In [1]:

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

In [2]:

```
df = pd.read_csv('C:\\Users\\ejaza\\OneDrive\\Desktop\\mobile_price_range_data.csv
df
```

Out[2]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt
0	842	0	2.2	0	1	0	7	0.6	188
1	1021	1	0.5	1	0	1	53	0.7	136
2	563	1	0.5	1	2	1	41	0.9	145
3	615	1	2.5	0	0	0	10	0.8	131
4	1821	1	1.2	0	13	1	44	0.6	141
...	...	...	...	...	...	...	...	...	...
1995	794	1	0.5	1	0	1	2	0.8	106
1996	1965	1	2.6	1	0	0	39	0.2	187
1997	1911	0	0.9	1	1	1	36	0.7	108
1998	1512	0	0.9	0	4	1	46	0.1	145
1999	510	1	2.0	1	5	1	45	0.9	168

2000 rows × 21 columns

In [3]:

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

Out[3]:

```
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 [4]:

```
df.duplicated().sum()
```

Out[4]:

```
0
```

In [5]:

```
df.dtypes
```

Out[5]:

```
battery_power      int64
blue               int64
clock_speed      float64
dual_sim           int64
fc                 int64
four_g             int64
int_memory         int64
m_dep            float64
mobile_wt          int64
n_cores            int64
pc                 int64
px_height          int64
px_width           int64
ram                int64
sc_h               int64
sc_w               int64
talk_time          int64
three_g            int64
touch_screen       int64
wifi               int64
price_range        int64
dtype: object
```

In [6]:

```
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 [7]:

```
df.describe()
```

Out[7]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memor
count	2000.000000	2000.0000	2000.000000	2000.000000	2000.000000	2000.000000	2000.00000
mean	1238.518500	0.4950	1.522250	0.509500	4.309500	0.521500	32.04650
std	439.418206	0.5001	0.816004	0.500035	4.341444	0.499662	18.14571
min	501.000000	0.0000	0.500000	0.000000	0.000000	0.000000	2.00000
25%	851.750000	0.0000	0.700000	0.000000	1.000000	0.000000	16.00000
50%	1226.000000	0.0000	1.500000	1.000000	3.000000	1.000000	32.00000
75%	1615.250000	1.0000	2.200000	1.000000	7.000000	1.000000	48.00000
max	1998.000000	1.0000	3.000000	1.000000	19.000000	1.000000	64.00000

8 rows × 21 columns

In [8]:

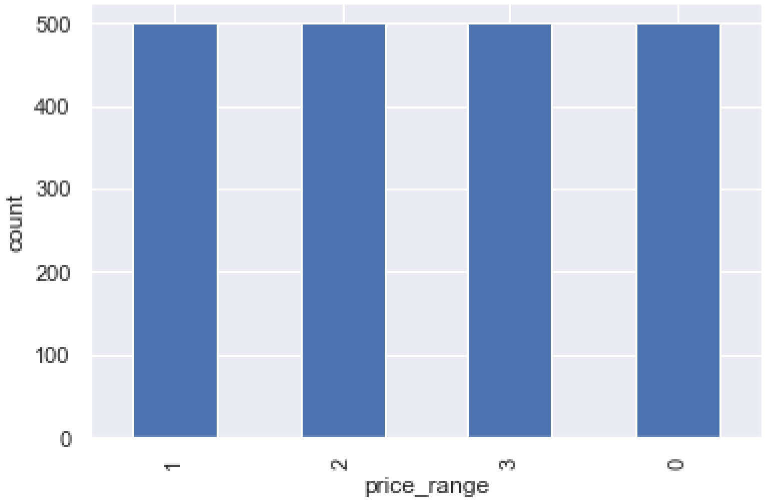
```
df.shape
```

Out[8]:

```
(2000, 21)
```

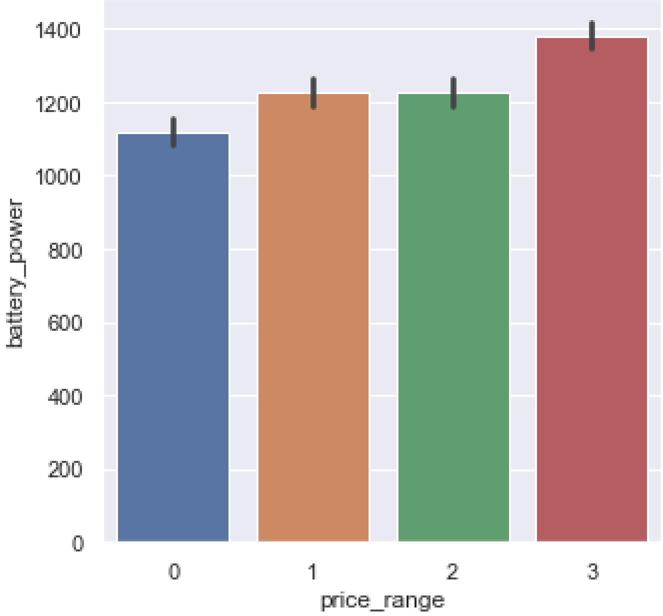
In [9]:

```
sns.set()
price_plot = df['price_range'].value_counts().plot(kind='bar')
plt.xlabel('price_range')
plt.ylabel('count')
plt.show()
# df['price_range'].value_counts()
```

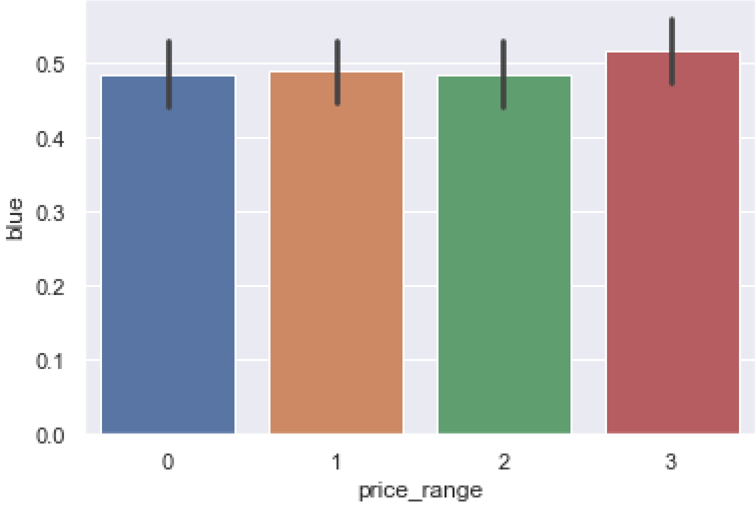


In [10]:

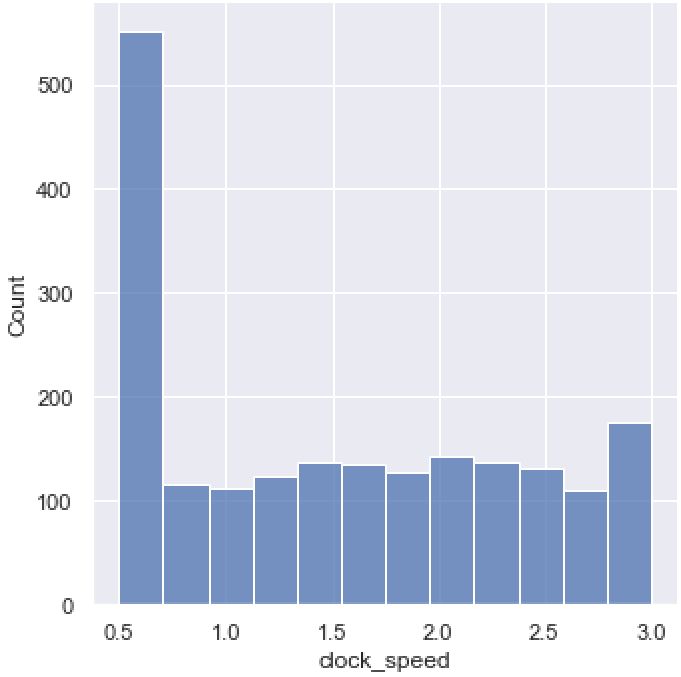
```
plt.figure(figsize = (5, 5))
sns.barplot(x = 'price_range', y = 'battery_power', data = df)
plt.show()
```



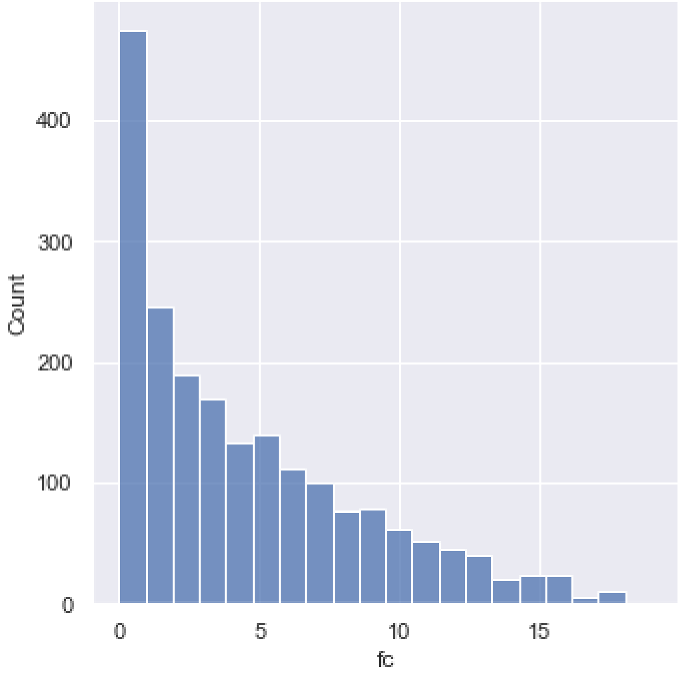
In [11]:
```
# sns.displot(df, x = df['blue'])
sns.barplot(x = 'price_range', y = 'blue', data = df)
plt.show()
```



In [12]:
```
sns.displot(df, x = df['clock_speed'])
plt.show()
```

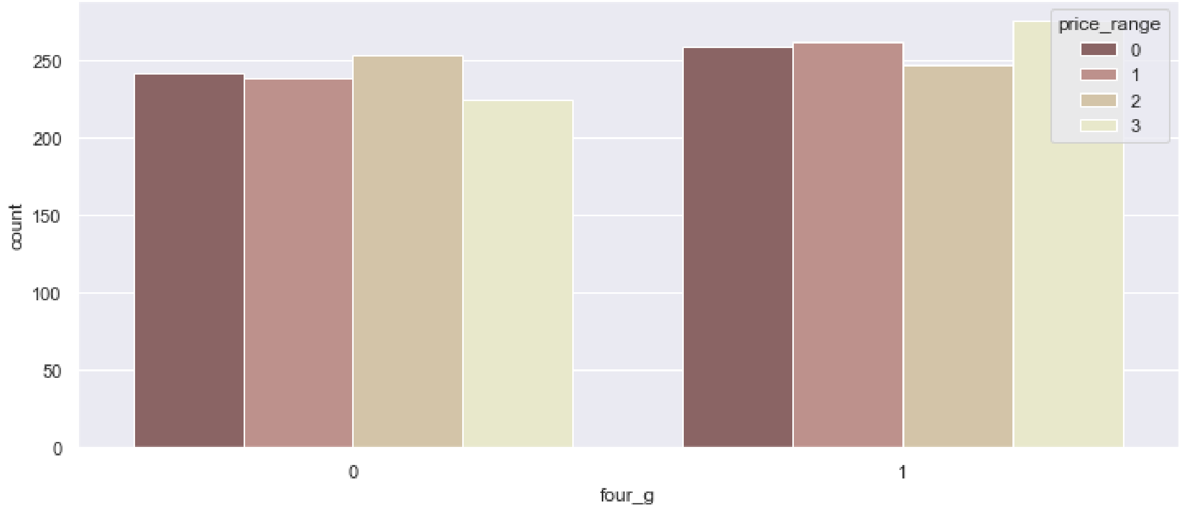


In [13]:
```
sns.displot(df, x = df['fc'])
plt.show()
```



In [14]:
```
plt.figure(figsize = (12, 5))
sns.countplot(df['four_g'], hue = df['price_range'], palette = 'pink')
plt.show()
```

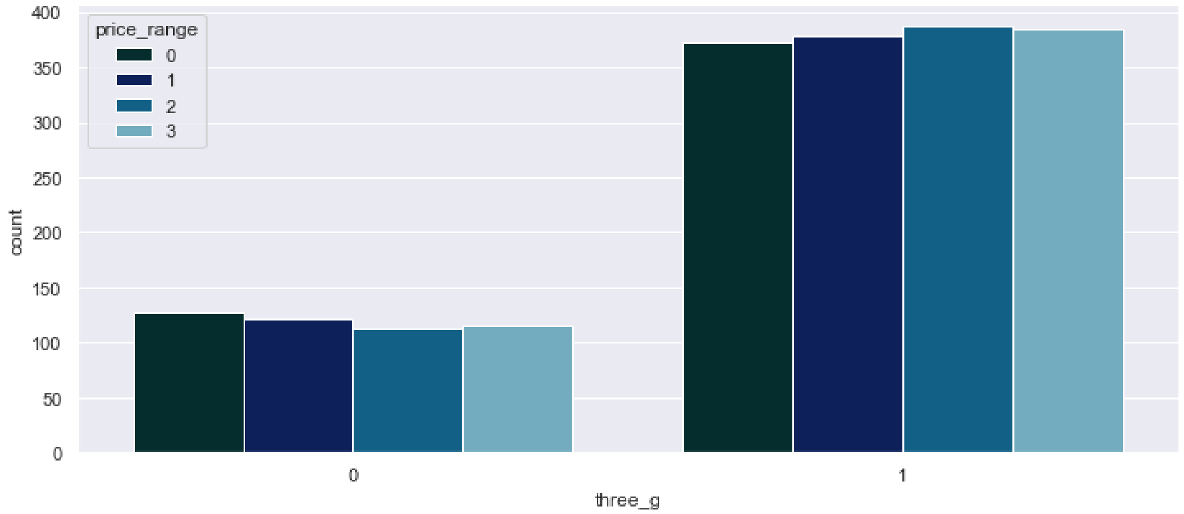
```
C:\Users\ejaza\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarnin
g: Pass the following variable as a keyword arg: x. From version 0.12, the only va
lid positional argument will be `data`, and passing other arguments without an exp
licit keyword will result in an error or misinterpretation.
  warnings.warn(
```



In [15]:

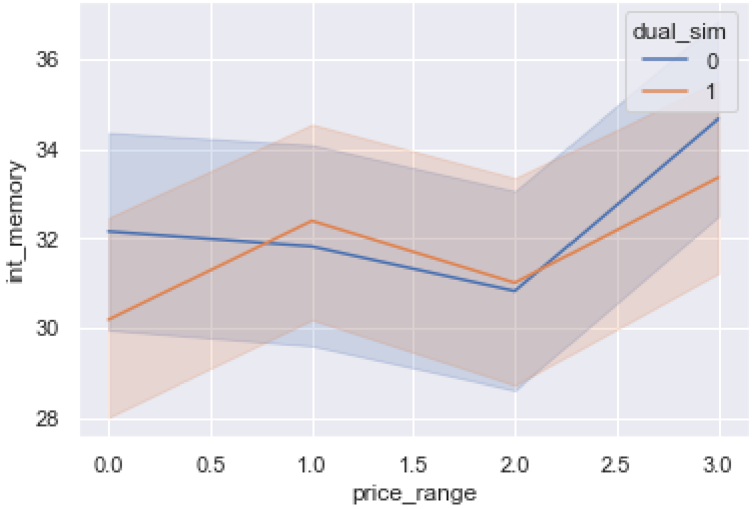
```
plt.figure(figsize = (12, 5))
sns.countplot(df['three_g'], hue = df['price_range'], palette = 'ocean')
plt.show()
```

```
C:\Users\ejaza\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarnin
g: Pass the following variable as a keyword arg: x. From version 0.12, the only va
lid positional argument will be `data`, and passing other arguments without an exp
licit keyword will result in an error or misinterpretation.
  warnings.warn(
```

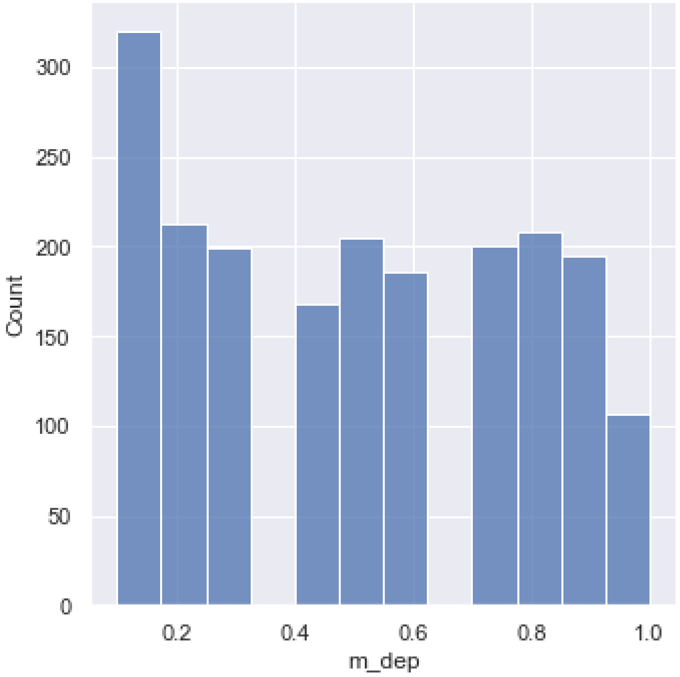


In [16]:

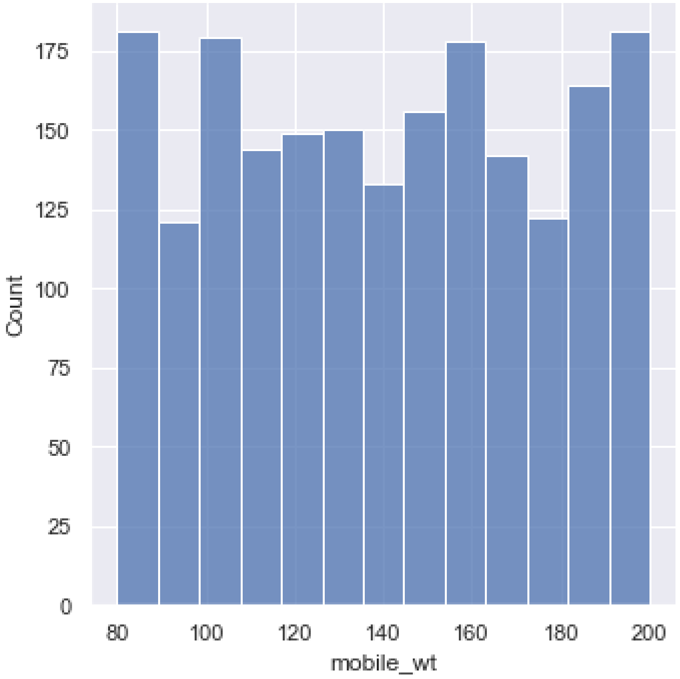
```
sns.lineplot(x = 'price_range', y = 'int_memory', data = df, hue = 'dual_sim')
plt.show()
```



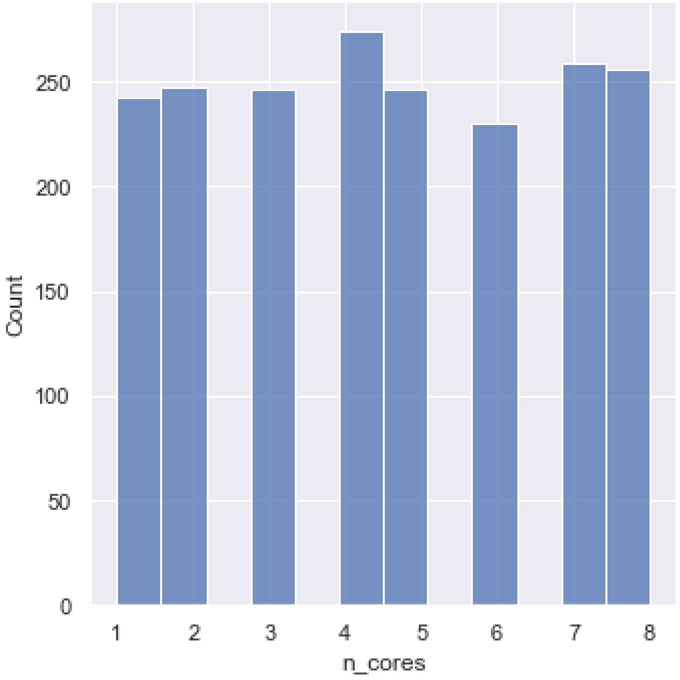
In [17]:
```
sns.displot(df, x = df['m_dep'])
plt.show()
```



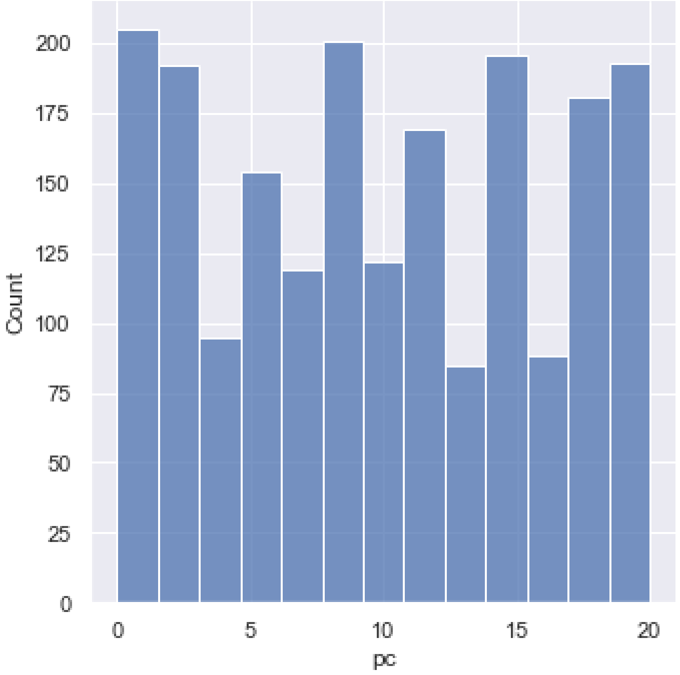
In [18]:
```
sns.displot(df, x = df['mobile_wt'])
plt.show()
```



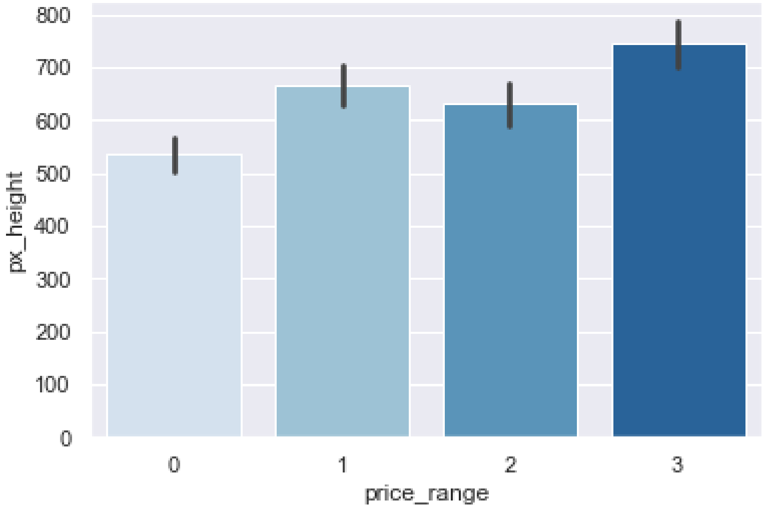
In [19]:
```
sns.displot(df, x = df['n_cores'])
plt.show()
```



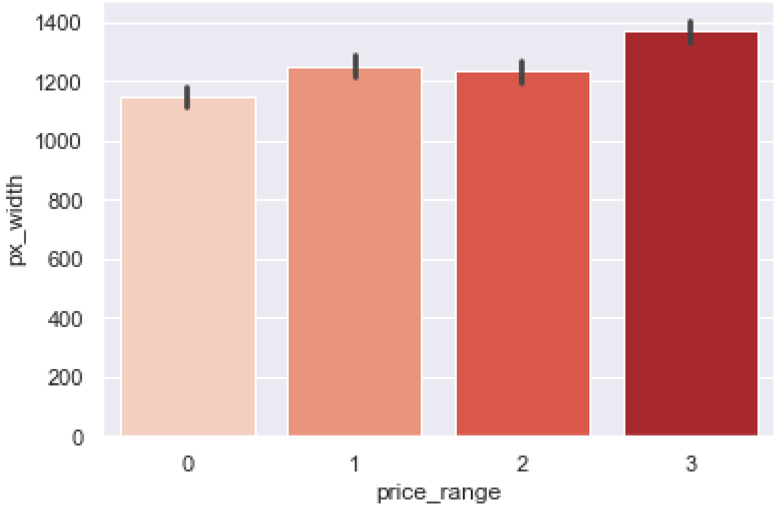
In [20]:
```
sns.displot(df, x = df['pc'])
plt.show()
```



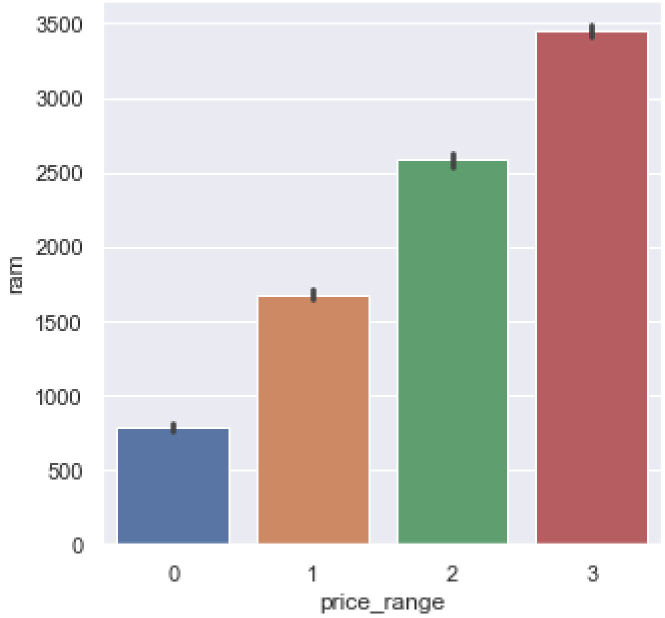
In [21]:
```
sns.barplot(x ='price_range', y = 'px_height', data = df, palette = 'Blues')
plt.show()
```



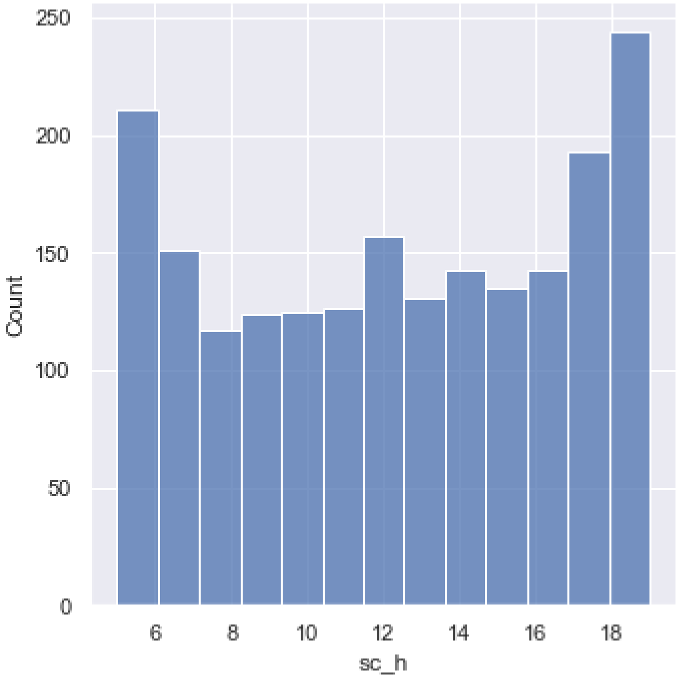
In [22]:
```
sns.barplot(x = 'price_range', y = 'px_width', data = df, palette = 'Reds')
plt.show()
```



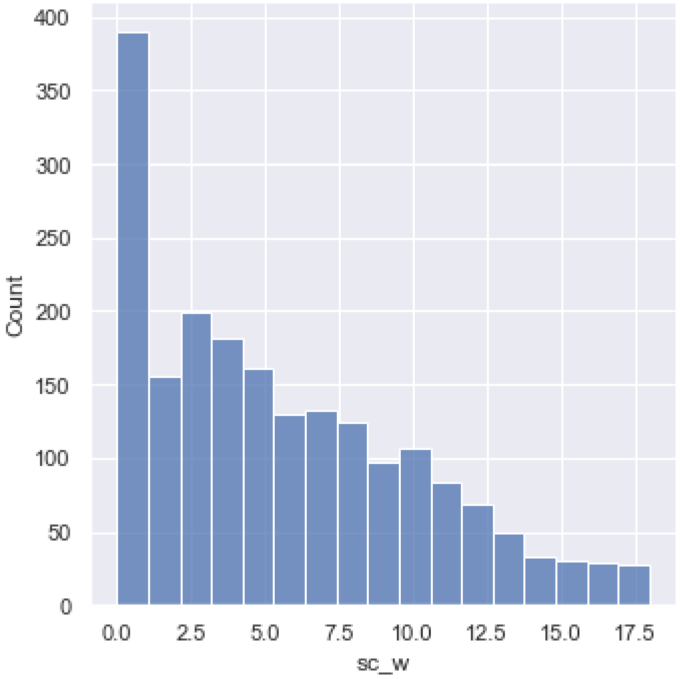
In [23]:
```
plt.figure(figsize = (5, 5))
sns.barplot(x = 'price_range', y = 'ram', data = df)
plt.show()
```



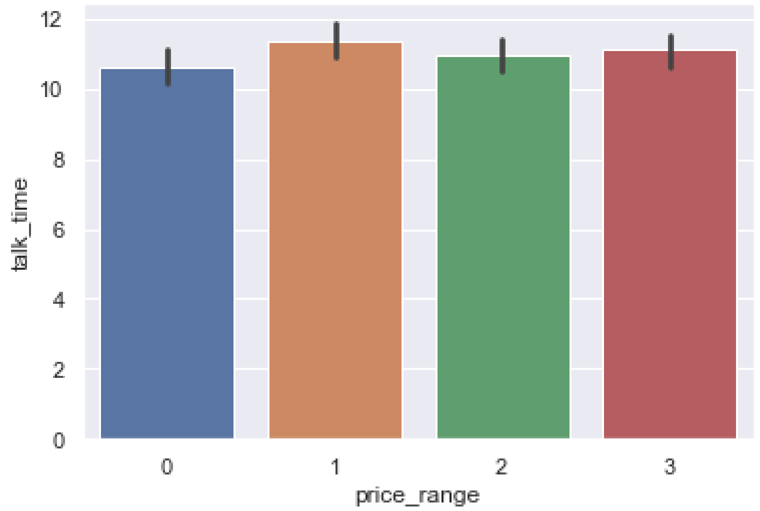
In [24]:
```
sns.displot(df, x = df['sc_h'])
plt.show()
```



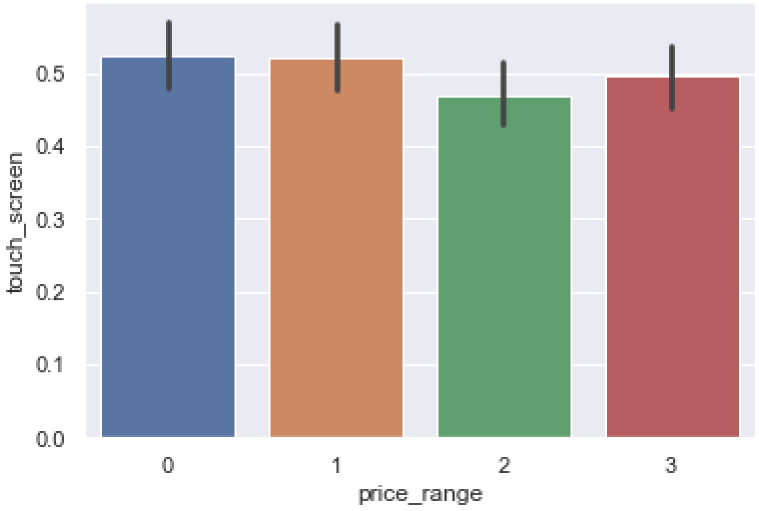
In [25]:
```
sns.displot(df, x = df['sc_w'])
plt.show()
```



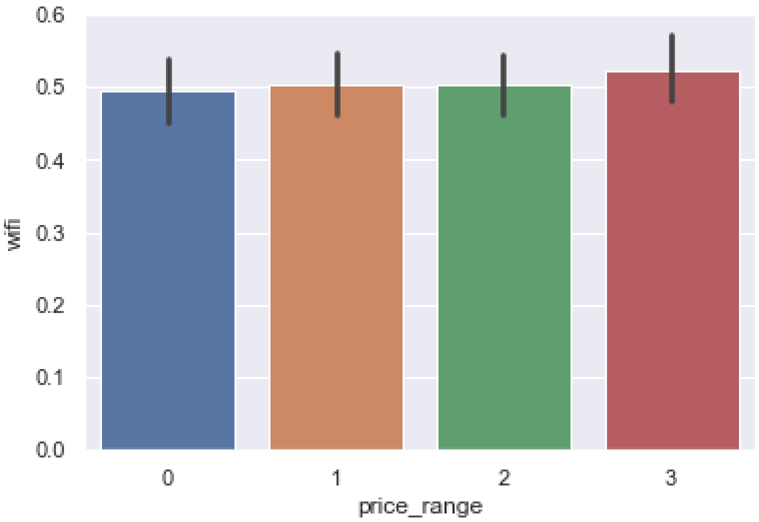
In [26]:
```
sns.barplot(x = 'price_range', y = 'talk_time', data = df)
plt.show()
```



In [27]:
```
sns.barplot(x = 'price_range', y = 'touch_screen', data = df)
plt.show()
```



In [28]:
```
sns.barplot(x = 'price_range', y = 'wifi', data = df)
plt.show()
```



In [29]:
```
x = df.iloc[:, :-1]
y = df.iloc[:, -1]
print(x.shape)
print(y.shape)
print(type(x))    # Number of Rows
print(type(y))    # Number of columns
```

```
(2000, 20)
(2000,)
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.series.Series'>
```

In [30]:

```
from sklearn.model_selection import train_test_split
```

In [31]:

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.25)
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(1500, 20)
(500, 20)
(1500,)
(500,)
```

In [32]:

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.fit_transform(x_test)
```

In [33]:

```
x_train
```

Out[33]:

```
array([[ 1.57664724,  1.02840321,  0.32913298, ...,  0.56503205,
         0.98675438,  0.96720415],
       [ 0.74504939, -0.97238125,  1.06508251, ...,  0.56503205,
         0.98675438, -1.03390789],
       [-0.90447619, -0.97238125,  0.57444949, ...,  0.56503205,
        -1.01342342, -1.03390789],
       ...,
       [ 1.61537919, -0.97238125, -0.40681654, ..., -1.76981112,
        -1.01342342,  0.96720415],
       [-0.9363731 ,  1.02840321, -1.26542432, ..., -1.76981112,
         0.98675438, -1.03390789],
       [-0.15489896, -0.97238125, -0.28415829, ..., -1.76981112,
        -1.01342342,  0.96720415]])
```

In [34]:

```
x_test
```

Out[34]:

```
array([[ 0.37894493,  0.95692675,  0.25222607, ..., -1.84009916,
         1.01613007, -0.95692675],
       [ 1.52562527, -1.04501206,  0.49710576, ...,  0.54344897,
         1.01613007,  1.04501206],
       [-0.0502087 , -1.04501206, -1.21705203, ...,  0.54344897,
         1.01613007,  1.04501206],
       ...,
       [-0.21823711,  0.95692675,  1.47662449, ...,  0.54344897,
         1.01613007, -0.95692675],
       [ 0.22908175, -1.04501206,  1.72150417, ...,  0.54344897,
        -0.98412598,  1.04501206],
       [-0.79271261,  0.95692675,  0.6195456 , ...,  0.54344897,
         1.01613007, -0.95692675]])
```

a)Logistic Regression

In [35]:

```
from sklearn.linear_model import LogisticRegression
```

In [36]:

```
m1 = LogisticRegression()
m1.fit(x_train, y_train)
```

Out[36]:
```
LogisticRegression()
```

In [37]:
```
# Accuracy
print('Training score', m1.score(x_train, y_train))
print('Testing score', m1.score(x_test, y_test))
```

```
Training score 0.9766666666666667
Testing score 0.938
```

In [38]:
```
ypred_m1 = m1.predict(x_test)
ypred_m1
```

Out[38]:
```
array([2, 2, 0, 2, 1, 2, 2, 1, 0, 2, 2, 1, 3, 3, 1, 1, 0, 3, 2, 3, 0, 2,
       3, 3, 2, 1, 0, 2, 0, 1, 0, 1, 3, 1, 2, 3, 2, 2, 0, 0, 2, 0, 0, 2,
       1, 1, 1, 1, 2, 1, 3, 1, 3, 3, 0, 1, 0, 1, 2, 3, 2, 2, 0, 3, 2, 0,
       3, 1, 3, 2, 0, 1, 1, 2, 0, 3, 2, 3, 3, 3, 3, 3, 0, 1, 2, 1, 0, 3,
       1, 2, 3, 2, 0, 1, 0, 2, 0, 1, 0, 3, 0, 3, 3, 0, 3, 2, 1, 2, 0, 3,
       2, 2, 1, 2, 3, 0, 1, 2, 0, 1, 0, 0, 3, 0, 2, 2, 0, 1, 3, 0, 1, 2,
       3, 3, 2, 3, 0, 0, 1, 1, 3, 2, 3, 0, 0, 0, 1, 2, 1, 2, 0, 0, 1, 0,
       1, 2, 2, 3, 2, 1, 3, 3, 0, 2, 1, 1, 1, 0, 0, 2, 1, 1, 3, 1, 0, 3,
       2, 1, 2, 2, 0, 0, 0, 2, 3, 3, 0, 1, 2, 0, 3, 1, 1, 1, 0, 0, 2, 2,
       2, 1, 2, 2, 1, 3, 2, 1, 1, 0, 1, 2, 0, 1, 1, 0, 0, 0, 0, 1, 0, 2,
       0, 3, 1, 3, 2, 3, 1, 0, 1, 0, 3, 2, 2, 0, 3, 0, 0, 3, 3, 1, 2, 0,
       0, 2, 3, 1, 1, 3, 1, 3, 3, 3, 0, 3, 0, 3, 3, 1, 0, 3, 0, 2, 0, 1,
       3, 2, 3, 0, 1, 0, 2, 3, 3, 3, 2, 2, 0, 0, 3, 0, 3, 2, 1, 0, 3, 0,
       1, 2, 2, 3, 0, 3, 1, 2, 3, 0, 0, 0, 3, 1, 2, 0, 3, 0, 2, 3, 1, 0,
       2, 2, 2, 0, 3, 2, 1, 0, 1, 3, 1, 2, 0, 2, 1, 2, 3, 0, 2, 2, 2, 3,
       3, 0, 0, 3, 0, 2, 0, 2, 1, 1, 3, 1, 3, 0, 1, 2, 2, 0, 1, 1, 1, 1,
       1, 2, 0, 3, 2, 3, 2, 3, 2, 0, 1, 1, 3, 2, 3, 3, 3, 1, 3, 3, 1, 3,
       2, 1, 3, 1, 1, 3, 1, 1, 2, 0, 0, 0, 3, 2, 0, 1, 3, 1, 3, 2, 1, 1,
       2, 3, 2, 1, 2, 1, 2, 1, 0, 2, 2, 1, 1, 1, 1, 0, 2, 0, 3, 0, 3, 2,
       2, 3, 2, 1, 1, 3, 2, 2, 1, 3, 1, 2, 0, 1, 2, 2, 0, 1, 1, 1, 3, 3,
       0, 2, 1, 3, 2, 0, 0, 3, 1, 2, 1, 1, 3, 0, 0, 3, 2, 1, 2, 0, 0, 2,
       0, 0, 3, 3, 2, 0, 3, 1, 2, 1, 0, 3, 1, 0, 1, 1, 0, 2, 2, 3, 1, 3,
       3, 0, 3, 1, 3, 2, 3, 0, 1, 0, 1, 0, 3, 0, 2, 1], dtype=int64)
```

In [39]:
```
from sklearn.metrics import confusion_matrix, classification_report, accuracy_scor
```

In [40]:
```
cm  = confusion_matrix(y_test, ypred_m1)
print(cm)
print(classification_report(y_test, ypred_m1))
# Accuracy score
lr_acc = accuracy_score(y_test, ypred_m1)
print(lr_acc)
```

```
[[112   0   0   0]
 [ 14 124   1   0]
 [  0   4 118   6]
 [  0   0   6 115]]
              precision    recall  f1-score   support

           0       0.89      1.00      0.94       112
           1       0.97      0.89      0.93       139
           2       0.94      0.92      0.93       128
           3       0.95      0.95      0.95       121

    accuracy                           0.94       500
   macro avg       0.94      0.94      0.94       500
weighted avg       0.94      0.94      0.94       500

0.938
```

b)KNN Classification

```
In [41]: from sklearn.neighbors import KNeighborsClassifier
```

```
In [42]: m2 = KNeighborsClassifier(n_neighbors = 11)
         m2.fit(x_train, y_train)
```

```
Out[42]: KNeighborsClassifier(n_neighbors=11)
```

```
In [43]: print('Training score', m2.score(x_train, y_train))
         print('Testing score', m2.score(x_test, y_test))
```

```
Training score 0.6773333333333333
Testing score 0.526
```

```
In [44]: ypred_m2 = m2.predict(x_test)
         ypred_m2
```

```
Out[44]: array([2, 1, 1, 2, 2, 2, 2, 2, 0, 2, 2, 2, 3, 3, 1, 0, 0, 3, 1, 3, 1, 3,
                3, 2, 2, 0, 0, 3, 0, 1, 0, 1, 2, 0, 2, 3, 1, 3, 1, 0, 1, 0, 1, 2,
                2, 0, 0, 0, 0, 2, 3, 0, 3, 3, 0, 1, 0, 2, 2, 3, 3, 1, 0, 1, 1, 0,
                1, 1, 2, 2, 1, 1, 1, 1, 0, 2, 1, 3, 3, 2, 3, 2, 0, 2, 1, 2, 0, 3,
                1, 3, 3, 1, 0, 0, 0, 0, 0, 1, 0, 3, 0, 3, 3, 0, 3, 2, 2, 2, 0, 3,
                2, 1, 1, 2, 2, 0, 1, 1, 0, 0, 0, 1, 3, 1, 2, 3, 1, 1, 3, 1, 0, 2,
                3, 3, 1, 2, 1, 0, 1, 0, 3, 2, 2, 0, 0, 0, 0, 1, 1, 1, 1, 0, 2, 0,
                1, 2, 2, 3, 1, 2, 1, 2, 1, 2, 0, 0, 1, 0, 0, 0, 1, 1, 2, 1, 0, 2,
                1, 2, 3, 3, 0, 0, 1, 2, 3, 2, 1, 0, 2, 0, 3, 2, 0, 2, 0, 0, 1, 0,
                2, 1, 3, 2, 1, 1, 3, 1, 1, 0, 1, 2, 1, 1, 2, 0, 0, 0, 0, 1, 0, 2,
                0, 1, 1, 3, 0, 2, 1, 0, 1, 0, 3, 2, 2, 0, 2, 0, 0, 2, 3, 1, 1, 0,
                1, 3, 2, 1, 1, 3, 2, 1, 3, 2, 0, 3, 1, 3, 3, 1, 0, 2, 0, 1, 2, 0,
                2, 1, 2, 0, 3, 1, 0, 3, 3, 3, 3, 0, 1, 0, 1, 0, 1, 1, 1, 2, 2, 1,
                0, 2, 1, 2, 0, 3, 0, 2, 2, 0, 0, 0, 3, 0, 0, 0, 1, 0, 0, 3, 0, 0,
                2, 0, 2, 2, 3, 0, 0, 0, 1, 1, 0, 0, 0, 2, 2, 3, 3, 0, 2, 2, 1, 2,
                3, 2, 0, 3, 1, 2, 0, 1, 1, 0, 3, 1, 3, 0, 1, 1, 3, 1, 2, 2, 1, 1,
                1, 0, 0, 3, 1, 3, 2, 2, 2, 0, 1, 0, 2, 2, 3, 3, 3, 3, 2, 3, 0, 3,
                2, 1, 1, 1, 1, 3, 2, 1, 2, 0, 0, 0, 3, 2, 2, 0, 2, 0, 3, 1, 2, 0,
                0, 2, 2, 1, 2, 2, 2, 1, 1, 3, 2, 1, 1, 1, 1, 0, 2, 0, 3, 0, 2, 2,
                2, 1, 1, 0, 2, 2, 3, 2, 0, 3, 0, 3, 1, 1, 2, 2, 0, 2, 0, 1, 1, 3,
                0, 2, 0, 2, 0, 0, 0, 3, 2, 0, 1, 1, 2, 0, 0, 2, 0, 0, 2, 0, 1, 1,
                1, 0, 1, 3, 2, 1, 2, 0, 1, 0, 2, 3, 1, 0, 2, 1, 2, 1, 3, 3, 1, 3,
                1, 0, 3, 0, 2, 1, 2, 1, 0, 0, 2, 0, 2, 0, 2, 1], dtype=int64)
```

```
In [45]: from sklearn.metrics import confusion_matrix, classification_report,accuracy_score
```

```
In [46]: cm = confusion_matrix(y_test, ypred_m2)
         print(cm)
         print(classification_report(y_test, ypred_m2))
         # Accuracy score
         knn_acc = accuracy_score(ypred_m2, y_test)
         print(knn_acc)
```

```
[[82 25  5  0]
 [49 60 28  2]
 [16 38 55 19]
 [ 0 13 42 66]]
              precision    recall  f1-score   support

           0       0.56      0.73      0.63       112
           1       0.44      0.43      0.44       139
           2       0.42      0.43      0.43       128
           3       0.76      0.55      0.63       121

    accuracy                           0.53       500
   macro avg       0.55      0.53      0.53       500
weighted avg       0.54      0.53      0.53       500

0.526
```

c)Decision Tree Classification

In [47]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [48]:
```
m3 = DecisionTreeClassifier()
m3.fit(x_train, y_train)
```

Out[48]:
```
DecisionTreeClassifier()
```

In [49]:
```
print('Training score', m3.score(x_train, y_train))
print('Testing score', m3.score(x_test, y_test))
```

```
Training score 1.0
Testing score 0.844
```

In [50]:
```
ypred_m3 = m3.predict(x_test)
ypred_m3
```

Out[50]:
```
array([2, 2, 0, 1, 2, 2, 2, 1, 0, 2, 2, 1, 2, 3, 1, 1, 0, 3, 2, 3, 1, 2,
       3, 3, 1, 1, 0, 2, 0, 1, 0, 1, 3, 1, 2, 3, 1, 2, 0, 0, 2, 0, 0, 3,
       1, 1, 1, 1, 3, 2, 3, 1, 2, 3, 0, 1, 0, 1, 2, 3, 2, 2, 1, 3, 1, 0,
       3, 2, 3, 2, 0, 1, 1, 2, 0, 3, 1, 3, 3, 3, 3, 3, 0, 1, 2, 0, 0, 3,
       1, 3, 3, 2, 0, 1, 0, 1, 1, 1, 0, 3, 0, 3, 2, 0, 3, 2, 1, 3, 0, 3,
       1, 2, 1, 1, 3, 0, 1, 1, 0, 0, 0, 0, 3, 0, 2, 2, 0, 1, 3, 0, 1, 2,
       3, 3, 2, 3, 0, 1, 1, 1, 3, 2, 2, 0, 0, 0, 2, 2, 1, 2, 0, 0, 1, 0,
       1, 1, 2, 3, 2, 1, 2, 3, 0, 2, 1, 1, 1, 0, 0, 2, 1, 1, 3, 1, 0, 3,
       3, 1, 2, 2, 0, 0, 0, 2, 3, 3, 0, 1, 2, 0, 3, 1, 1, 1, 0, 0, 3, 2,
       2, 1, 2, 3, 1, 2, 1, 1, 1, 0, 1, 2, 0, 1, 1, 0, 0, 0, 1, 1, 0, 2,
       0, 2, 1, 3, 2, 3, 1, 0, 1, 0, 3, 2, 2, 0, 3, 0, 0, 3, 3, 1, 3, 0,
       0, 2, 2, 2, 1, 3, 1, 2, 3, 3, 0, 3, 0, 3, 3, 1, 0, 3, 0, 2, 0, 1,
       3, 2, 3, 1, 1, 0, 1, 2, 3, 3, 2, 1, 0, 0, 2, 0, 3, 1, 1, 0, 3, 0,
       1, 2, 2, 3, 0, 3, 1, 2, 3, 1, 0, 0, 3, 1, 2, 0, 3, 0, 1, 3, 1, 0,
       2, 2, 3, 0, 3, 2, 1, 0, 1, 3, 1, 2, 0, 2, 1, 2, 2, 0, 2, 2, 2, 3,
       2, 0, 0, 3, 0, 3, 0, 2, 0, 1, 3, 1, 3, 0, 1, 2, 2, 0, 1, 1, 1, 1,
       1, 1, 1, 3, 2, 3, 3, 3, 2, 1, 1, 0, 3, 2, 3, 3, 3, 1, 3, 3, 1, 3,
       1, 1, 3, 1, 2, 3, 1, 1, 2, 0, 1, 0, 3, 2, 0, 1, 3, 1, 3, 1, 1, 1,
       1, 3, 2, 1, 2, 1, 2, 2, 0, 2, 2, 2, 1, 2, 1, 0, 2, 0, 3, 0, 2, 2,
       1, 2, 2, 1, 1, 3, 2, 1, 1, 3, 1, 3, 0, 1, 2, 2, 0, 1, 1, 1, 3, 3,
       0, 2, 1, 3, 2, 0, 0, 3, 1, 2, 1, 1, 3, 0, 0, 3, 2, 1, 1, 0, 0, 2,
       0, 0, 3, 3, 2, 0, 3, 1, 2, 1, 0, 3, 1, 0, 1, 2, 0, 2, 2, 3, 1, 3,
       3, 0, 3, 1, 3, 2, 3, 1, 1, 0, 1, 0, 2, 0, 2, 1], dtype=int64)
```

In [51]:
```
from sklearn.metrics import confusion_matrix, classification_report,accuracy_score
```

In [52]:

```
cm = confusion_matrix(y_test, ypred_m3)
print(cm)
print(classification_report(y_test, ypred_m3))
# Accuracy score
dtc_acc = accuracy_score(y_test, ypred_m3)
print(dtc_acc)
```

```
[[105   7   0   0]
 [ 14 116   9   0]
 [  0  24  94  10]
 [  0   0  14 107]]
              precision    recall  f1-score   support

           0       0.88      0.94      0.91       112
           1       0.79      0.83      0.81       139
           2       0.80      0.73      0.77       128
           3       0.91      0.88      0.90       121

    accuracy                           0.84       500
   macro avg       0.85      0.85      0.85       500
weighted avg       0.84      0.84      0.84       500

0.844
```

d)Random Forest Classification

In [53]:

```
from sklearn.ensemble import RandomForestClassifier
```

In [54]:

```
m4 = RandomForestClassifier()
m4.fit(x_train, y_train)
```

Out[54]:

```
RandomForestClassifier()
```

In [55]:

```
print('Training score', m4.score(x_train, y_train))
print('Testing score', m4.score(x_test, y_test))
```

```
Training score 1.0
Testing score 0.864
```

In [56]:

```
ypred_m4 = m4.predict(x_test)
ypred_m4
```

Out[56]:

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

In [57]:

```
from sklearn.metrics import confusion_matrix, classification_report, accuracy_scor
```

In [58]:

```
cm = confusion_matrix(y_test, ypred_m4)
print(cm)
print(classification_report(y_test, ypred_m4))
# Accuracy score
rfc_acc = accuracy_score(y_test, ypred_m4)
print(rfc_acc)
```

```
[[109   3   0   0]
 [ 20 108  11   0]
 [  0  16 103   9]
 [  0   0   9 112]]
              precision    recall  f1-score   support

           0       0.84      0.97      0.90       112
           1       0.85      0.78      0.81       139
           2       0.84      0.80      0.82       128
           3       0.93      0.93      0.93       121

    accuracy                           0.86       500
   macro avg       0.86      0.87      0.87       500
weighted avg       0.86      0.86      0.86       500

0.864
```

e)SVM classifier with linear kernel

In [59]:

```
from sklearn.svm import SVC
```

Linear Kernel

In [60]:

```
m5 = SVC(kernel='linear', C=10)
m5.fit(x_train, y_train)
```

```
Out[60]: SVC(C=10, kernel='linear')
```

```
In [61]: print('Training score', m5.score(x_train, y_train))
         print('Testing score', m5.score(x_test, y_test))
```

```
Training score 0.984
Testing score 0.954
```

```
In [62]: ypred_m5 = m5.predict(x_test)
         ypred_m5
```

```
Out[62]: array([2, 2, 0, 2, 1, 2, 2, 1, 0, 2, 2, 1, 3, 3, 1, 1, 0, 3, 2, 3, 0, 2,
                3, 3, 2, 1, 0, 2, 0, 1, 0, 1, 3, 1, 2, 3, 2, 2, 0, 0, 2, 0, 0, 2,
                1, 1, 1, 1, 2, 1, 3, 1, 3, 3, 0, 1, 0, 1, 2, 3, 2, 2, 0, 2, 2, 0,
                3, 1, 3, 2, 0, 1, 1, 2, 0, 3, 2, 3, 3, 3, 3, 3, 0, 1, 2, 1, 0, 3,
                1, 3, 3, 2, 0, 1, 0, 1, 1, 1, 0, 3, 1, 3, 2, 0, 3, 2, 1, 3, 0, 3,
                2, 2, 1, 2, 3, 0, 1, 1, 0, 1, 0, 0, 3, 0, 2, 2, 0, 1, 3, 0, 1, 2,
                3, 3, 2, 3, 0, 0, 1, 1, 3, 2, 3, 0, 0, 0, 1, 2, 1, 2, 0, 0, 1, 0,
                1, 2, 2, 3, 2, 1, 3, 3, 0, 2, 1, 1, 1, 0, 0, 2, 1, 1, 2, 1, 0, 3,
                2, 1, 2, 2, 0, 0, 0, 2, 3, 3, 0, 1, 2, 0, 3, 2, 1, 1, 0, 0, 2, 2,
                2, 1, 2, 3, 1, 3, 2, 1, 1, 0, 1, 2, 0, 1, 1, 0, 0, 0, 0, 1, 0, 2,
                0, 2, 1, 3, 2, 3, 1, 0, 1, 0, 3, 2, 2, 0, 3, 0, 0, 3, 3, 1, 2, 0,
                0, 2, 3, 1, 1, 3, 1, 3, 3, 3, 0, 3, 0, 3, 3, 1, 0, 3, 0, 2, 0, 1,
                3, 2, 3, 0, 1, 0, 2, 3, 3, 3, 2, 1, 0, 0, 3, 0, 3, 1, 1, 0, 3, 0,
                1, 2, 2, 3, 0, 3, 1, 2, 3, 0, 0, 0, 3, 1, 2, 0, 3, 0, 2, 3, 1, 0,
                2, 2, 2, 0, 3, 2, 1, 0, 1, 3, 1, 2, 0, 2, 1, 2, 3, 0, 2, 2, 2, 3,
                3, 0, 0, 3, 0, 2, 0, 2, 1, 0, 3, 1, 3, 0, 1, 2, 2, 0, 1, 1, 1, 1,
                1, 2, 0, 3, 2, 3, 2, 3, 2, 0, 1, 1, 3, 2, 3, 3, 3, 1, 3, 3, 1, 3,
                2, 1, 3, 1, 1, 3, 1, 1, 2, 0, 1, 0, 3, 2, 0, 1, 3, 1, 3, 2, 1, 1,
                2, 3, 2, 1, 2, 1, 2, 1, 0, 2, 2, 1, 1, 1, 1, 0, 2, 0, 3, 0, 3, 2,
                2, 3, 2, 1, 1, 3, 2, 2, 1, 3, 1, 2, 0, 1, 2, 2, 0, 1, 1, 1, 3, 3,
                0, 2, 1, 3, 2, 0, 0, 3, 1, 2, 1, 1, 3, 0, 0, 3, 2, 1, 2, 0, 0, 2,
                0, 0, 3, 3, 2, 0, 3, 1, 2, 1, 0, 3, 1, 0, 1, 1, 0, 2, 2, 3, 1, 3,
                3, 0, 3, 1, 3, 2, 3, 0, 1, 0, 1, 0, 3, 0, 2, 1], dtype=int64)
```

```
In [63]: from sklearn.metrics import confusion_matrix, classification_report, accuracy_scor
```

```
In [64]: cm = confusion_matrix(y_test, ypred_m5)
         print(cm)
         print(classification_report(y_test, ypred_m5))
         # Accuracy score
         svc_acc = accuracy_score(y_test, ypred_m5)
         print(svc_acc)
```

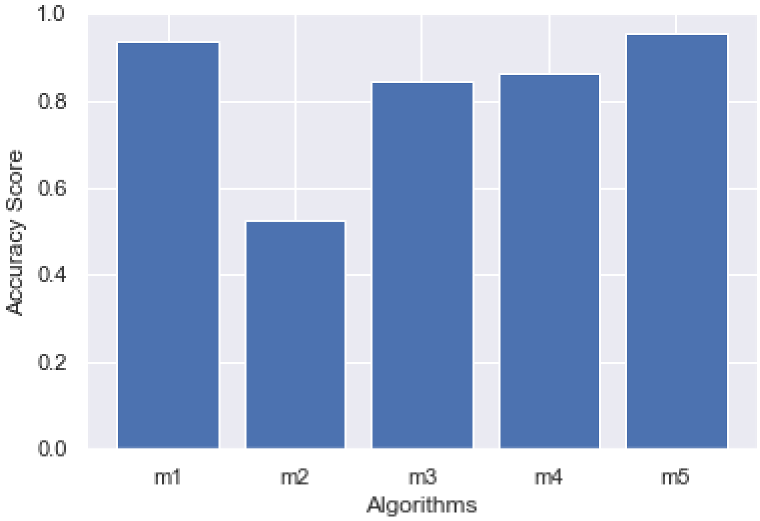
```
[[112   0   0   0]
 [ 12 127   0   0]
 [  0   6 120   2]
 [  0   0   3 118]]
              precision    recall  f1-score   support

           0       0.90      1.00      0.95       112
           1       0.95      0.91      0.93       139
           2       0.98      0.94      0.96       128
           3       0.98      0.98      0.98       121

    accuracy                           0.95       500
   macro avg       0.95      0.96      0.95       500
weighted avg       0.96      0.95      0.95       500

0.954
```

```
In [66]: # Accuracies of all models bar graph
         plt.bar(x = ['m1', 'm2', 'm3', 'm4', 'm5'], height = [lr_acc, knn_acc, dtc_acc, rf
         plt.xlabel("Algorithms")
         plt.ylabel("Accuracy Score")
         plt.show()
```



Model Accuracy Score Logistic Regression 0.938 KNN Classification 0.526 Decision Tree Classification 0.844 Random Forest Classification 0.864 SVM Classifier with Linear Kernel 0.954

Report

The results of our tests were quantified in terms of the Accuracy score of our prediction. Building up from the relatively good performance of SVM Classifier with linear kernel, it produced the best Accuracy score on test data. Hence, we can conclude that the SVM classifier with linear kernel works best on our dataset for prediction.

```
In [ ]:
```

```
In [ ]:
```