification_report: ")
print(classification_report(y_test,y_pred))
```

```
Classification_report:
              precision    recall  f1-score   support

           0       1.00      0.97      0.99       117
           1       0.94      1.00      0.97       126
           2       0.95      0.92      0.94       131
           3       0.96      0.95      0.96       126

    accuracy                           0.96       500
   macro avg       0.96      0.96      0.96       500
weighted avg       0.96      0.96      0.96       500
```

In [42]:
```
test1 = pd.DataFrame()
```

In [43]:
```
test1['price_org'] = y_test
```

In [44]:
```
test1['logistic_pred'] = y_pred
```

In [45]:
```
test1
```

Out[45]:

	price_org	logistic_pred
256	0	0
352	0	0
298	0	0
581	3	3
1288	1	1
...	...	...
1613	2	2
692	3	3
1553	0	0
931	2	2
1682	1	1

500 rows × 2 columns

In [80]:
```
m1.score(x_test,y_test)
```

Out[80]:
```
0.962
```

b) KNN Classification

In [46]:
```
m2 = KNeighborsClassifier(n_neighbors=21)
m2.fit(x_train,y_train)
```

Out[46]:
```
KNeighborsClassifier(n_neighbors=21)
```

In [47]:
```
y_predkn = m1.predict(x_test)
print("Training Score : ",m1.score(x_train,y_train))
print("Testing Score : ",m1.score(x_test,y_test))
```

```
Training Score :  0.9606404269513009
Testing Score :  0.962
```

In [48]:
```
matrix = confusion_matrix(y_test,y_predkn)# x_test goes into the rows
print(matrix)
```

```
[[114   3   0   0]
 [  0 126   0   0]
 [  0   5 121   5]
 [  0   0   6 120]]
```

In [49]:
```
print(classification_report(y_test,y_predkn))
```

```
              precision    recall  f1-score   support

           0       1.00      0.97      0.99       117
           1       0.94      1.00      0.97       126
           2       0.95      0.92      0.94       131
           3       0.96      0.95      0.96       126

    accuracy                           0.96       500
   macro avg       0.96      0.96      0.96       500
weighted avg       0.96      0.96      0.96       500
```

In [50]:
```
test1['kn_pred'] = y_predkn
test1
```

Out[50]:

	price_org	logistic_pred	kn_pred
256	0	0	0
352	0	0	0
298	0	0	0
581	3	3	3
1288	1	1	1
...	...	...	...
1613	2	2	2
692	3	3	3
1553	0	0	0
931	2	2	2
1682	1	1	1

500 rows × 3 columns

In [82]:
```
m2.score(x_test,y_test)
```

```
C:\Users\shaik\python\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other r
eduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it ac
ts along. In SciPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `a
xis` over which the statistic is taken will be eliminated, and the value None will no longer be accepted. Set `
keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

Out[82]:
```
0.916
```

c) SVM Classifier with linear and rbf kernel

In [51]:
```
s1 = SVC(kernel='linear',C=1)
s1.fit(x_train,y_train)
```

Out[51]:
```
SVC(C=1, kernel='linear')
```

In [52]:
```
y_predsvm = s1.predict(x_test)
```

In [53]:
```
print("Training Score : ",s1.score(x_train,y_train))
print("Testing Score : ",s1.score(x_test,y_test))
```

```
Training Score :  0.961974649766511
Testing Score :  0.966
```

In [54]:
```
matrix = confusion_matrix(y_test,y_predsvm)
print(matrix)
```

```
[[115   2   0   0]
 [  0 126   0   0]
 [  0   5 122   4]
 [  0   0   6 120]]
```

In [55]:
```
print(classification_report(y_test,y_predsvm))
```

```
              precision    recall  f1-score   support

           0       1.00      0.98      0.99       117
           1       0.95      1.00      0.97       126
           2       0.95      0.93      0.94       131
           3       0.97      0.95      0.96       126

    accuracy                           0.97       500
   macro avg       0.97      0.97      0.97       500
weighted avg       0.97      0.97      0.97       500
```

In [76]:
```
test1['svm_pred'] = y_predsvm
test1
```

Out[76]:

	price_org	logistic_pred	kn_pred	svm_pred	rbf_pred
256	0	0	0	0	0
352	0	0	0	0	0
298	0	0	0	0	0
581	3	3	3	3	3
1288	1	1	1	1	1
...	...	...	...	...	...
1613	2	2	2	2	2
692	3	3	3	3	3
1553	0	0	0	0	0
931	2	2	2	2	2
1682	1	1	1	1	1

500 rows × 5 columns

In [78]:
```
s1.score(x_test,y_test)
```

Out[78]:
```
0.966
```

In [57]:
```
s2 = SVC(kernel='rbf',gamma=0.00001,C=10)
s2.fit(x_train,y_train)
```

Out[57]:
```
SVC(C=10, gamma=1e-05)
```

In [67]:
```
y_predrbf = s2.predict(x_test)
y
```

Out[67]:
```
0       1
1       2
2       2
3       2
4       1
       ..
1995    0
1996    2
1997    3
1998    0
1999    3
Name: price_range, Length: 1999, dtype: int64
```

In [59]:
```
print("Training Score : ",s1.score(x_train,y_train))
print("Testing Score : ",s1.score(x_test,y_test))
```

```
Training Score :  0.961974649766511
Testing Score :  0.966
```

In [60]:
```
matrix = confusion_matrix(y_test,y_predrbf)# x_test goes into the rows
print(matrix)
```

```
[[109   8   0   0]
 [  3 119   4   0]
 [  0   6 119   6]
 [  0   0   9 117]]
```

In [61]:
```
print(classification_report(y_test,y_predrbf))
```

```
              precision    recall  f1-score   support

           0       0.97      0.93      0.95       117
           1       0.89      0.94      0.92       126
           2       0.90      0.91      0.90       131
           3       0.95      0.93      0.94       126

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

In [62]:
```
test1.head(20)
```

Out[62]:

	price_org	logistic_pred	kn_pred	svm_pred
256	0	0	0	0
352	0	0	0	0
298	0	0	0	0
581	3	3	3	3
1288	1	1	1	1
1765	2	2	2	2
420	1	1	1	1
1587	1	1	1	1
65	3	3	3	3
1611	2	2	2	2
56	0	0	0	0
1998	0	0	0	0
1117	2	2	2	2
582	0	0	0	0
1232	2	2	2	2
316	1	1	1	1
744	2	2	2	2
128	3	3	3	3
1442	1	1	1	1
1512	3	3	3	3

In [79]:
```
s2.score(x_test,y_test)
```

Out[79]:
```
0.928
```

In [71]:
```

```

```
print svm accuracy: 0.95
```

d) Decision Tree Classifier

In [64]:
```
from sklearn.tree import DecisionTreeClassifier
dt=DecisionTreeClassifier()

dt.fit(x_train,y_train)
```

Out[64]:
```
DecisionTreeClassifier()
```

In [65]:
```
print("Decision Tree Classifier Accuracy: ", dt.score(x_test,y_test))
```

```
Decision Tree Classifier Accuracy:  0.874
```

e) Random Forest Classifier

In [72]:
```
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier(n_estimators=100, random_state=1)
```

In [73]:
```
rf.fit(x_train,y_train)
```

Out[73]:
```
RandomForestClassifier(random_state=1)
```

In [74]:
```
print("Random Forest Accuracy: ", rf.score(x_test,y_test))
```

```
Random Forest Accuracy:  0.92
```

4)Predict the price range for test data

In [75]:
```
test1['rbf_pred'] = y_predrbf
test1.head(20)
```

Out[75]:

	price_org	logistic_pred	kn_pred	svm_pred	rbf_pred
256	0	0	0	0	0
352	0	0	0	0	0
298	0	0	0	0	0
581	3	3	3	3	3
1288	1	1	1	1	1
1765	2	2	2	2	2
420	1	1	1	1	1
1587	1	1	1	1	1
65	3	3	3	3	2
1611	2	2	2	2	2
56	0	0	0	0	0
1998	0	0	0	0	0
1117	2	2	2	2	2
582	0	0	0	0	0
1232	2	2	2	2	2
316	1	1	1	1	1
744	2	2	2	2	2
128	3	3	3	3	3
1442	1	1	1	1	1
1512	3	3	3	3	3

6)Report the model with the best accuracy

a)Logistic Regression Score(in percentage): 96.2% b)KNN Score(in percentage): 91.6% c)(i)SVM Classifier with linear(in percentage)=96.6% (ii)SVM Classifier with rbf kernel(in percentage)=92.8% d)Decision Tree score(in percentage):87.4% e)Random Forest score(in percenatge):92% Therefore SVM Classifier with linear model scoring high accuracy so SVM Classifier with linear model is a best accuracy model

In [ ]:
```
```

In [ ]:
```
```