

Ranking Features of a Smartphone

Dataset information

- battery_power - Total energy a battery can store in one time measured in mAh
- blue - Has bluetooth (1) or not (0)
- clock_speed - speed at which microprocessor executes instructions
- dual_sim - Has dual sim support (1) or not (0)
- fc - Front Camera mega pixels
- four_g - Has 4G (1) or not (0)
- 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
- three_g - Has 3G (1) or not (0)
- touch_screen - Has touch screen (1) or not (0)
- wifi - Has wifi (1) or not (0)
- price_range - This is the target variable with value of 0(low cost), 1(medium cost), 2(high cost) and 3(very high cost).

Importing the necessary Libraries

```
In [1]: # importing necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

Importing the dataset and viewing it

In [2]: *#reading the dataset*

```
data=pd.read_csv(r"C:\Users\lenovo\Desktop\MobileTrain.csv")
data.head()
```

Out[2]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_c
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	

5 rows × 21 columns



Descriptive Statistics

In [3]: `data.shape`

Out[3]: (2000, 21)

In [4]: `data.info()`

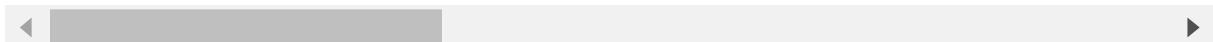
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 21 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   battery_power    2000 non-null    int64  
 1   blue              2000 non-null    int64  
 2   clock_speed      2000 non-null    float64 
 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 [5]: `#statistical description`
`data.describe()`

Out[5]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory
count	2000.000000	2000.0000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000
mean	1238.518500	0.4950	1.522250	0.509500	4.309500	0.521500	32.0465
std	439.418206	0.5001	0.816004	0.500035	4.341444	0.499662	18.1457
min	501.000000	0.0000	0.500000	0.000000	0.000000	0.000000	2.0000
25%	851.750000	0.0000	0.700000	0.000000	1.000000	0.000000	16.0000
50%	1226.000000	0.0000	1.500000	1.000000	3.000000	1.000000	32.0000
75%	1615.250000	1.0000	2.200000	1.000000	7.000000	1.000000	48.0000
max	1998.000000	1.0000	3.000000	1.000000	19.000000	1.000000	64.0000

8 rows × 21 columns



In [6]: *#datatypes of each column*
data.dtypes

Out[6]:

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 [7]: *#unique values in each column*
data.nunique()

Out[7]:

battery_power	1094
blue	2
clock_speed	26
dual_sim	2
fc	20
four_g	2
int_memory	63
m_dep	10
mobile_wt	121
n_cores	8
pc	21
px_height	1137
px_width	1109
ram	1562
sc_h	15
sc_w	19
talk_time	19
three_g	2
touch_screen	2
wifi	2
price_range	4
dtype:	int64

```
In [8]: ## missing value handling
data.isnull().sum()
```

```
Out[8]: 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
```

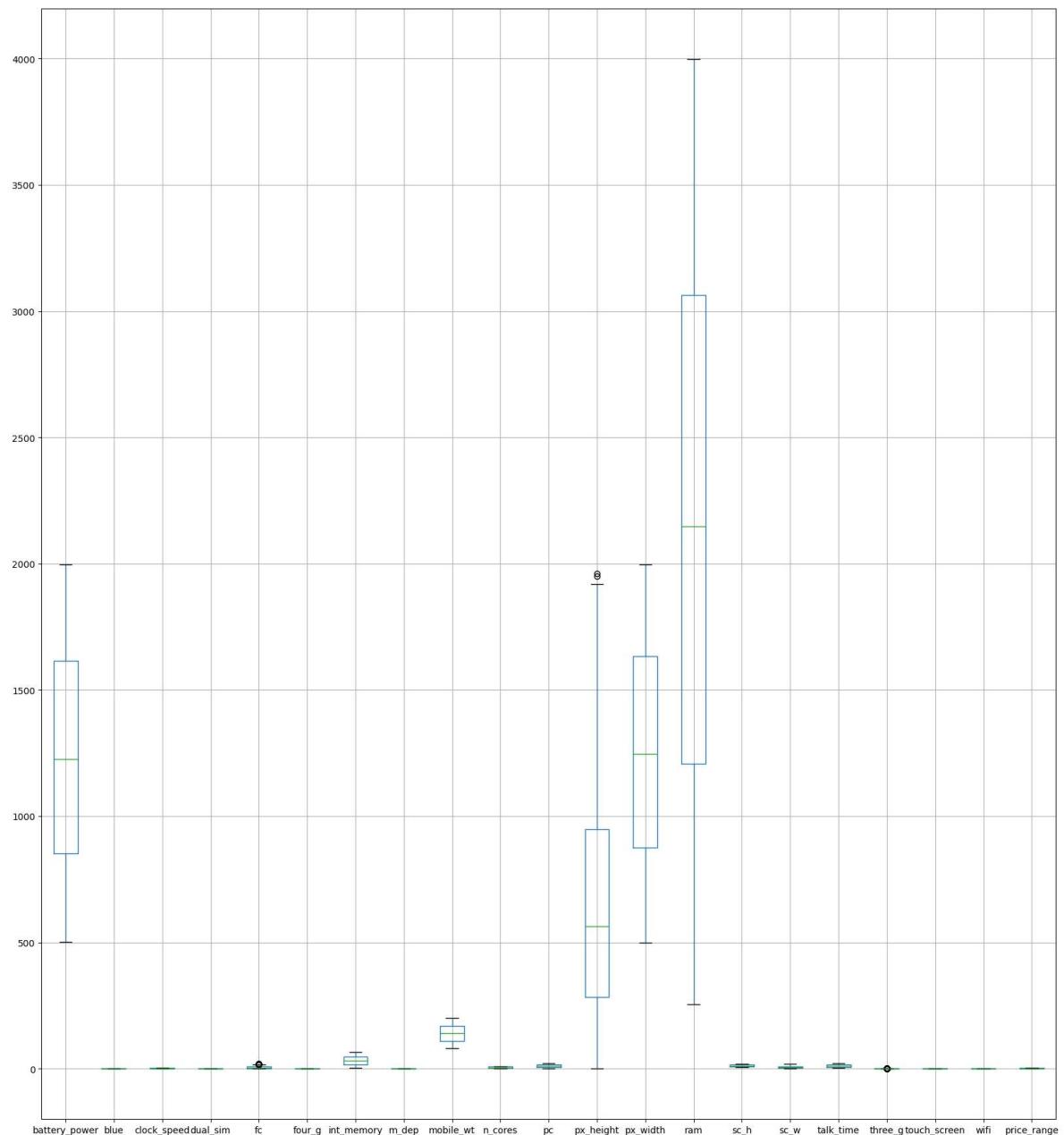
There are no null values

```
In [9]: #duplicate handling
data.duplicated().sum()
```

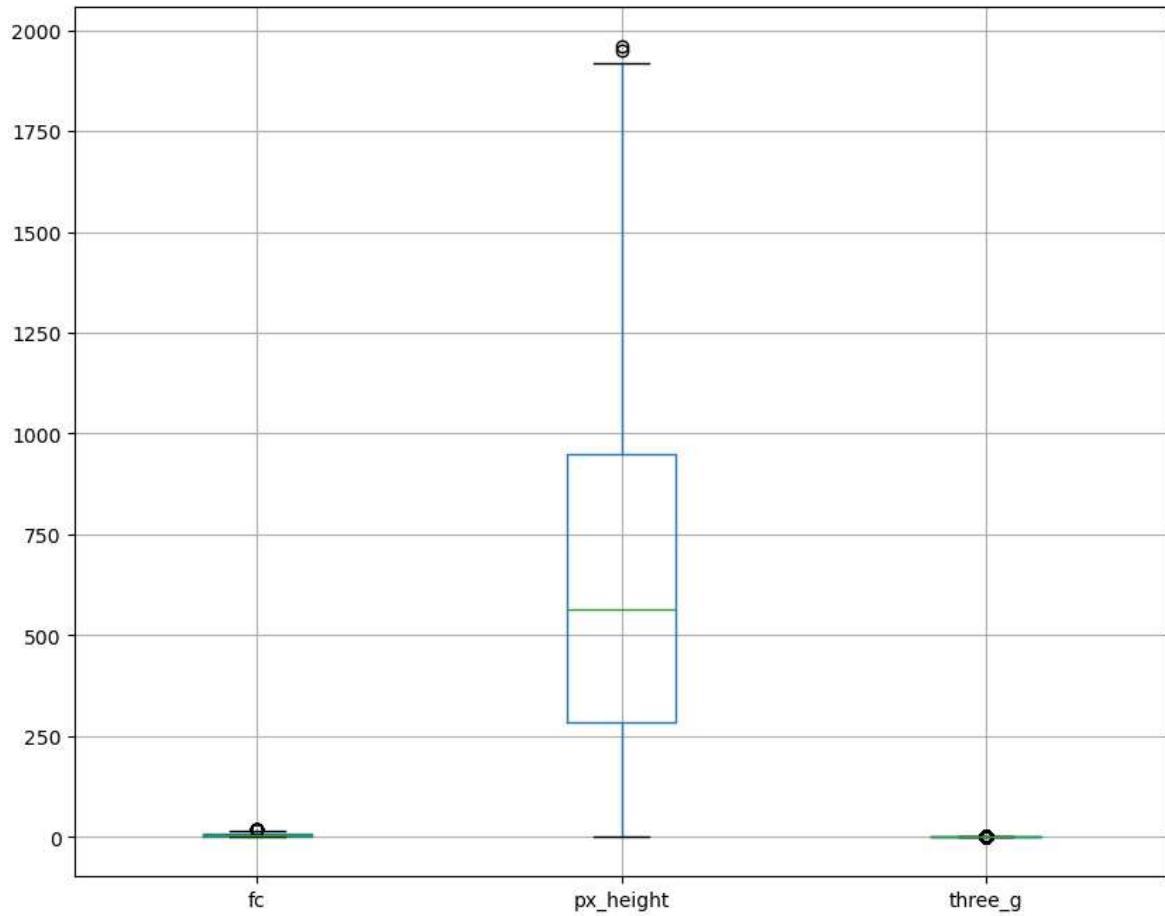
```
Out[9]: 0
```

There are no duplicated values

```
In [10]: # Outlier detection  
data.boxplot(figsize=(20,22))  
plt.show()
```



```
In [11]: data.boxplot(column=['fc', 'px_height', 'three_g'], figsize=(10,8))
plt.show()
```



```
In [12]: data.nunique()
```

```
Out[12]: battery_power    1094
blue              2
clock_speed      26
dual_sim         2
fc               20
four_g           2
int_memory       63
m_dep            10
mobile_wt        121
n_cores          8
pc               21
px_height        1137
px_width         1109
ram              1562
sc_h             15
sc_w             19
talk_time        19
three_g          2
touch_screen     2
wifi             2
price_range      4
dtype: int64
```

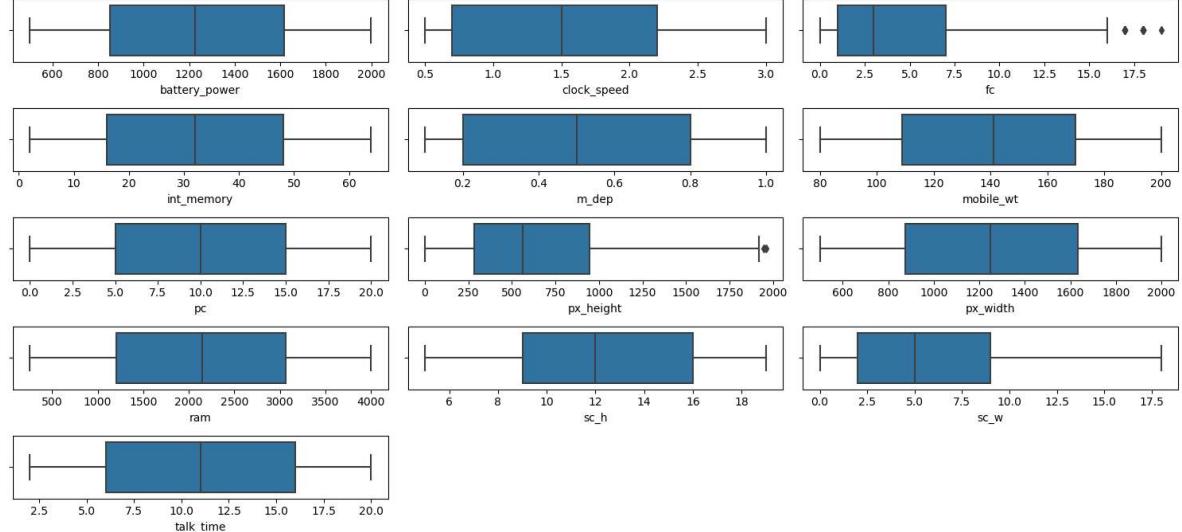
In [13]: `data['three_g'].value_counts()`

Out[13]:

1	1523
0	477
Name: three_g, dtype: int64	

In [14]:

```
num_col= ['battery_power', 'clock_speed', 'fc', 'int_memory', 'm_dep', 'mobile_wt', 'px_height', 'px_width', 'ram', 'sc_h', 'sc_w', 'talk_time', 'three_g']
plt.figure(figsize=(15, 7))
for i in range(0, len(num_col)):
    plt.subplot(5, 3, i+1)
    sns.boxplot(x=data[num_col[i]], orient='v')
    plt.tight_layout()
```



There are outliers in the columns 'fc', 'three_g' and 'px_height'. Since these are genuine values we need not replace or remove it.

Exploratory Data Analysis

In [15]: `data.head(2)`

Out[15]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_c
0	842	0	2.2	0	0	1	0	7	0.6	188
1	1021	1	0.5	1	0	1	1	53	0.7	136

2 rows × 21 columns

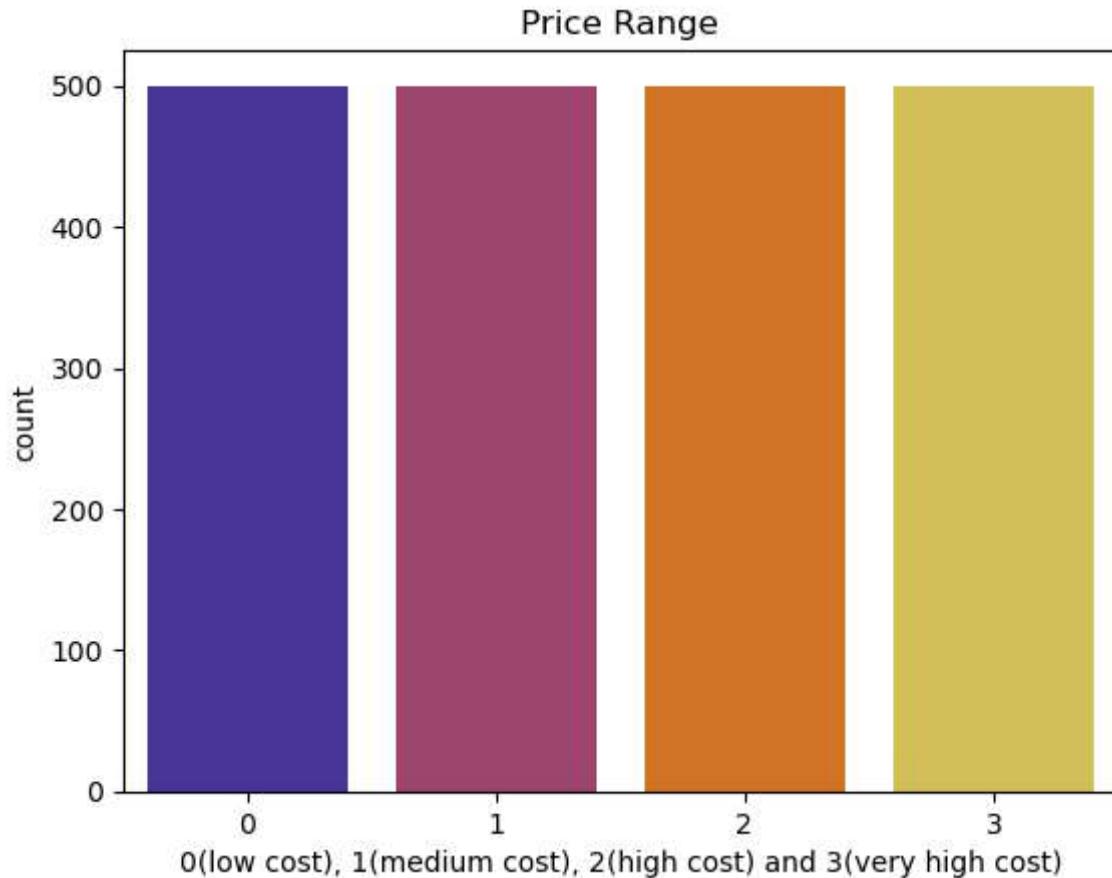
```
In [16]: data.nunique()
```

```
Out[16]: battery_power    1094
blue              2
clock_speed      26
dual_sim         2
fc                20
four_g            2
int_memory       63
m_dep             10
mobile_wt        121
n_cores           8
pc                21
px_height        1137
px_width         1109
ram               1562
sc_h              15
sc_w              19
talk_time         19
three_g           2
touch_screen      2
wifi              2
price_range       4
dtype: int64
```

```
In [17]: data['price_range'].value_counts()
```

```
Out[17]: 1    500
2    500
3    500
0    500
Name: price_range, dtype: int64
```

```
In [18]: sns.countplot(x='price_range',data=data, palette='CMRmap')
plt.title("Price Range")
plt.xlabel('0(low cost), 1(medium cost), 2(high cost) and 3(very high cost)')
plt.show()
```

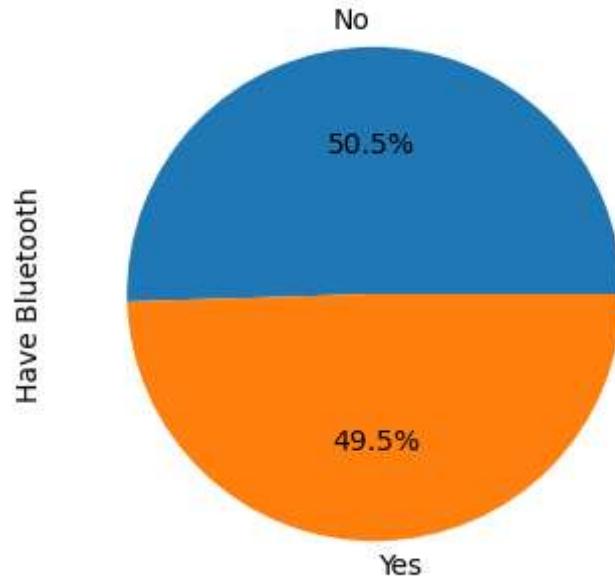


- The dataset is balanced

```
In [19]: data['blue'].value_counts()
```

```
Out[19]: 0    1010
1    990
Name: blue, dtype: int64
```

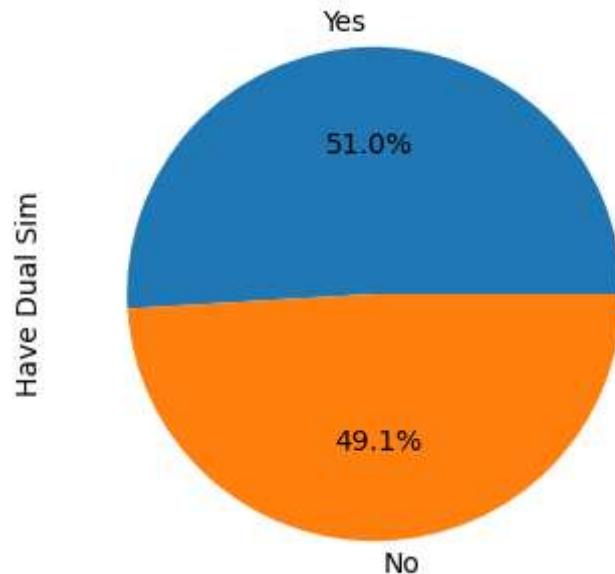
```
In [20]: labels=['No', 'Yes']
plt.figure(figsize=(5,5))
data['blue'].value_counts().plot(kind="pie", autopct='%.1f%%', labels=labels,fig
plt.ylabel('Have Bluetooth')
plt.show()
```



```
In [21]: data['dual_sim'].value_counts()
```

```
Out[21]: 1    1019
0     981
Name: dual_sim, dtype: int64
```

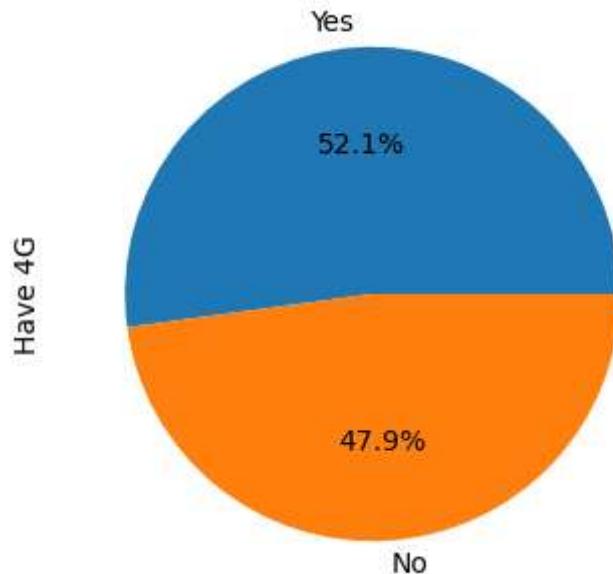
```
In [22]: labels=['Yes', 'No']
plt.figure(figsize=(5,5))
data['dual_sim'].value_counts().plot(kind="pie", autopct='%.1f%%', labels=labels
plt.ylabel('Have Dual Sim')
plt.show()
```



```
In [23]: data['four_g'].value_counts()
```

```
Out[23]: 1    1043
0    957
Name: four_g, dtype: int64
```

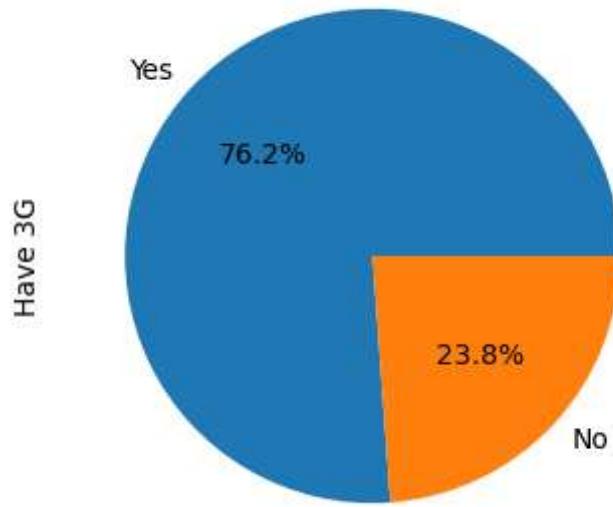
```
In [24]: labels=['Yes', 'No']
plt.figure(figsize=(5,5))
data['four_g'].value_counts().plot(kind="pie", autopct='%.1f%%', labels=labels, f
plt.ylabel('Have 4G')
plt.show()
```



```
In [25]: data['three_g'].value_counts()
```

```
Out[25]: 1    1523
0    477
Name: three_g, dtype: int64
```

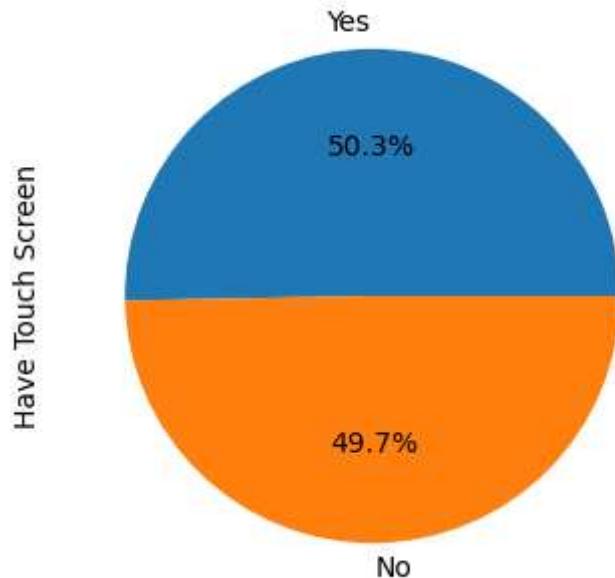
```
In [26]: labels=['Yes', 'No']
plt.figure(figsize=(5,5))
data['three_g'].value_counts().plot(kind="pie", autopct='%.1f%%', labels=labels,
plt.ylabel('Have 3G')
plt.show()
```



```
In [27]: data['touch_screen'].value_counts()
```

```
Out[27]: 1    1006
0    994
Name: touch_screen, dtype: int64
```

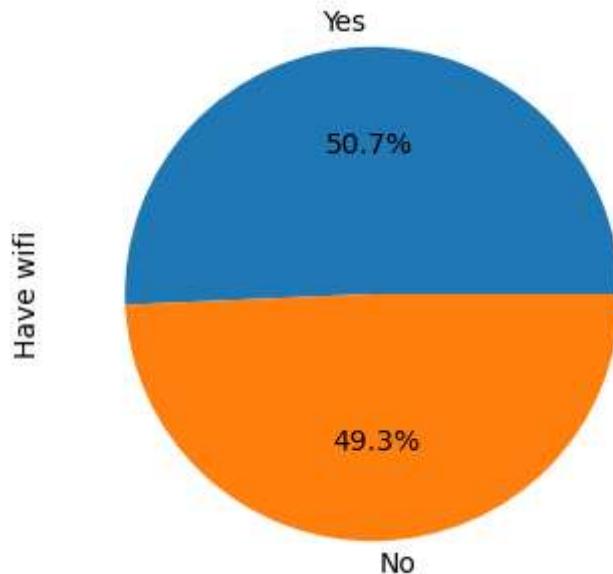
```
In [28]: labels=['Yes', 'No']
plt.figure(figsize=(5,5))
data['touch_screen'].value_counts().plot(kind="pie", autopct='%.1f%%', labels=la
plt.ylabel('Have Touch Screen')
plt.show()
```



```
In [29]: data['wifi'].value_counts()
```

```
Out[29]: 1    1014
0     986
Name: wifi, dtype: int64
```

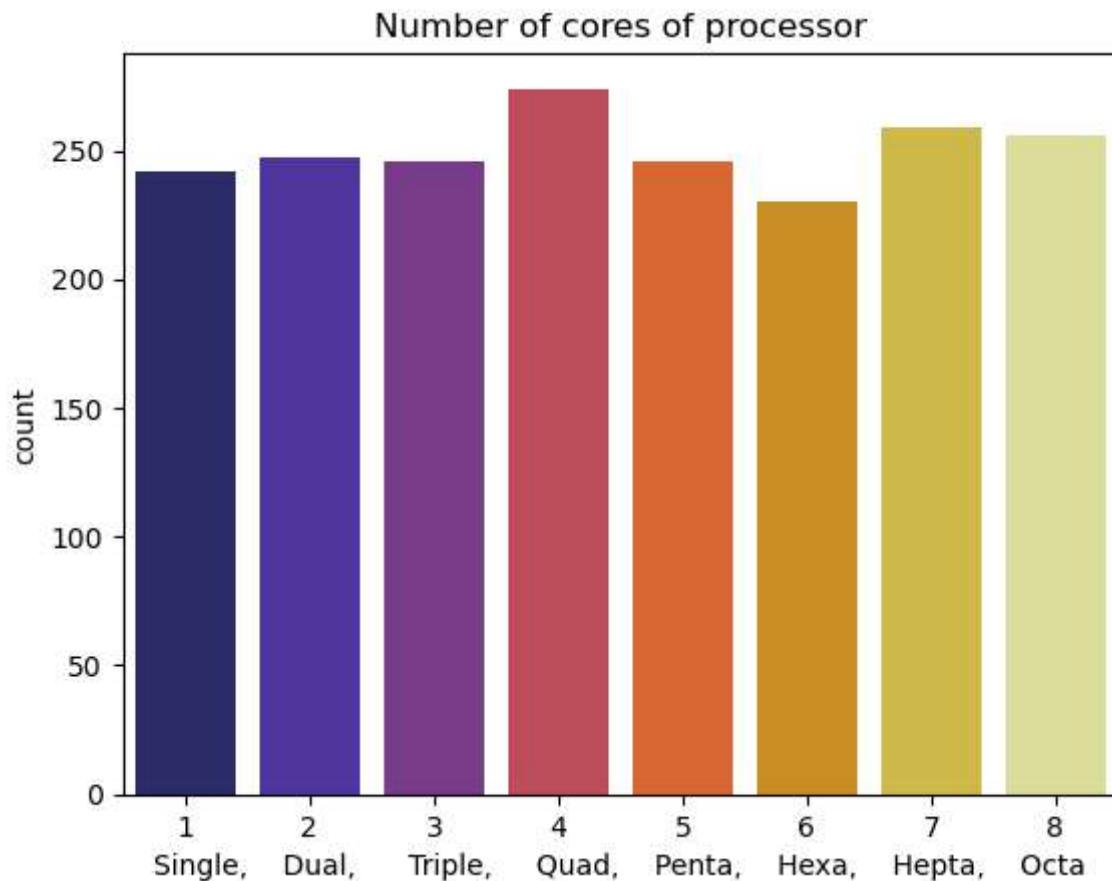
```
In [30]: labels=['Yes', 'No']
plt.figure(figsize=(5,5))
data['wifi'].value_counts().plot(kind="pie", autopct='%.1f%%', labels=labels,fig
plt.ylabel('Have wifi')
plt.show()
```



```
In [31]: data['n_cores'].value_counts()
```

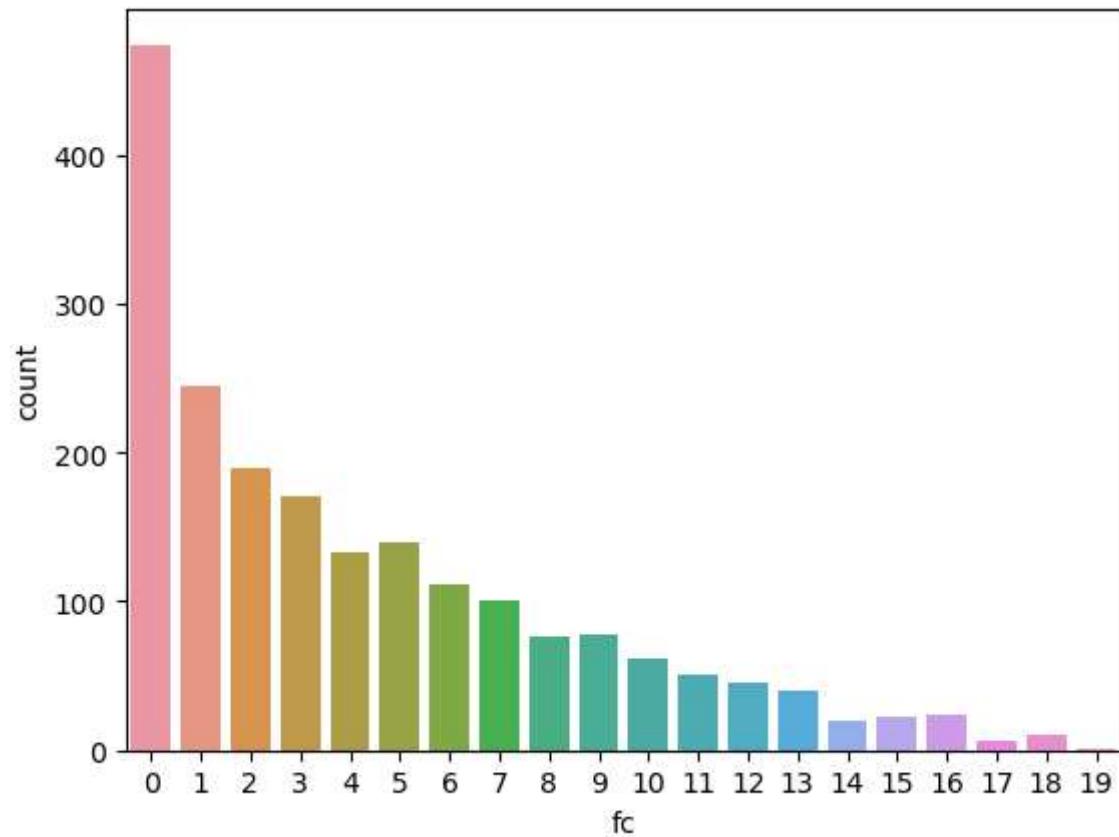
```
Out[31]: 4    274
7    259
8    256
2    247
3    246
5    246
1    242
6    230
Name: n_cores, dtype: int64
```

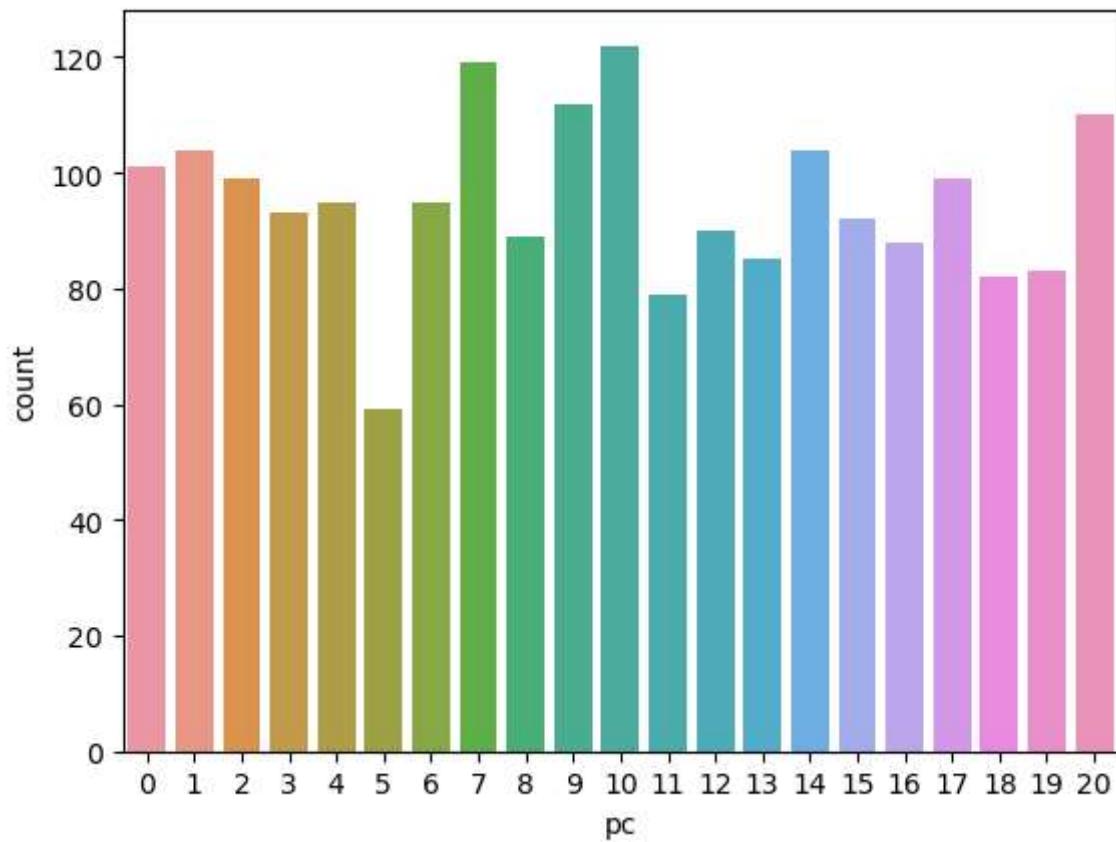
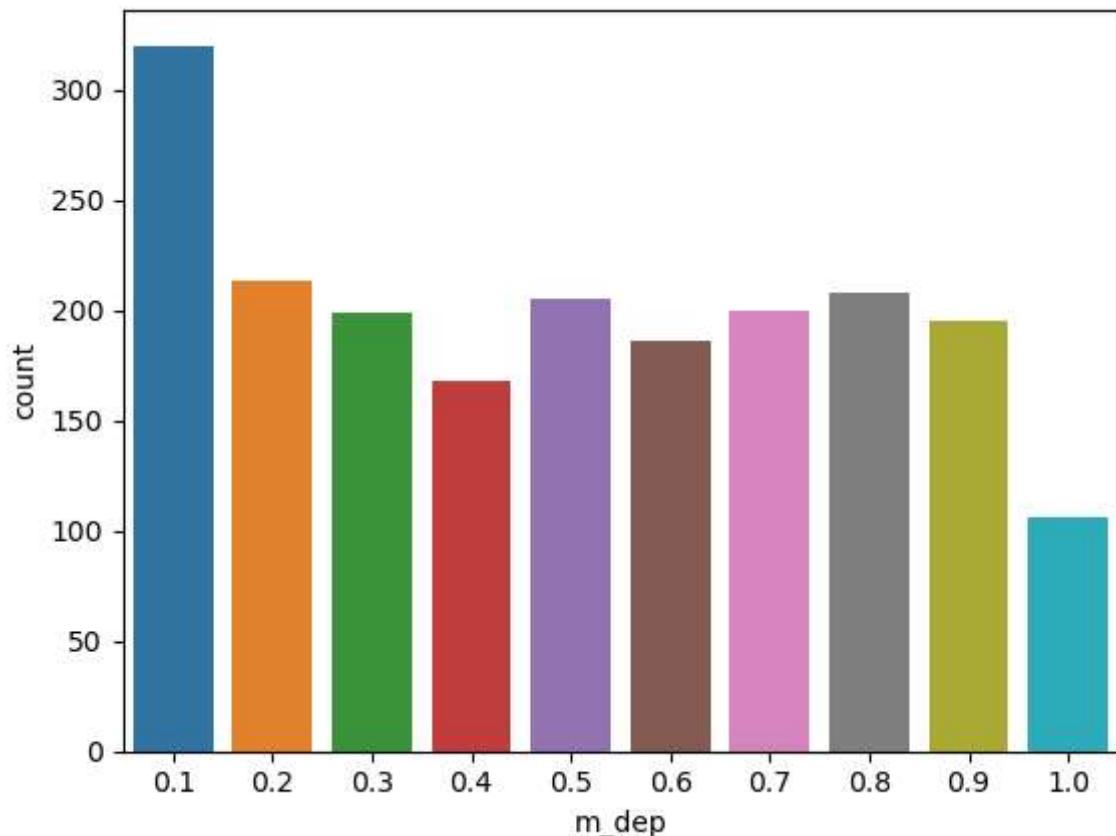
```
In [32]: sns.countplot(x='n_cores', data=data, palette='CMRmap')
plt.title("Number of cores of processor")
plt.xlabel('Single,     Dual,     Triple,     Quad,     Penta,     Hexa,     Hepta,     Octa')
plt.show()
```

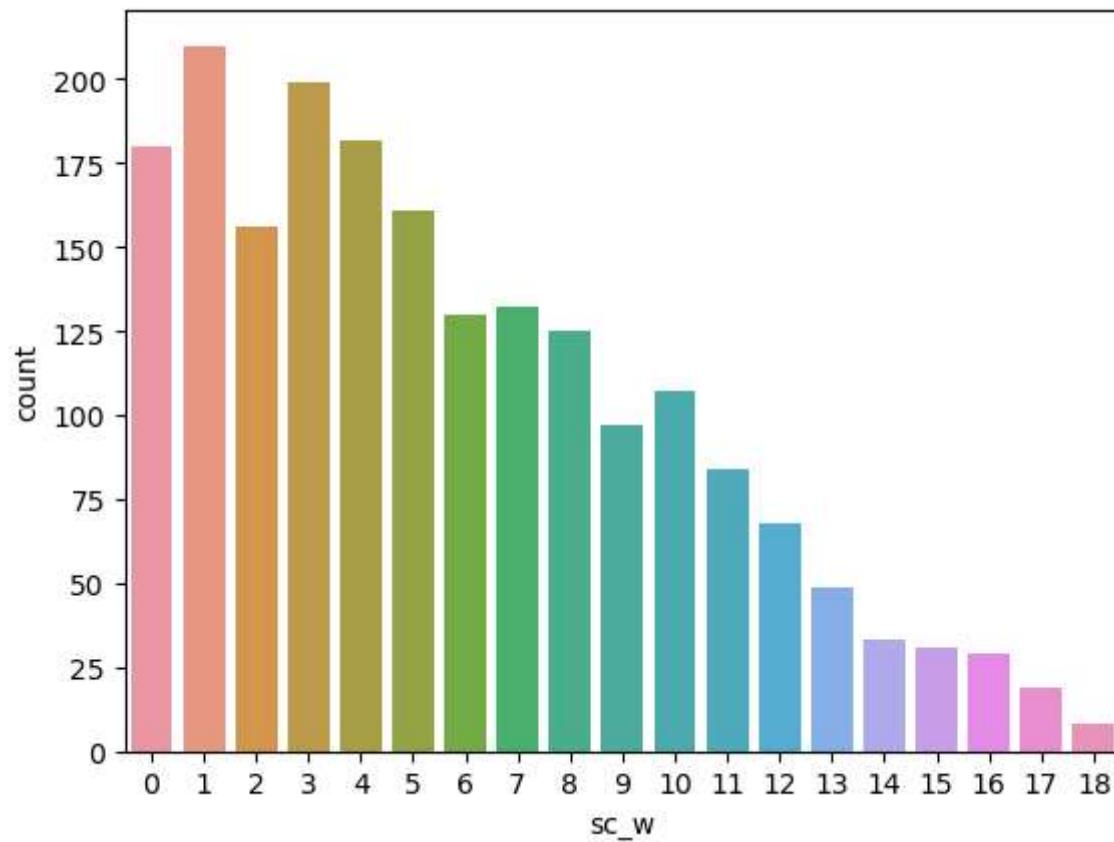
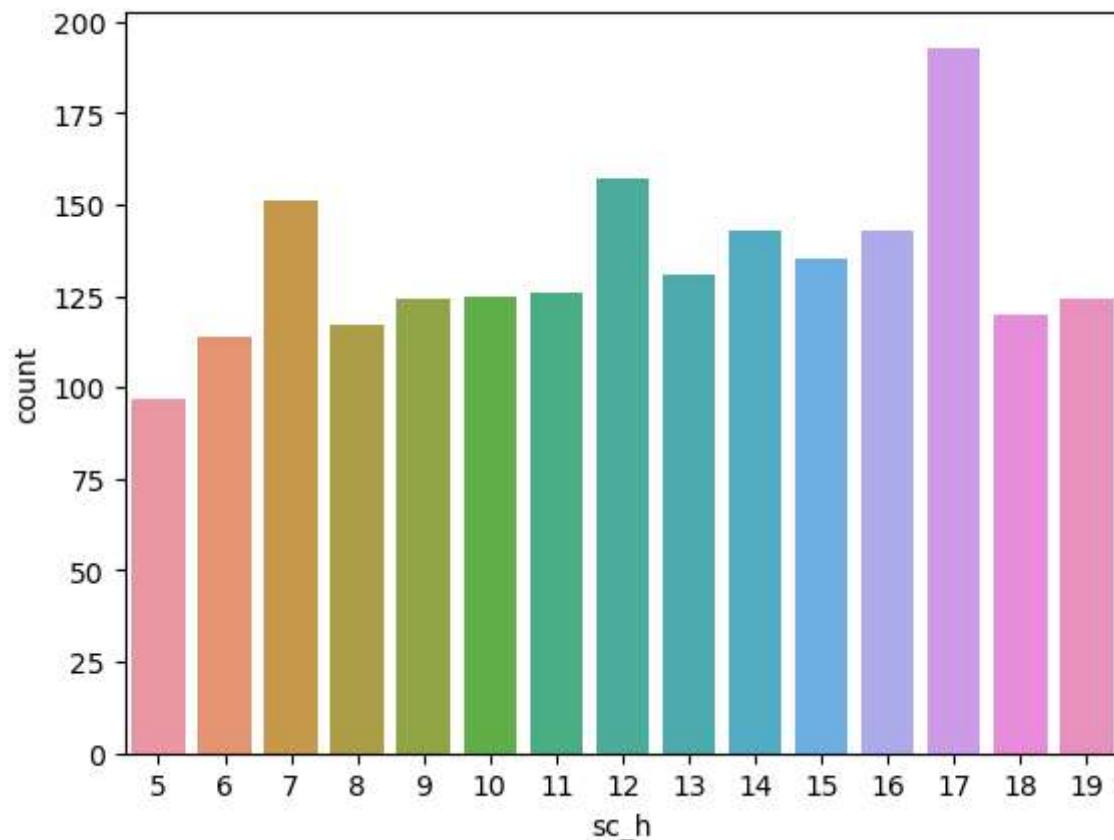


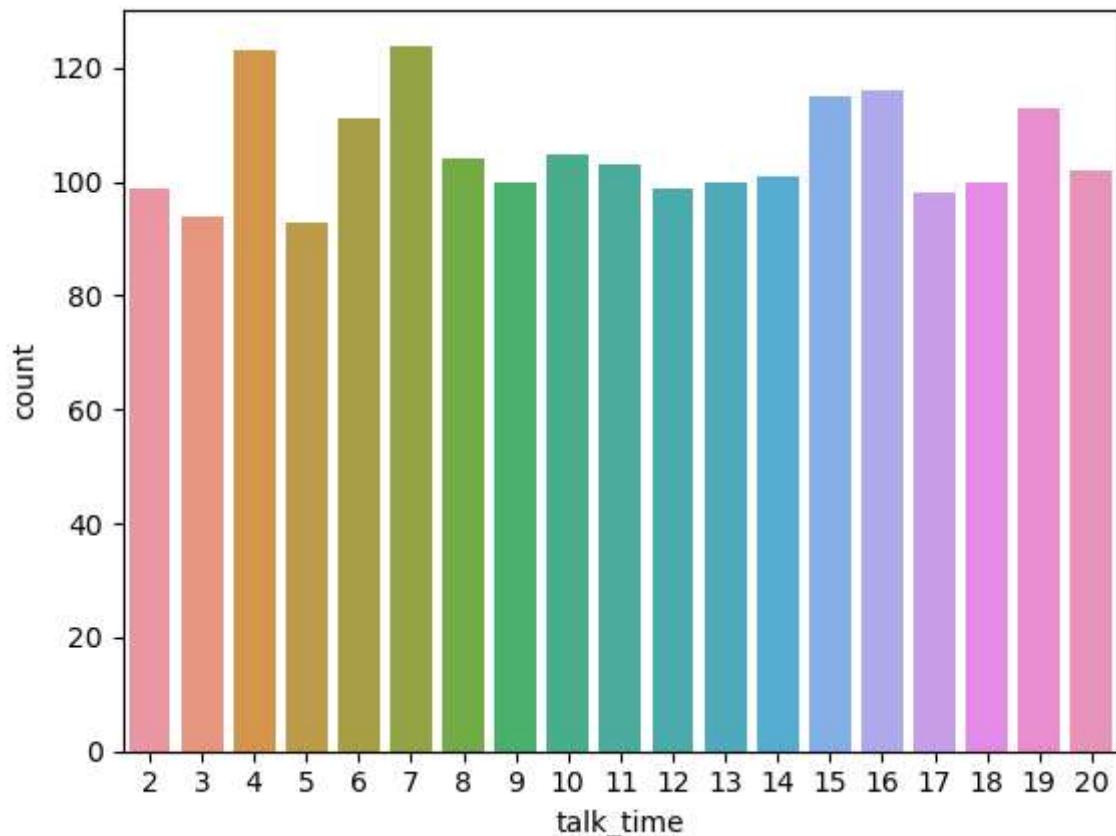
- We can see that almost half of the phones have bluetooth, dual sim, 4G, touchscreen and wifi.
- Almost three-fourth of the phones have 3G.
- The different types of cores are also spread out almost evenly.

```
In [33]: #CountPlot for various columns
for i in num_col:
    if (data[i].nunique())<=25:
        sns.countplot(x=data[i])
        plt.show()
```



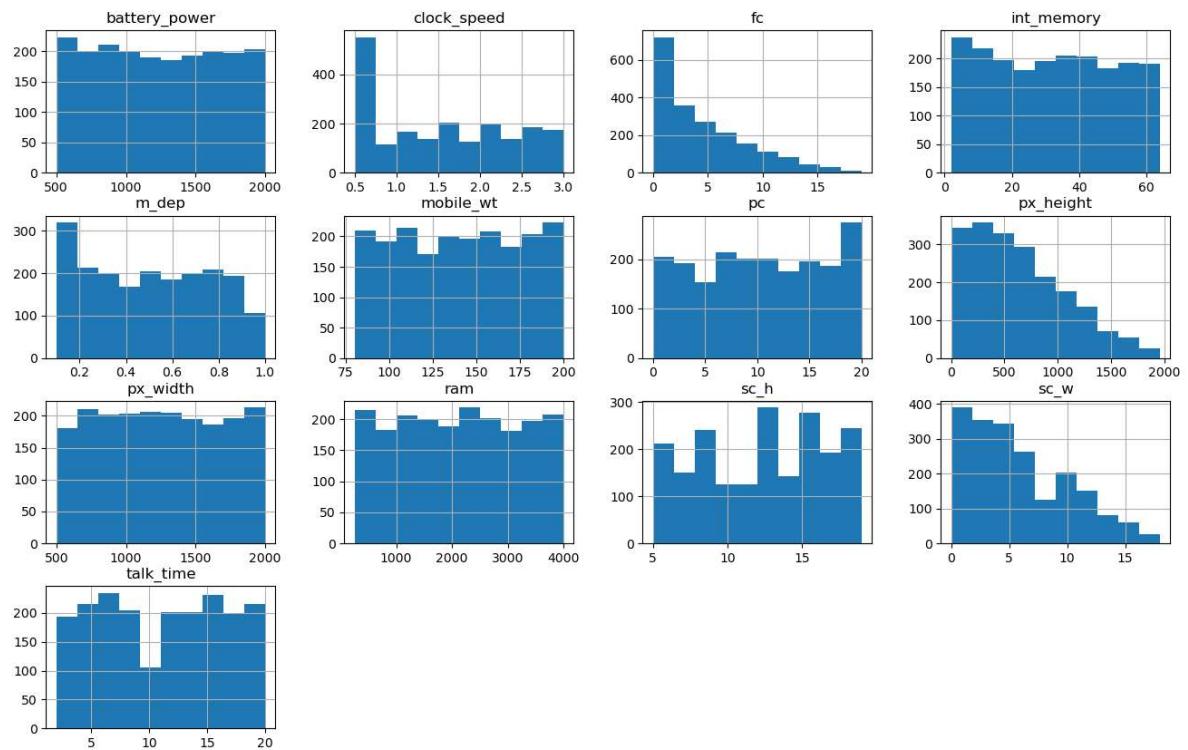






- Majority of phones in this dataset have a front camera. Only a quarter of the phone in this dataset doesn't have front camera.
- Almost 6 % of phones in this dataset doesn't have camera.
- Talktime is evenly spread out.

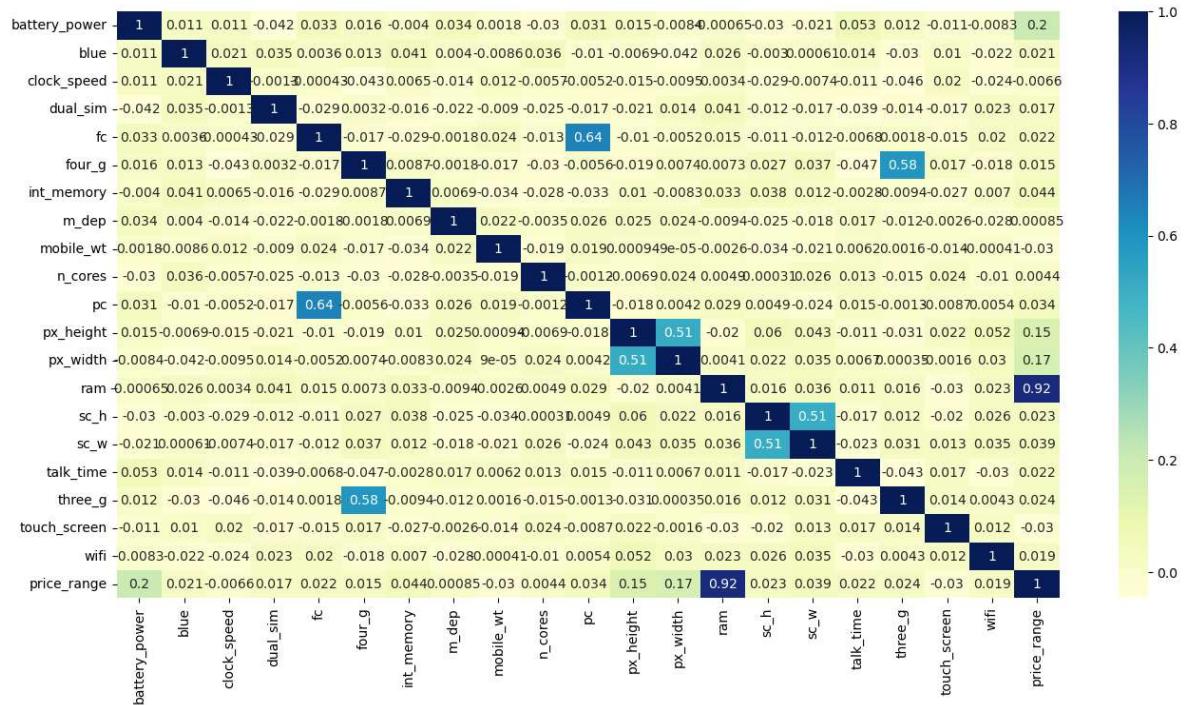
```
In [34]: data.hist(num_col,figsize=(16,10))  
plt.show()
```



- fc has exponential distribution
- the pixel height distribution is right skewed.
- others have multimodal distribution.

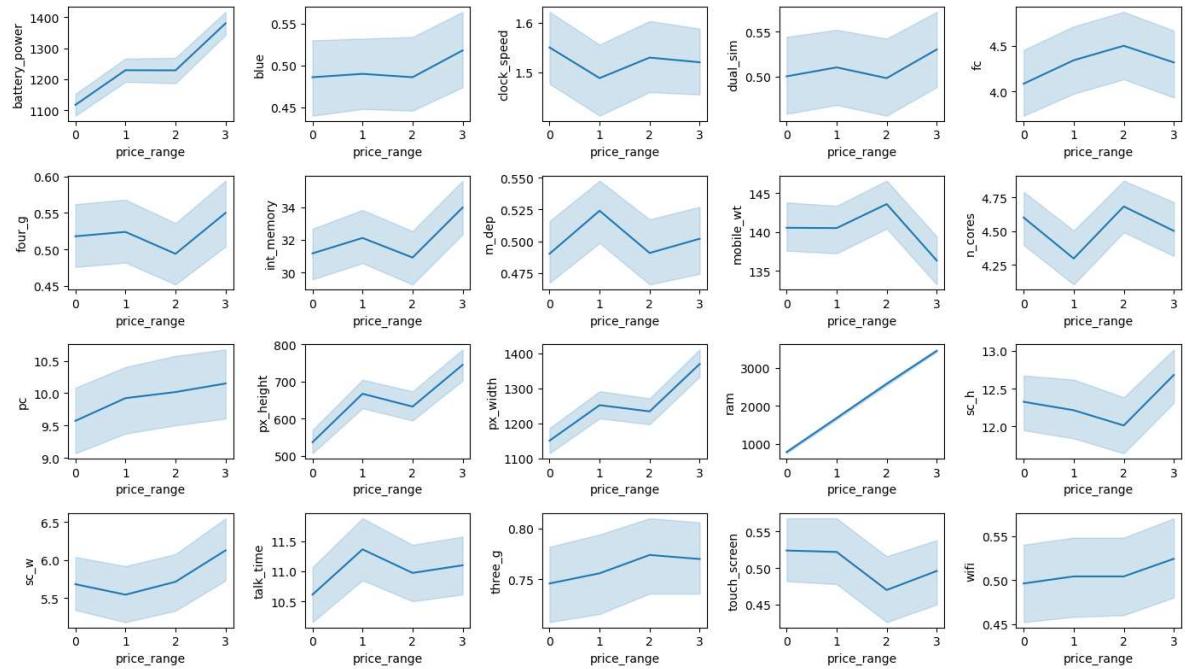
In [35]: *# plotting correlations on a heatmap*

```
plt.figure(figsize=(16,8))
sns.heatmap(data.corr(), cmap="YlGnBu", annot=True)
plt.show()
```



- Almost all features exhibit low correlation with each other.
- 'ram' is highly correlated with price range.
- None of the features are negatively correlated.

```
In [36]: a = data.drop('price_range', axis=1)
plt.figure(figsize=(14,8))
for i, j in enumerate(a):
    plt.subplot(4,5, i+1)
    sns.lineplot(y=data[j], x=data["price_range"])
plt.tight_layout()
```



- The higher the price range higher the average ram size.
- We can see that the higher the price range it tends to have a higher battery power except on the price range 1 to 2. It seems that from price range 1 to 2 there's no significant difference in the average battery power.
- Clock speed is highest for mobiles in lower price range.
- The price range of 0 has more products with lower pixel width and pixel height while the highest price range has more products with higher pixel width and pixel height.
- It seems that talk time doesn't really affect the price range.
- Higher price range has a higher megapixel primary camera.

Key Findings:

- All columns are numerical.
- We can see that almost half of the phones have bluetooth, dual sim, 4G, touchscreen and wifi.
- Almost three-fourth of the phones have 3G.
- The different types of cores are also spread out almost evenly.
- Majority of phones in this dataset have a front camera. Only a quarter of the phone in this dataset doesn't have front camera.
- Almost 6 % of phones in this dataset doesn't have camera.
- Talktime is evenly spread out.
- fc has exponential distribution
- the pixel height distribution is right skewed.

- others have multimodal distribution.
- Darker shades represent low correlation
- All features exhibit low correlation with each other.
- 'ram' is highly correlated with price range.
- None of the features are negatively correlated.

In [37]: `data.head()`

Out[37]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_c
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	

5 rows × 21 columns



In [38]: `data.shape`

Out[38]: (2000, 21)

Splitting Independent and dependent features

In [39]: `# Split the dataset into features and target`
`x = data.drop('price_range', axis=1)`
`y = data['price_range']`

In [40]: `# Perform 80 : 20 Train-Test split`
`from sklearn.model_selection import train_test_split`
`x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2, random_state=42)`

In [41]: `# Observe the shapes`
`x_train.shape, y_train.shape, x_test.shape, y_test.shape`

Out[41]: ((1600, 20), (1600,), (400, 20), (400,))

In [42]: `x_train.head()`

Out[42]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt
1849	1786	0	1.2	0	11	0	61	0.1	123
1866	907	1	2.6	1	7	0	44	0.2	160
908	1254	0	1.7	0	0	1	45	0.4	146
614	539	1	0.6	1	9	0	21	0.1	145
1628	999	0	2.6	1	11	1	56	1.0	165

Feature Scaling

- Using Normalization (Min-Max Scaling)

In [43]: `# Import the class StandardScaler
from sklearn.preprocessing import MinMaxScaler
Create an instance of the class
scaler = MinMaxScaler()
Use the fit_transform method to perform the transformation on train data
x_train.loc[:,:] = scaler.fit_transform(x_train)
Use the fit_transform method to perform the transformation on train data
x_test.loc[:,:] = scaler.transform(x_test)` #Normalization

In [44]: `x_train.head()`

Out[44]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mol
1849	0.858383	0.0	0.28	0.0	0.578947	0.0	0.951613	0.000000	0.1
1866	0.271209	1.0	0.84	1.0	0.368421	0.0	0.677419	0.111111	0.6
908	0.503006	0.0	0.48	0.0	0.000000	1.0	0.693548	0.333333	0.1
614	0.025384	1.0	0.04	1.0	0.473684	0.0	0.306452	0.000000	0.1
1628	0.332665	0.0	0.84	1.0	0.578947	1.0	0.870968	1.000000	0.1

Model Building and Evaluation

1. Desicion Tree Classifier

```
In [45]: from sklearn.tree import DecisionTreeClassifier
dt=DecisionTreeClassifier(max_depth=3)
dt_model=dt.fit(x_train,y_train)
y_pred=dt_model.predict(x_test)
```

```
In [46]: from sklearn.metrics import confusion_matrix , accuracy_score, precision_score
print('Accuracy is: ',accuracy_score(y_test,y_pred))
```

Accuracy is: 0.7275

```
In [47]: import sklearn.metrics as metrics
score=100*(metrics.f1_score(y_test,y_pred,average='macro'))
score
```

Out[47]: 72.75173762379924

```
In [48]: print(classification_report(y_test,y_pred))
```

	precision	recall	f1-score	support
0	0.88	0.71	0.78	100
1	0.57	0.84	0.68	100
2	0.71	0.52	0.60	100
3	0.85	0.84	0.84	100
accuracy			0.73	400
macro avg	0.75	0.73	0.73	400
weighted avg	0.75	0.73	0.73	400

2. Random Forest Classifier

```
In [49]: from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier()
rf_model=rf.fit(x_train,y_train)
y_pred=rf_model.predict(x_test)
```

```
In [50]: print('Accuracy is: ',accuracy_score(y_test,y_pred))
```

Accuracy is: 0.8625

```
In [51]: score=100*(metrics.f1_score(y_test,y_pred,average='macro'))
score
```

Out[51]: 86.12827268927497

```
In [52]: print(classification_report(y_test,y_pred))
```

	precision	recall	f1-score	support
0	0.86	0.97	0.91	100
1	0.83	0.75	0.79	100
2	0.81	0.83	0.82	100
3	0.95	0.90	0.92	100
accuracy			0.86	400
macro avg	0.86	0.86	0.86	400
weighted avg	0.86	0.86	0.86	400

3.Gradient Boosting Classifier

```
In [53]: from sklearn.ensemble import GradientBoostingClassifier
# Define the gradient boosting classifier model
gb = GradientBoostingClassifier(random_state=42)

# Train the model on the training data
gb_model=gb.fit(x_train, y_train)
y_pred=gb_model.predict(x_test)
```

```
In [54]: print('Accuracy is: ',accuracy_score(y_test,y_pred))
```

Accuracy is: 0.8775

```
In [55]: score=100*(metrics.f1_score(y_test,y_pred,average='macro'))
score
```

Out[55]: 87.7220927562373

```
In [56]: print(classification_report(y_test,y_pred))
```

	precision	recall	f1-score	support
0	0.89	0.95	0.92	100
1	0.86	0.81	0.84	100
2	0.83	0.86	0.84	100
3	0.94	0.89	0.91	100
accuracy			0.88	400
macro avg	0.88	0.88	0.88	400
weighted avg	0.88	0.88	0.88	400

4. XG Boost Classifier

```
In [57]: import xgboost as xgb

# define the XGBoost model
model = xgb.XGBClassifier()
# train the model on the training data
xgb_model=model.fit(x_train, y_train)
# make predictions on the testing data
y_pred = xgb_model.predict(x_test)
```

```
In [58]: # evaluate the accuracy of the model
print('Accuracy is: ',accuracy_score(y_test,y_pred))
```

Accuracy is: 0.905

```
In [59]: score=100*(metrics.f1_score(y_test,y_pred,average='macro'))
score
```

Out[59]: 90.46921223087027

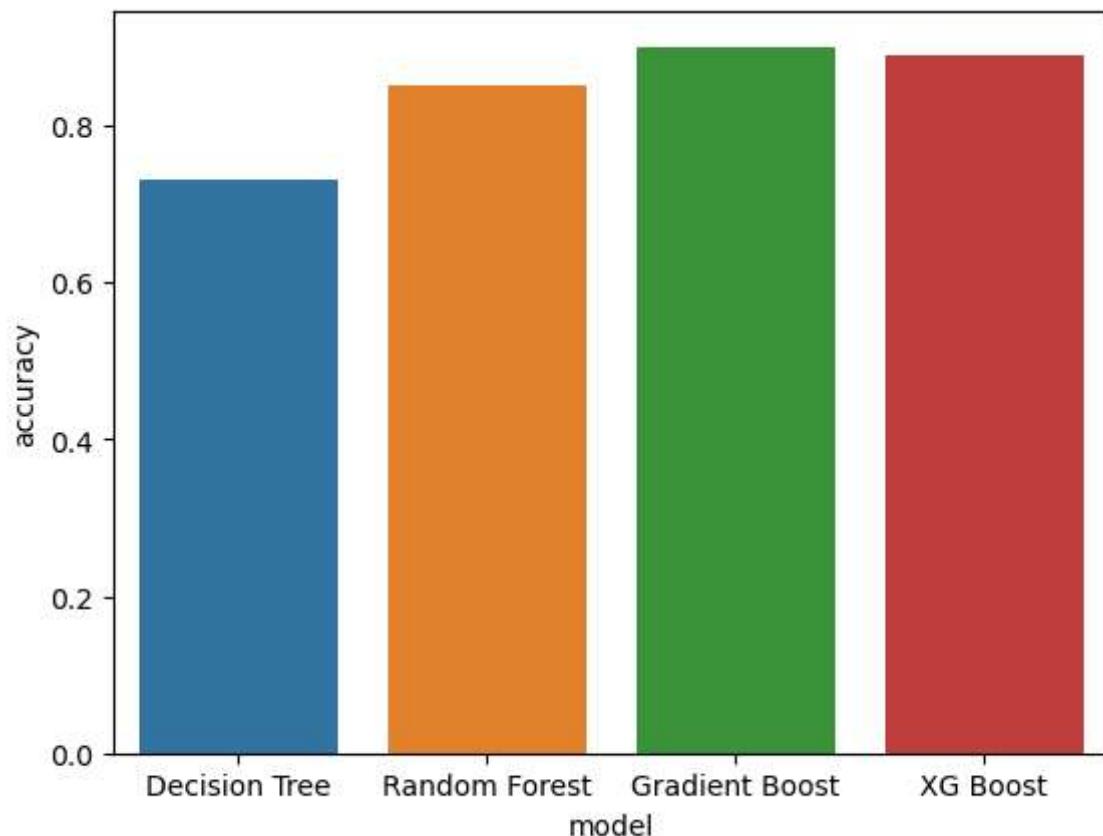
```
In [60]: print(classification_report(y_test,y_pred))
```

	precision	recall	f1-score	support
0	0.90	0.98	0.94	100
1	0.93	0.83	0.88	100
2	0.84	0.92	0.88	100
3	0.96	0.89	0.92	100
accuracy			0.91	400
macro avg	0.91	0.91	0.90	400
weighted avg	0.91	0.91	0.90	400

Choosing Best Model

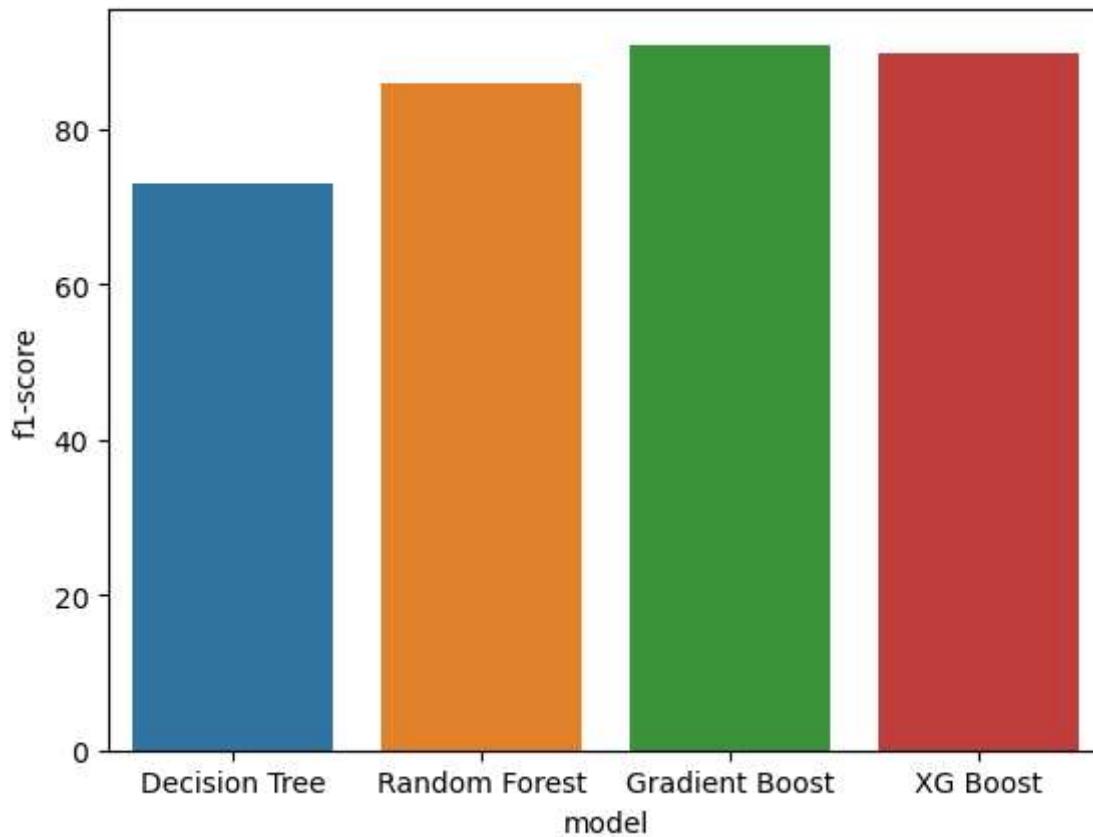
```
In [61]: classifiers=["Decision Tree", "Random Forest", 'Gradient Boost', 'XG Boost']  
accuracy = [0.73,0.85,0.90,0.89]  
f1_score = [72.9,85.8,90.8,89.75]  
df_af=pd.DataFrame({'model':classifiers,"accuracy":accuracy,"f1-score":f1_score})  
sns.barplot(data=df_af,x="model",y="accuracy")
```

```
Out[61]: <AxesSubplot:xlabel='model', ylabel='accuracy'>
```



```
In [62]: sns.barplot(data=df_af,x="model",y="f1-score")
```

```
Out[62]: <AxesSubplot:xlabel='model', ylabel='f1-score'>
```



Gradient Boost Classifier is the best model.

- We need to identify the most important features and rank the features based on their importance scores.
- We could Identify the most important features using feature selection techniques such as correlation analysis, mutual information, and feature importance scores from machine learning models.

Identifying important features and ranking them

```
In [63]: # Compute the feature importance scores using a machine Learning model
importance_scores = gb_model.feature_importances_

# Create a DataFrame to store the feature names and their importance scores
feature_scores = pd.DataFrame({'Feature': x.columns, 'Importance': importance_scores})

# Sort the features based on their importance scores in descending order
feature_scores = feature_scores.sort_values(by='Importance', ascending=False)

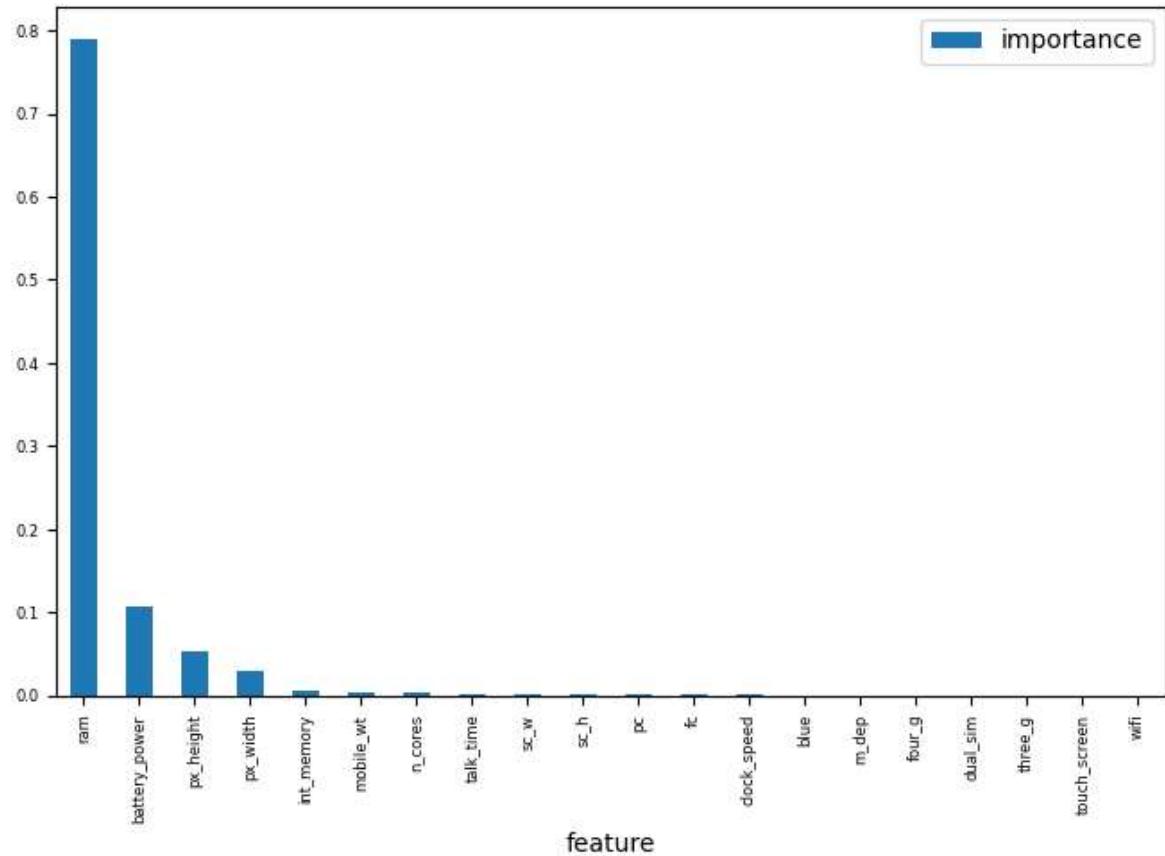
# Rank the features based on their importance scores
feature_scores['Rank'] = np.arange(1, len(x.columns) + 1)

# Display the ranked features
print(feature_scores[['Rank', 'Feature', 'Importance']])
```

	Rank	Feature	Importance
13	1	ram	0.789013
0	2	battery_power	0.107990
11	3	px_height	0.053234
12	4	px_width	0.029948
6	5	int_memory	0.005609
8	6	mobile_wt	0.004876
9	7	n_cores	0.002508
14	8	sc_h	0.001418
15	9	sc_w	0.001267
16	10	talk_time	0.001085
10	11	pc	0.000994
4	12	fc	0.000544
2	13	clock_speed	0.000536
7	14	m_dep	0.000463
5	15	four_g	0.000349
3	16	dual_sim	0.000125
18	17	touch_screen	0.000033
19	18	wifi	0.000011
1	19	blue	0.000000
17	20	three_g	0.000000

In [64]: *# Plotting features according to its importance*

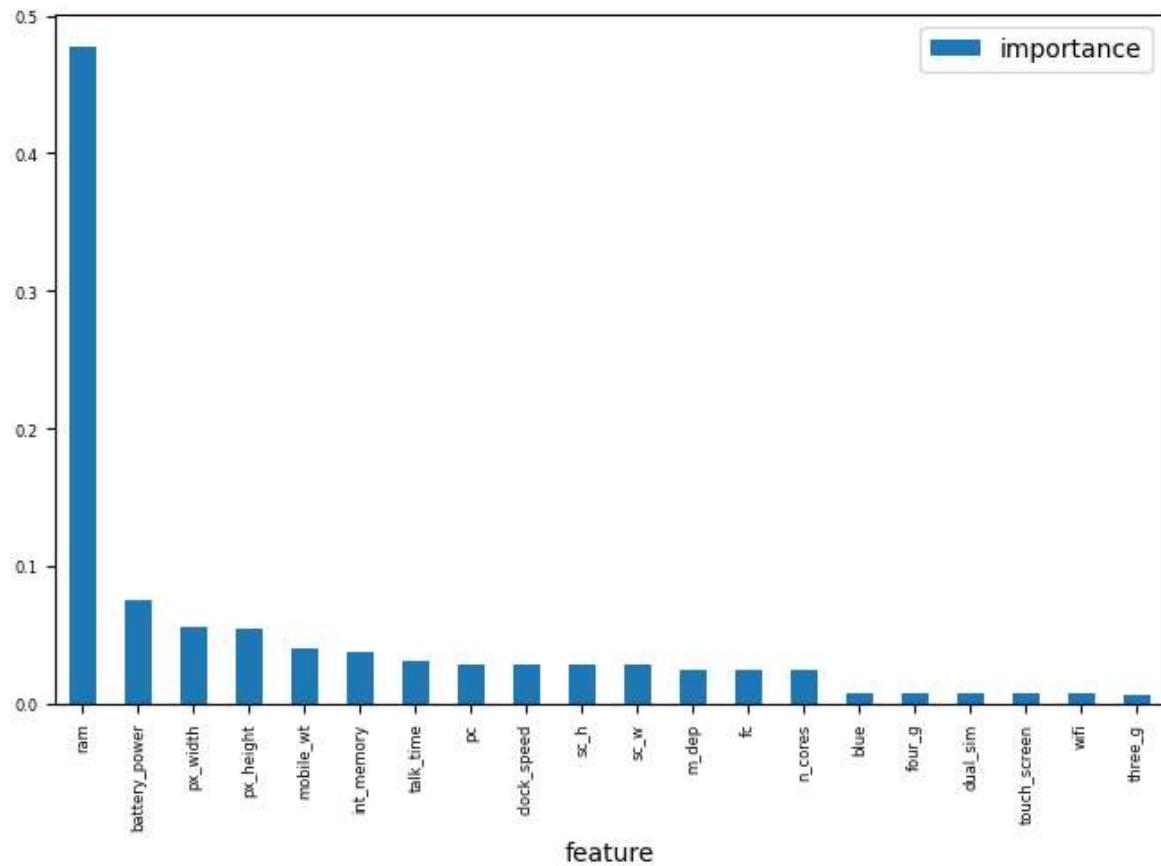
```
importances = pd.DataFrame({'feature': x.columns, 'importance': np.round(gb_m
importances = importances.sort_values('importance', ascending=False)
importances.plot.bar(x='feature', figsize=(8,5), fontsize=6)
plt.show()
```



- For gradient boosting model

In [65]: # Most important features are

```
importances = pd.DataFrame({'feature': x.columns, 'importance': np.round(rf_mc
importances = importances.sort_values('importance', ascending=False)
importances.plot.bar(x='feature', figsize=(8,5), fontsize=6)
plt.show()
```



- For Random Forest model

- - - - - X X X X X X X X - - - - -

In []: