Mobile Price Classification

The dataset for this project originates from the [Classify Mobile Price Range]

Context:

Bob has started his own mobile company. He wants to give tough fight to big companies like Apple,Samsung etc. He does not know how to estimate price of mobiles his company creates. In this competitive mobile phone market you cannot simply assume things. To solve this problem he collects sales data of mobile phones of various companies.

Bob wants to find out some relation between features of a mobile phone(eg:- RAM,Internal Memory etc) and its selling price. But he is not so good at Machine Learning. So he needs your help to solve this problem.

Problem Statement:

In this problem you do not have to predict actual price but a price range indicating how high the price is

Attribute Information

The data is contains values of different features of a mobile collected from different sources. The objective is caluclate price range of a mobile.

Input variables:

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

2) **blue**: Has bluetooth or not

3) **clock_speed**: speed at which microprocessor executes instructions

4) **dual_sim**: Has dual sim support or not

5) **fc**: Front Camera mega pixels

6) **four_g**: Has 4G or not

7) **int_memory**: Internal Memory in Gigabytes

8) **m_dep**: Mobile Depth in cm

9) **mobile_wt**: Weight of mobile phone

10) **n_cores**: Number of cores of processor

11) **pc**: Primary Camera mega pixels

12) **px_height**: Pixel Resolution Height

13) **px_width**: Pixel Resolution Width

14) **ram**: Random Access Memory in Mega Bytes

15) **sc_h**: Screen Height of mobile in cm

16) **sc_w**: Screen Width of mobile in cm

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

18) **three_g**: Has 3G or not

19) **touch_screen**: Has touch screen or not

20) **wifi**: Has wifi or not

Output variable (desired target):

21) **price_range**: This is the target variable with value of 0 (cheap), 1 (mid priced), 2 (costly) and 3(expensive).

Table of Contents

1. Environment Setup

goto toc

1.1. Install Packages

Install required packages

goto toc

In [130…

```
# Install pandas
! pip install pandas

# Install matplotlib
! pip install matplotlib

# Install seaborn
! pip install seaborn

# Install sklearn
! pip install sklearn

# Install tqdm to visualize iterations
! pip install tqdm
```

```
Requirement already satisfied: pandas in c:\users\arun\anaconda3\envs\data_science\l
ib\site-packages (1.2.4)
Requirement already satisfied: numpy>=1.16.5 in c:\users\arun\anaconda3\envs\data_sc
ience\lib\site-packages (from pandas) (1.20.1)
Requirement already satisfied: python-dateutil>=2.7.3 in c:\users\arun\anaconda3\env
s\data_science\lib\site-packages (from pandas) (2.8.1)
Requirement already satisfied: pytz>=2017.3 in c:\users\arun\anaconda3\envs\data_sci
ence\lib\site-packages (from pandas) (2021.1)
Requirement already satisfied: six>=1.5 in c:\users\arun\anaconda3\envs\data_science
\lib\site-packages (from python-dateutil>=2.7.3->pandas) (1.15.0)
Requirement already satisfied: matplotlib in c:\users\arun\anaconda3\envs\data_scien
ce\lib\site-packages (3.3.4)
Requirement already satisfied: python-dateutil>=2.1 in c:\users\arun\anaconda3\envs
\data_science\lib\site-packages (from matplotlib) (2.8.1)
Requirement already satisfied: numpy>=1.15 in c:\users\arun\anaconda3\envs\data_scie
nce\lib\site-packages (from matplotlib) (1.20.1)
Requirement already satisfied: cycler>=0.10 in c:\users\arun\anaconda3\envs\data_sci
ence\lib\site-packages (from matplotlib) (0.10.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in c:\users
\arun\anaconda3\envs\data_science\lib\site-packages (from matplotlib) (2.4.7)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\arun\anaconda3\envs\dat
a_science\lib\site-packages (from matplotlib) (1.3.1)
Requirement already satisfied: pillow>=6.2.0 in c:\users\arun\anaconda3\envs\data_sc
ience\lib\site-packages (from matplotlib) (8.2.0)
Requirement already satisfied: six in c:\users\arun\anaconda3\envs\data_science\lib
\site-packages (from cycler>=0.10->matplotlib) (1.15.0)
Requirement already satisfied: seaborn in c:\users\arun\anaconda3\envs\data_science
\lib\site-packages (0.11.1)
Requirement already satisfied: scipy>=1.0 in c:\users\arun\anaconda3\envs\data_scien
ce\lib\site-packages (from seaborn) (1.6.2)
Requirement already satisfied: pandas>=0.23 in c:\users\arun\anaconda3\envs\data_sci
ence\lib\site-packages (from seaborn) (1.2.4)
Requirement already satisfied: numpy>=1.15 in c:\users\arun\anaconda3\envs\data_scie
nce\lib\site-packages (from seaborn) (1.20.1)
Requirement already satisfied: matplotlib>=2.2 in c:\users\arun\anaconda3\envs\data_
science\lib\site-packages (from seaborn) (3.3.4)
Requirement already satisfied: cycler>=0.10 in c:\users\arun\anaconda3\envs\data_sci
ence\lib\site-packages (from matplotlib>=2.2->seaborn) (0.10.0)
Requirement already satisfied: python-dateutil>=2.1 in c:\users\arun\anaconda3\envs
\data_science\lib\site-packages (from matplotlib>=2.2->seaborn) (2.8.1)
Requirement already satisfied: pillow>=6.2.0 in c:\users\arun\anaconda3\envs\data_sc
ience\lib\site-packages (from matplotlib>=2.2->seaborn) (8.2.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in c:\users
\arun\anaconda3\envs\data_science\lib\site-packages (from matplotlib>=2.2->seaborn)
(2.4.7)
```

```
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\arun\anaconda3\envs\dat
a_science\lib\site-packages (from matplotlib>=2.2->seaborn) (1.3.1)
Requirement already satisfied: six in c:\users\arun\anaconda3\envs\data_science\lib
\site-packages (from cycler>=0.10->matplotlib>=2.2->seaborn) (1.15.0)
Requirement already satisfied: pytz>=2017.3 in c:\users\arun\anaconda3\envs\data_sci
ence\lib\site-packages (from pandas>=0.23->seaborn) (2021.1)
Requirement already satisfied: sklearn in c:\users\arun\anaconda3\envs\data_science
\lib\site-packages (0.0)
Requirement already satisfied: scikit-learn in c:\users\arun\anaconda3\envs\data_sci
ence\lib\site-packages (from sklearn) (0.24.1)
Requirement already satisfied: scipy>=0.19.1 in c:\users\arun\anaconda3\envs\data_sc
ience\lib\site-packages (from scikit-learn->sklearn) (1.6.2)
Requirement already satisfied: joblib>=0.11 in c:\users\arun\anaconda3\envs\data_sci
ence\lib\site-packages (from scikit-learn->sklearn) (1.0.1)
Requirement already satisfied: numpy>=1.13.3 in c:\users\arun\anaconda3\envs\data_sc
ience\lib\site-packages (from scikit-learn->sklearn) (1.20.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\arun\anaconda3\envs
\data_science\lib\site-packages (from scikit-learn->sklearn) (2.1.0)
Requirement already satisfied: tqdm in c:\users\arun\anaconda3\envs\data_science\lib
\site-packages (4.59.0)
```

1.2. Load Dependencies

Import required packages

goto toc

In [123…

```
# Import libraries necessary for this project
import numpy as np
import pandas as pd
import scipy.stats as stats
import math
from tqdm import tqdm
import matplotlib.pyplot as plt

# Pretty display for notebooks
%matplotlib inline

import seaborn as sns

# Set default setting of seaborn
sns.set()
```

In [124…

```
# Import required function for feature selection
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif

# Import the required function for normalization
from sklearn.preprocessing import StandardScaler

# Import train and test split function
from sklearn.model_selection import train_test_split
```

In [125…

```
# Import Classifiers to be used

# Import Grid Search Cross Validation for tunning
from sklearn.model_selection import GridSearchCV

# Import KNN classifier
from sklearn.neighbors import KNeighborsClassifier
```

```python
# Import Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier

# Import Naive bayes classifier
from sklearn.naive_bayes import GaussianNB

# Import KNN classifier
from sklearn.neighbors import KNeighborsClassifier

# Import Gradient Boosting Classifier
from sklearn.ensemble import GradientBoostingClassifier, GradientBoostingRegressor
```

In [126…

```python
# Import packages to calculate performance of the models
from sklearn import metrics

# Function to compute confusion metric
from sklearn.metrics import confusion_matrix

# Function to generate classification report
from sklearn.metrics import classification_report
```

In [127…

```python
# To save the model import pickle
import pickle
```

2. Load dataset

Read data from mobile_price.csv file using pandas method read_csv().

goto toc

In [6]:

```python
# read the data
raw_data = pd.read_csv('data/mobile_price.csv')

# print the first five rows of the data
raw_data.head()
```

Out[6]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_core
0	842	0	2.2	0	1	0	7	0.6	188	[illegible]
1	1021	1	0.5	1	0	1	53	0.7	136	[illegible]
2	563	1	0.5	1	2	1	41	0.9	145	[illegible]
3	615	1	2.5	0	0	0	10	0.8	131	[illegible]
4	1821	1	1.2	0	13	1	44	0.6	141	[illegible]

5 rows × 21 columns

3. Data Types and Dimensions

goto toc

In [7]:
```python
# check the data types of the features
raw_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
```

Note:

Features like **blue**, **dual_sim**, **four_g**, **n_cores**, **three_g**, **touch_screen**, **wifi** and **price_range** are actually categorical in nature but are represemted as numeric so we need to convert them for better analysis.

In [8]:
```python
# create copy of the dataframe
data = raw_data.copy()

# Create list of features to be converted into category
features = ['blue', 'dual_sim', 'four_g', 'n_cores', 'three_g', 'touch_screen', 'wif

# Convert numeric to categorical
for col in features:
    data[col] = pd.Categorical(data[col])

# Check for datatypes
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   category
 2   clock_speed    2000 non-null   float64 
 3   dual_sim       2000 non-null   category
 4   fc             2000 non-null   int64   
 5   four_g         2000 non-null   category
 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   category
```

```
 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   category
 18  touch_screen   2000 non-null   category
 19  wifi           2000 non-null   category
 20  price_range    2000 non-null   category
dtypes: category(8), float64(2), int64(11)
memory usage: 220.2 KB
```

In [9]:

```python
# Get categorical features
categorical_features = data.select_dtypes('category').columns.values.tolist()

# Get nuemric features
numerical_features = [col for col in data.columns.values if col not in categorical_f
```

In [11]:

```python
print("Mobile Price Classification Data Set has \033[4m\033[1m{}\033[0m\033[0m data
print(f"Numeric features: \033[4m\033[1m{len(numerical_features)}\033[0m\033[0m \nCa
```

```
Mobile Price Classification Data Set has 2000 data points with 21 variables each.
Numeric features: 13
Categorical features: 8
```

4. Data Preprocessing

Data preprocessing is a data mining technique which is used to transform the raw data in a useful and efficient format.

...goto toc

4.1. Data Cleaning

Data cleaning refers to preparing data for analysis by removing or modifying data that is incomplete, irrelevant, duplicated, or improperly formatted.

...goto toc

Missing Data Treatment

If the missing values are not handled properly we may end up drawing an inaccurate inference about the data. Due to improper handling, the result obtained will differ from the ones where the missing values are present.

In [12]:

```python
# get the count of missing values
missing_values = data.isnull().sum()

# print the count of missing values
print(missing_values)
```

```
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
```

Note: There are no missing values in the dataset so we can proceed further

---

Summary

Number of Instances	Number of Attributes	Numeric Features	Categorical Features	Missing Values
2000	21	13	8	Null

4.2. Exploratory Analysis

The preliminary analysis of data to discover relationships between measures in the data and to gain an insight on the trends, patterns, and relationships among various entities present in the data set with the help of statistics and visualization tools is called Exploratory Data Analysis (EDA).

Exploratory data analysis is cross-classified in two different ways where each method is either graphical or non-graphical. And then, each method is either univariate, bivariate or multivariate.

...goto toc

4.2.1. Numerical Features

Analysis of only numeric features

...goto toc

In [15]:

```
# Get only numeric features for analysis
numeric_data = data[numerical_features]
numeric_data.head()
```

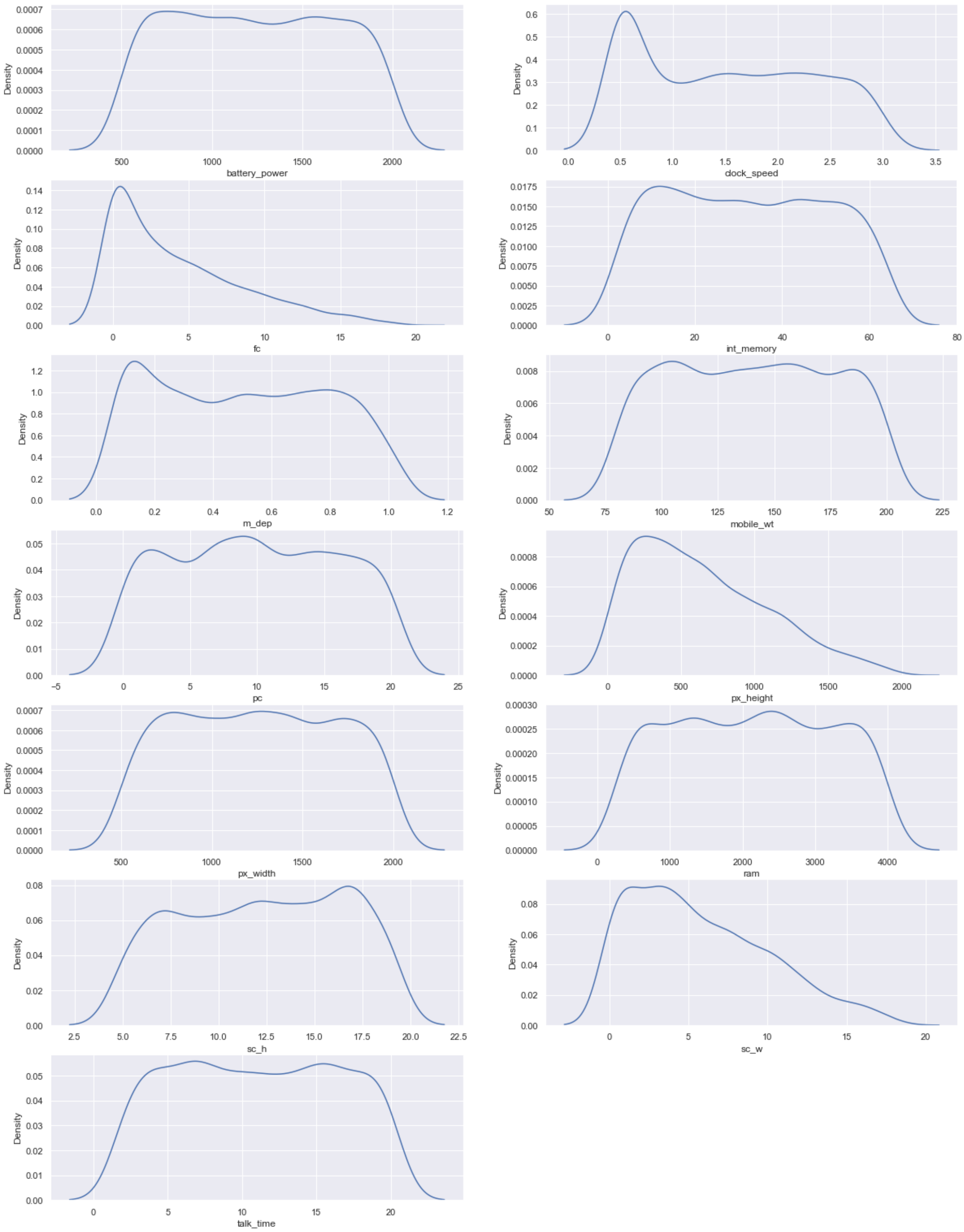
Out[15]:

	battery_power	clock_speed	fc	int_memory	m_dep	mobile_wt	pc	px_height	px_width	ram
0	842	2.2	1	7	0.6	188	2	20	756	2549

	battery_power	clock_speed	fc	int_memory	m_dep	mobile_wt	pc	px_height	px_width	ram
1	1021	0.5	0	53	0.7	136	6	905	1988	2631
2	563	0.5	2	41	0.9	145	6	1263	1716	2603
3	615	2.5	0	10	0.8	131	9	1216	1786	2769
4	1821	1.2	13	44	0.6	141	14	1208	1212	1411

In [87]:

```
# PLot KDE for all features

# Set size of the figure
plt.figure(figsize=(20,35))

# Iterate on list of features
for i, col in enumerate(numerical_features):
    if numeric_data[col].dtype != 'object':
        ax = plt.subplot(9, 2, i+1)
        kde = sns.kdeplot(numeric_data[col], ax=ax)
        plt.xlabel(col)

# Save the plot
plt.savefig("Numeric_Features_1.png")

# Show plots
plt.show()
```



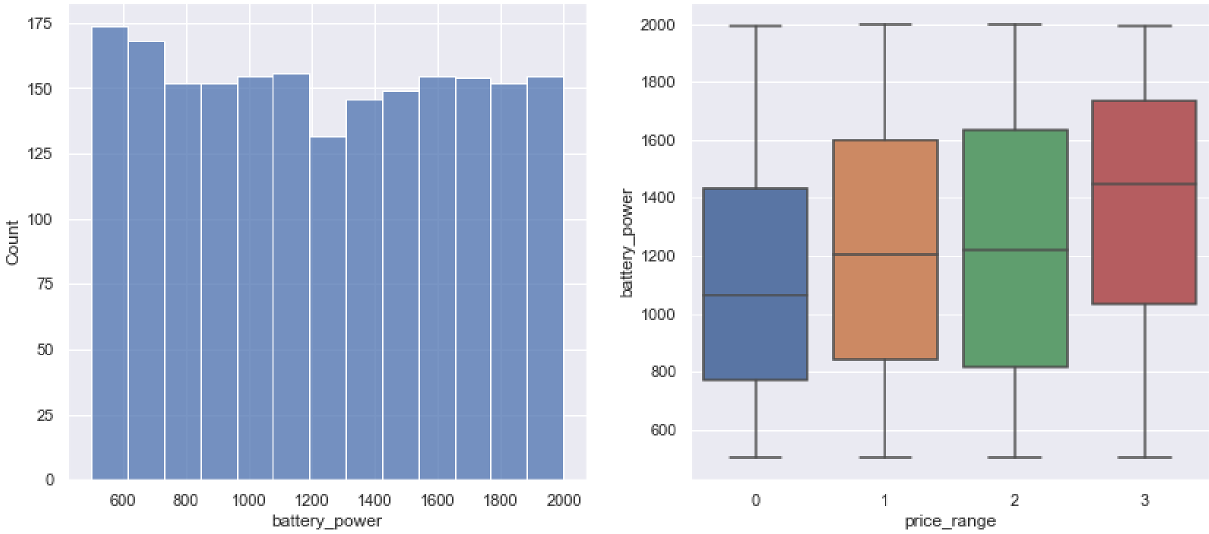
In [94]:

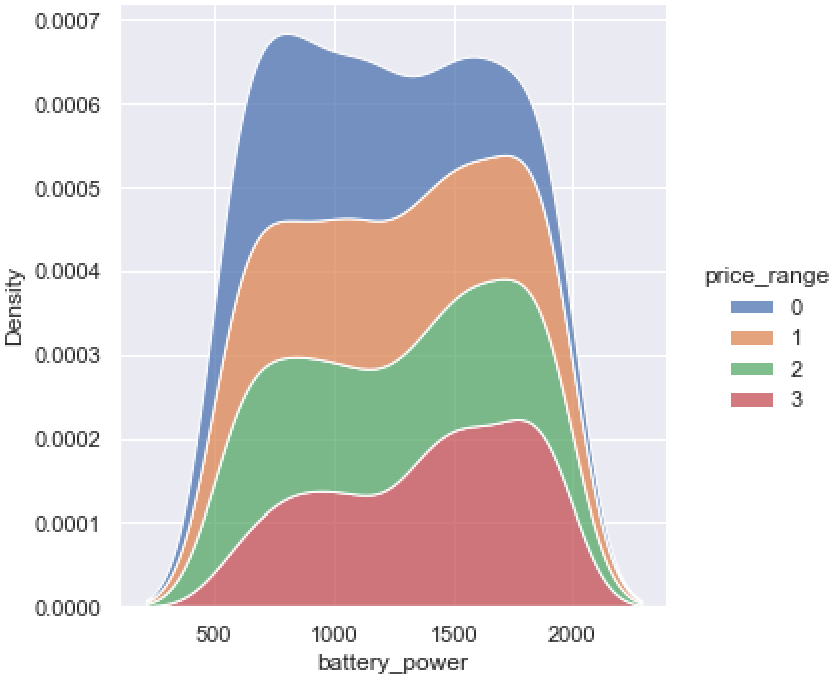
```
# Function to plot a numeric feature
def plot_numeric_feature(data, numerical_features):
    # Iterate throw each feature
    for feature in numerical_features:
        print("-"*150)
        print(f"Feature : \033[4m\033[1m{feature}\033[0m\033[0m")
        print("-"*150)

        # Create subplots figure
        fig, axes = plt.subplots(1, 2, figsize=(14, 6))

        # Plot histogram
        sns.histplot(data=data, x=feature, ax = axes[0])
```

```
        # Boxplot
        sns.boxplot(y = feature , x = 'price_range', data = data, ax = axes[1] )

        # Displot of given feature with respect to output variable
        sns.displot(data=data, x=feature, hue="price_range", multiple="stack", kind=

        # Save the plot
        fig.savefig(f"{feature}_Feature.png")


        # Show all plots
        plt.show()
```

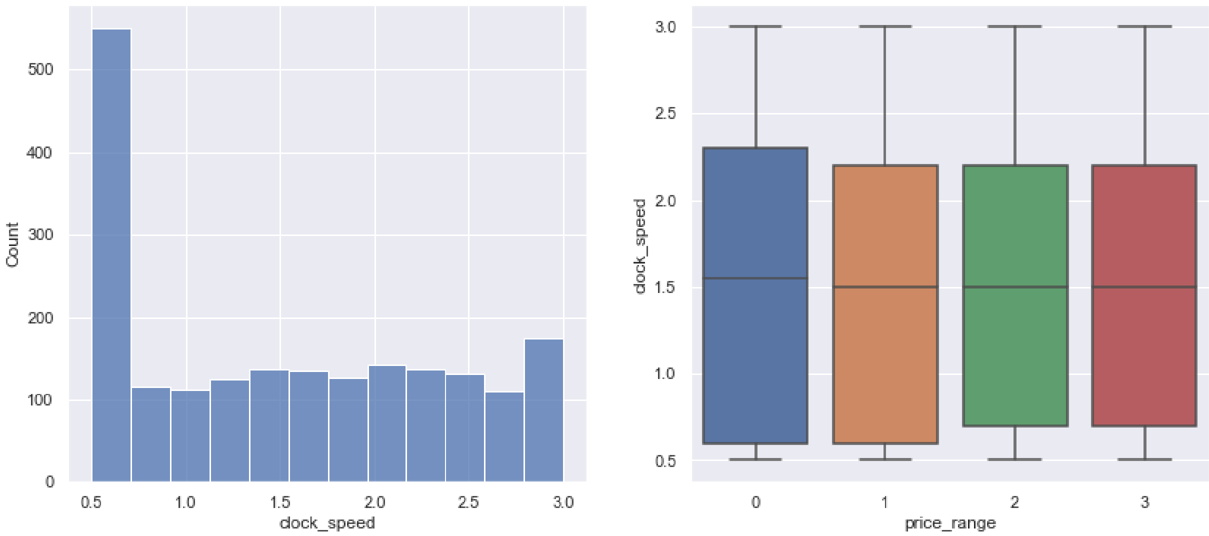
In [95]:
```
plot_numeric_feature(data, numerical_features)
```

```
-----------------------------------------------------------------------------------
------------------------------------------------------------------
Feature : battery_power
-----------------------------------------------------------------------------------
------------------------------------------------------------------
```

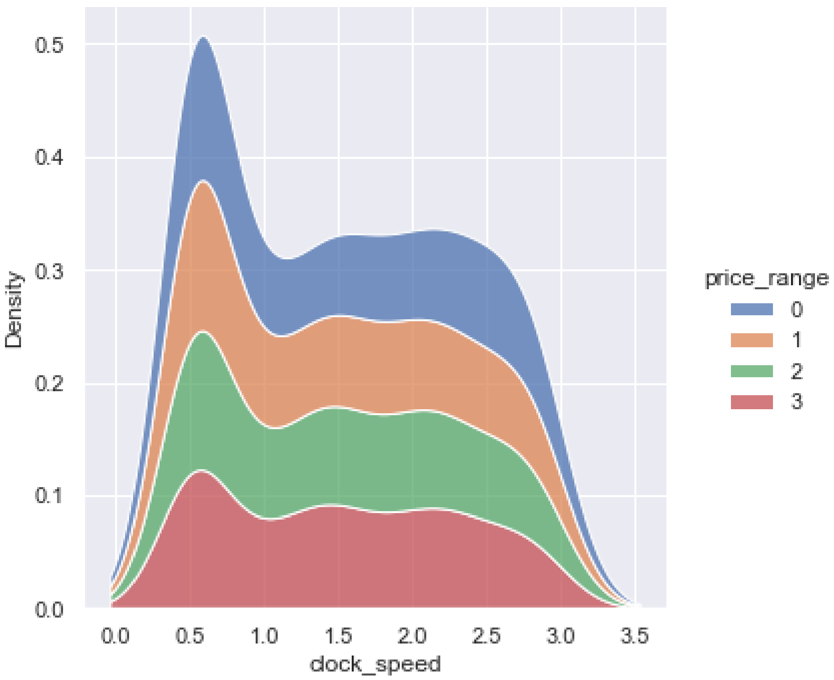





```
-----------------------------------------------------------------------------------
------------------------------------------------------------------
Feature : clock_speed
-----------------------------------------------------------------------------------
------------------------------------------------------------------
```

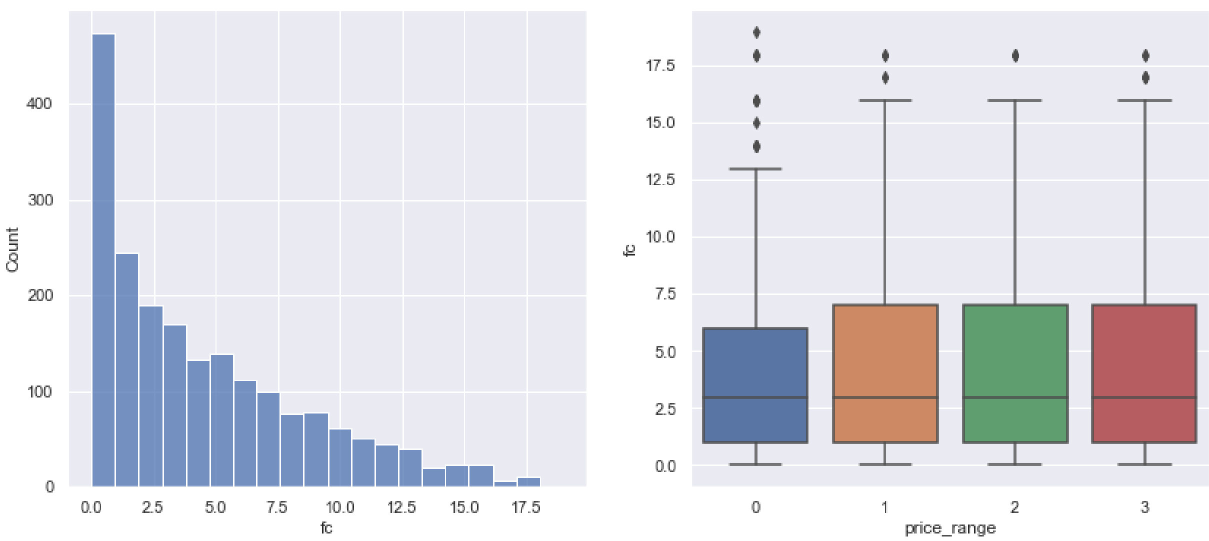


Count
clock_speed
clock_speed
price_range

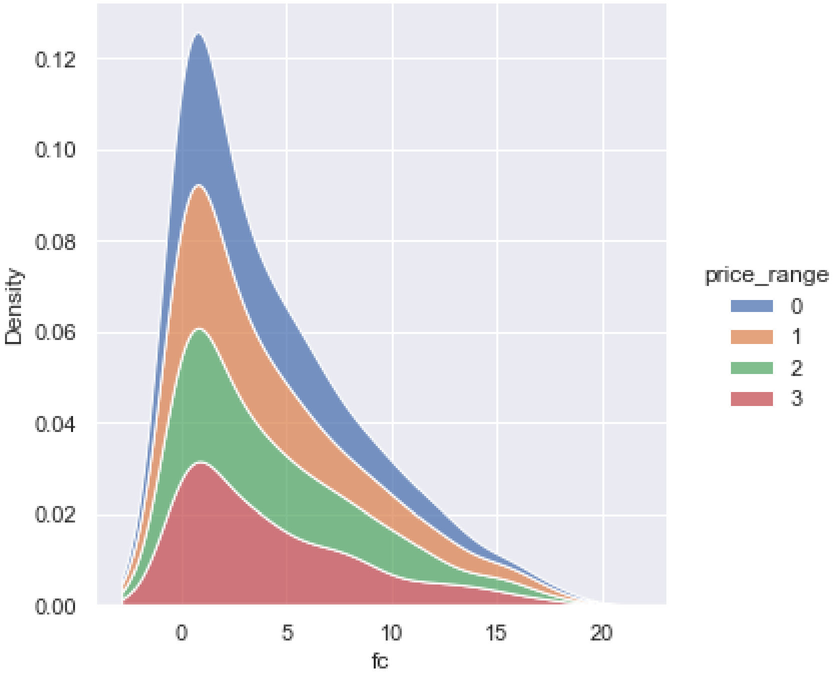


Density
clock_speed
price_range
0
1
2
3

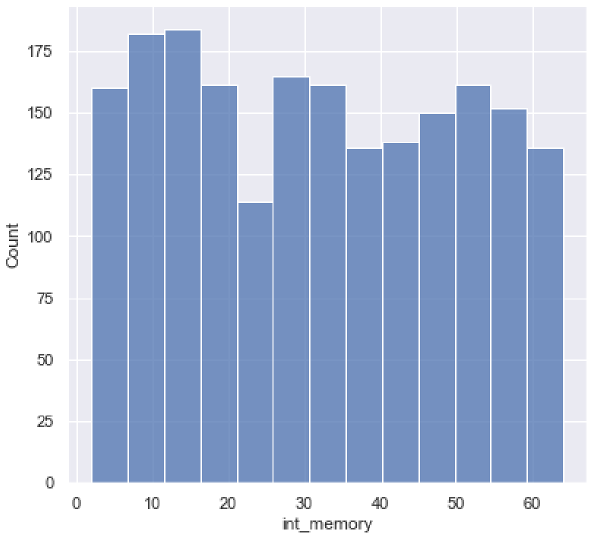

```
---------------------------------------------------------------------------------
----------------------------------------------------------------
Feature : fc
---------------------------------------------------------------------------------
----------------------------------------------------------------
```

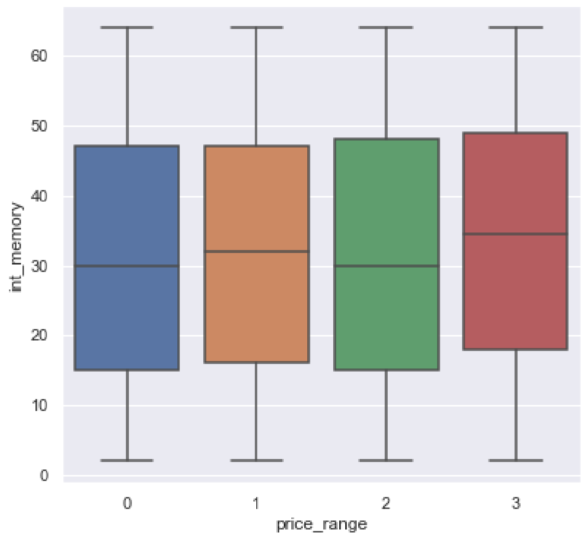


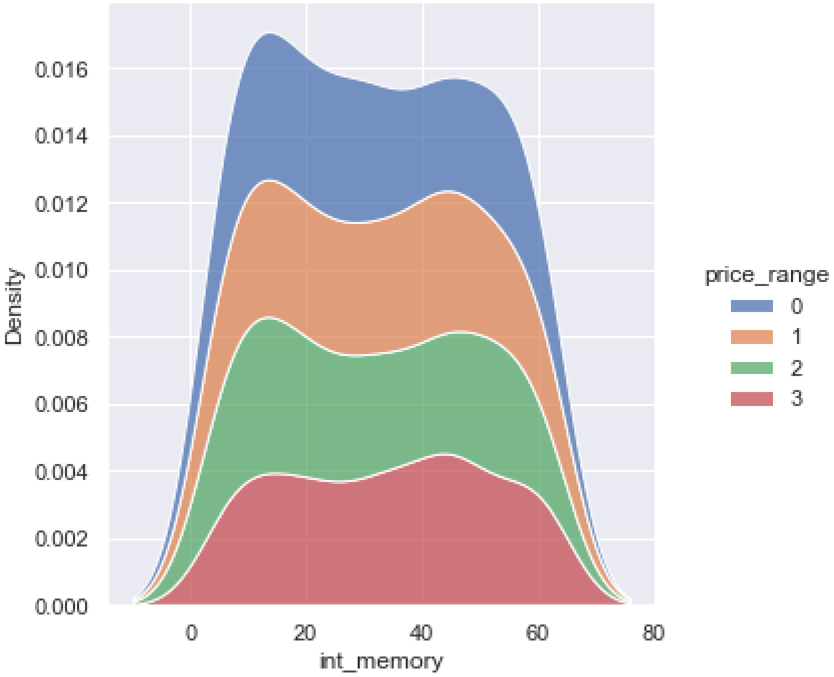
Count
fc
fc
price_range



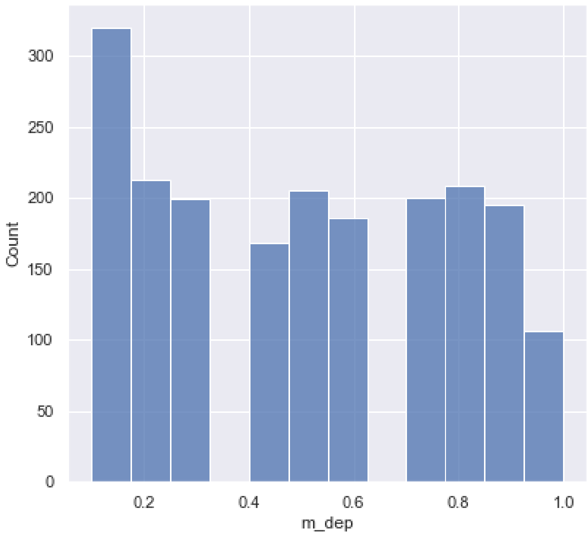
Feature : **int_memory**

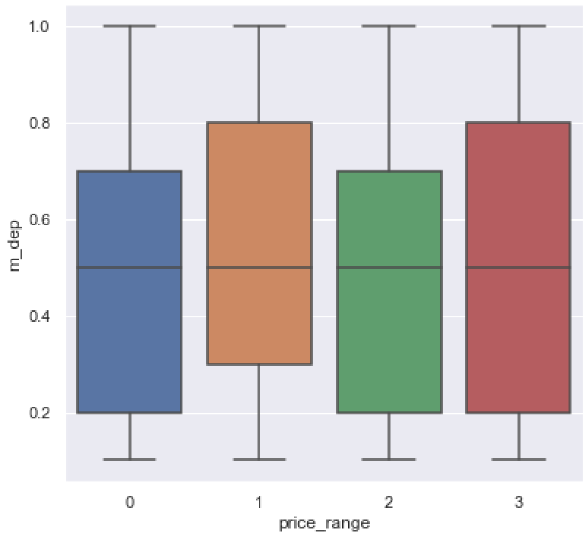


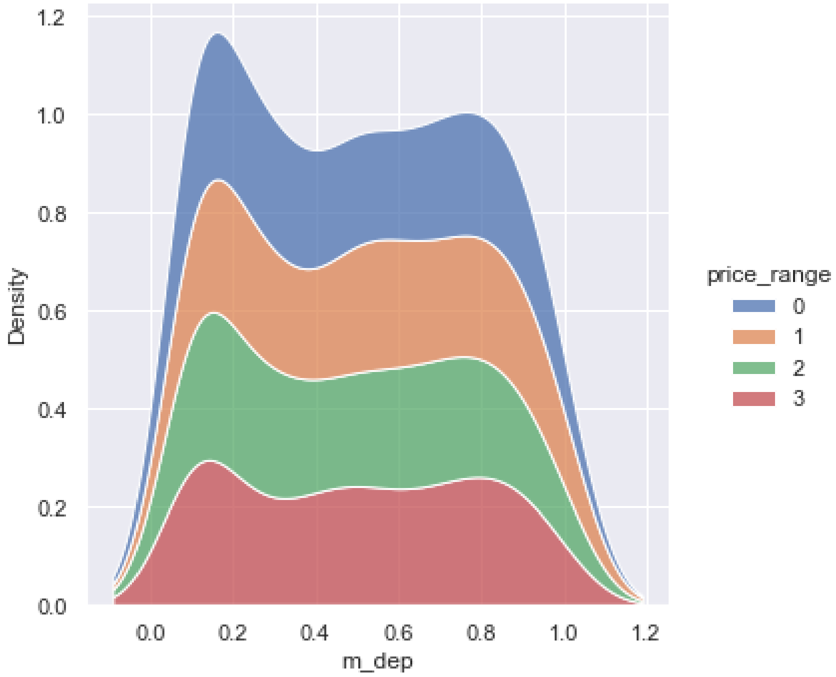




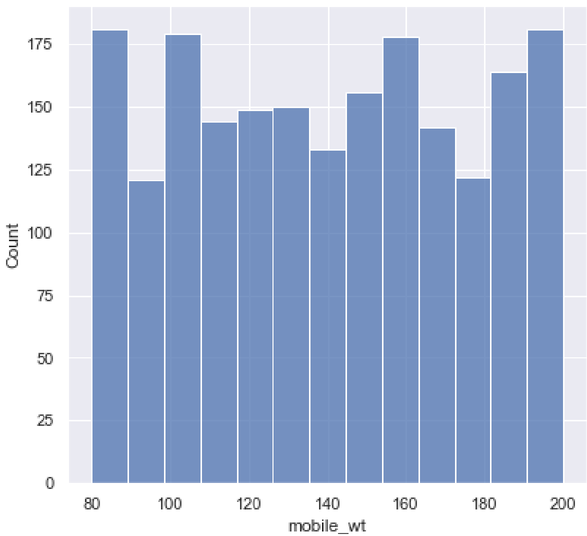
Feature : **m_dep**

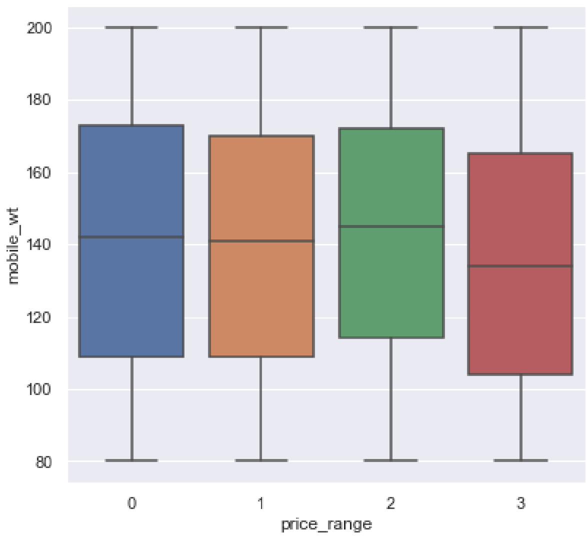


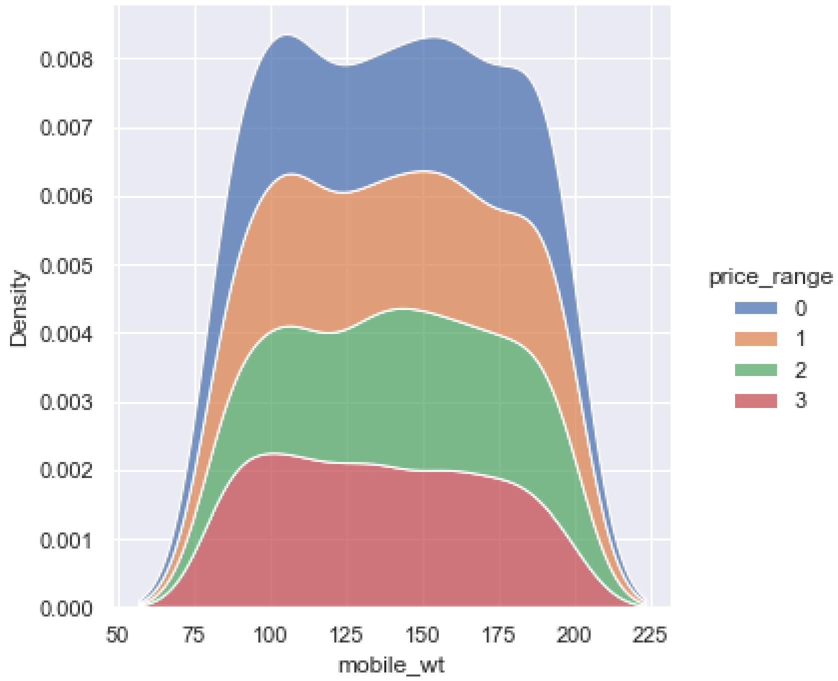




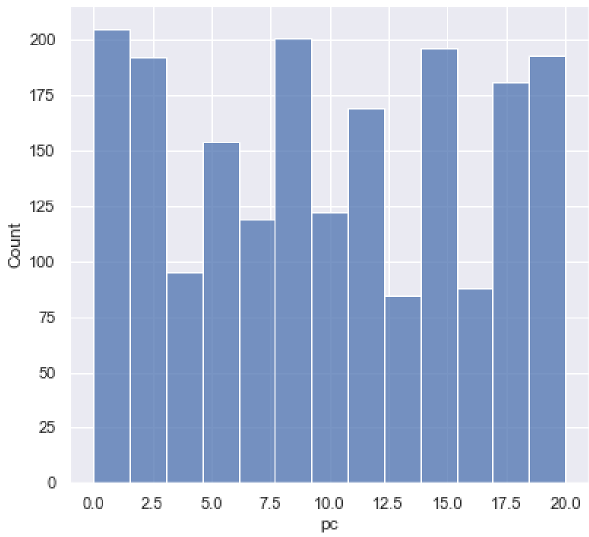

```
--------------------------------------------------------------------------------------------------------------------------------------------------------------
Feature : mobile_wt
--------------------------------------------------------------------------------------------------------------------------------------------------------------
```

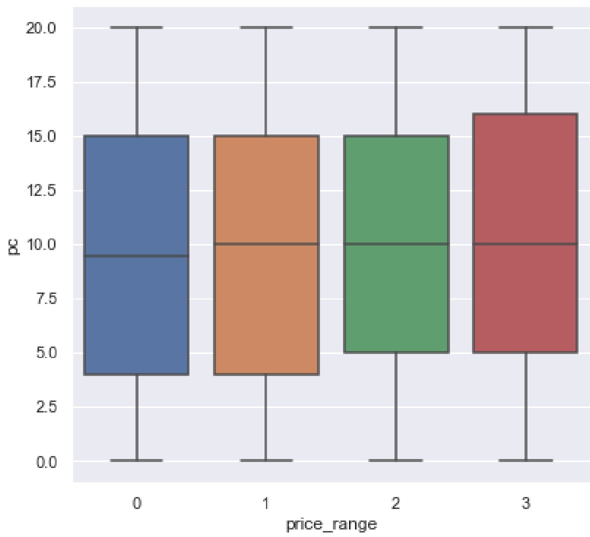




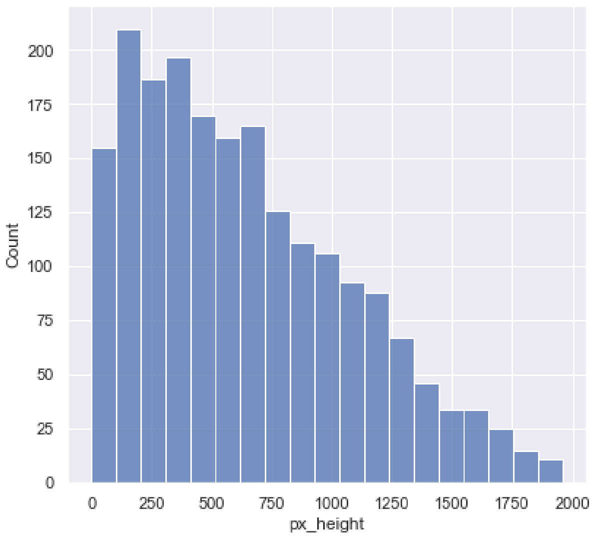

0.008
0.007
0.006
0.005
0.004
0.003
0.002
0.001
0.000
Density
50
75
100
125
150
175
200
225
mobile_wt
price_range
0
1
2
3

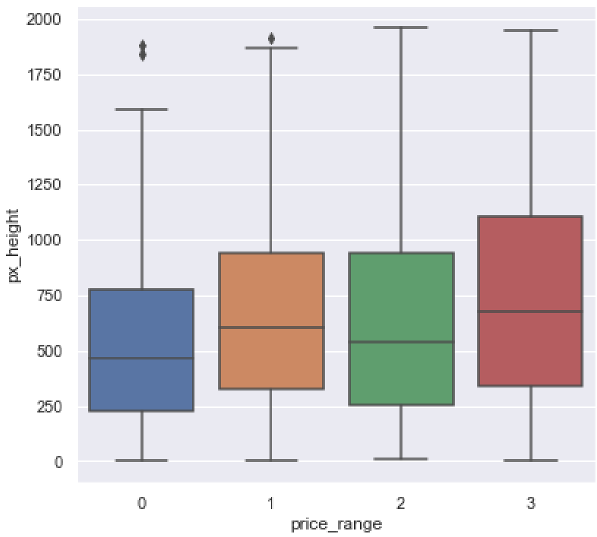
```
------------------------------------------------------------------------------------------------------------------------------------------------------------------------
Feature : pc
------------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

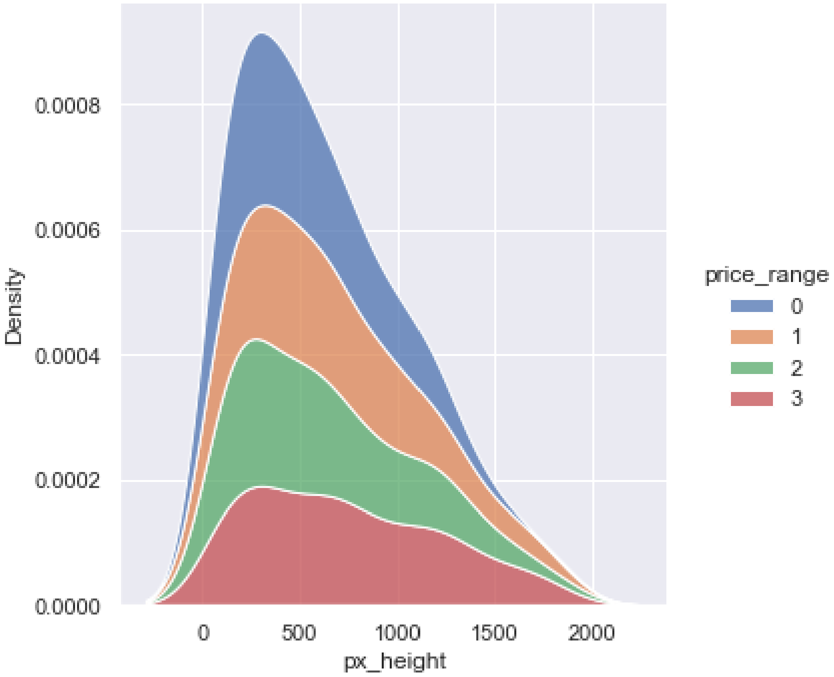

200
175
150
125
100
75
50
25
0
Count
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
pc


20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
pc
0
1
2
3
price_range

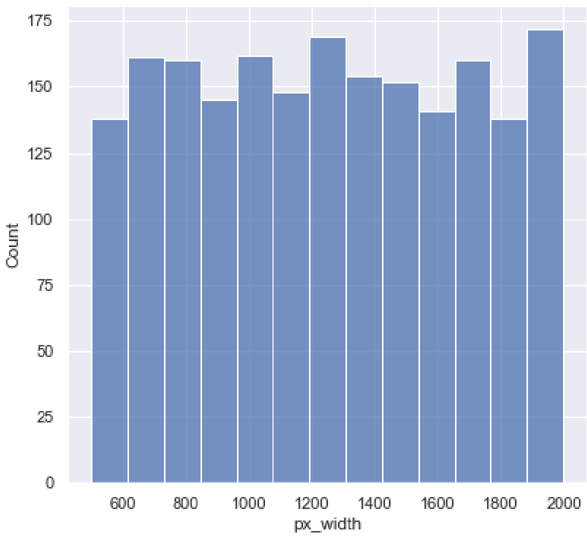
Feature : **px_height**

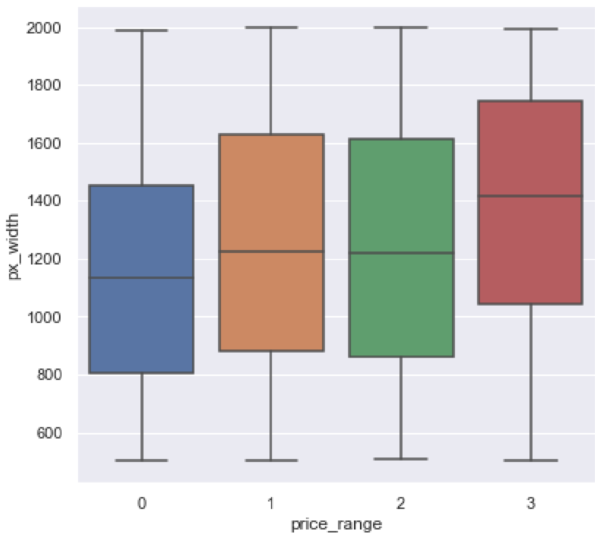


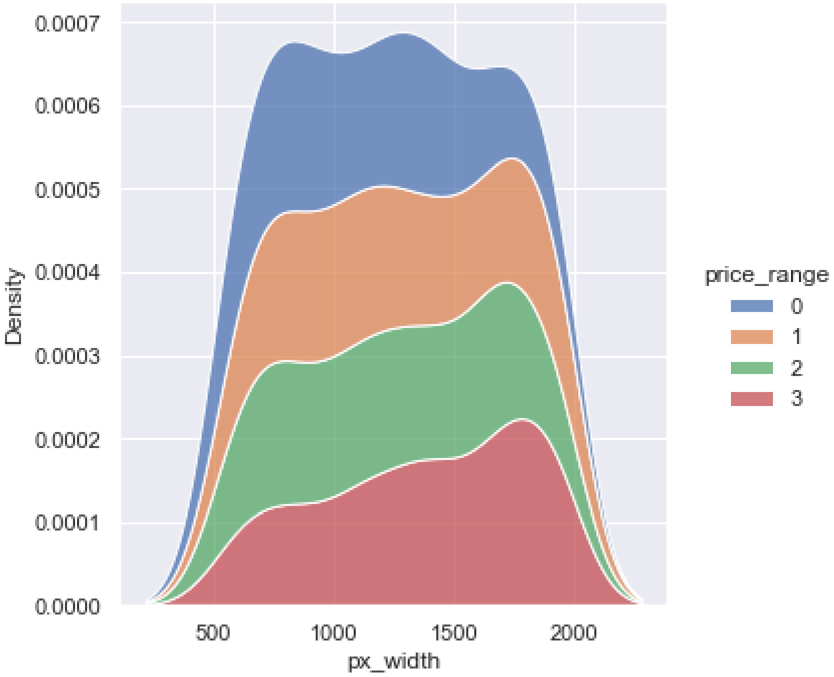




Feature : **px_width**

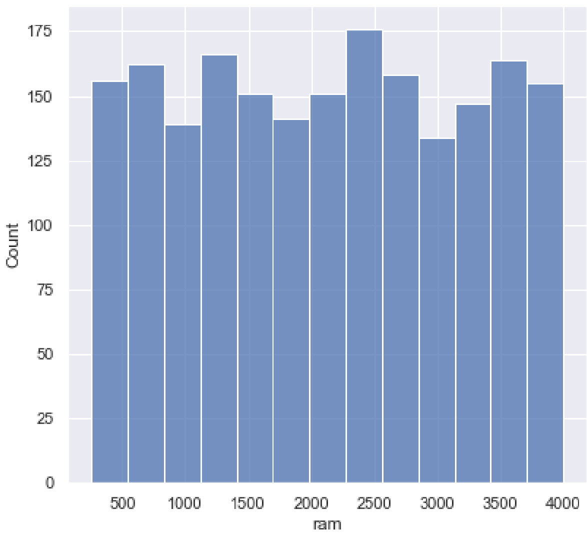




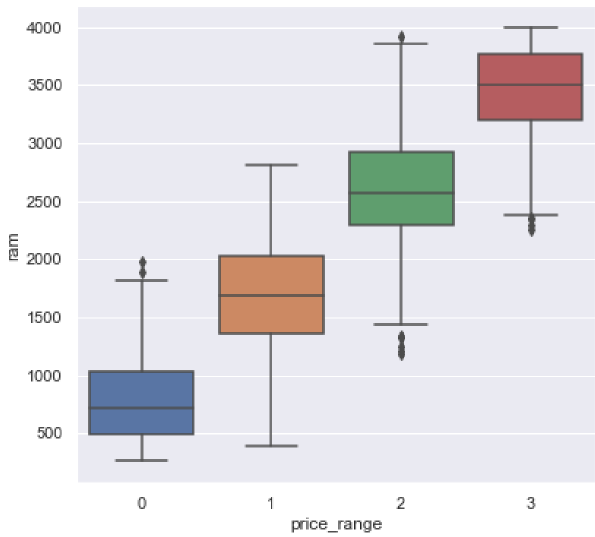


0.0007
0.0006
0.0005
0.0004
0.0003
0.0002
0.0001
0.0000
Density
500
1000
1500
2000
px_width
price_range
0
1
2
3

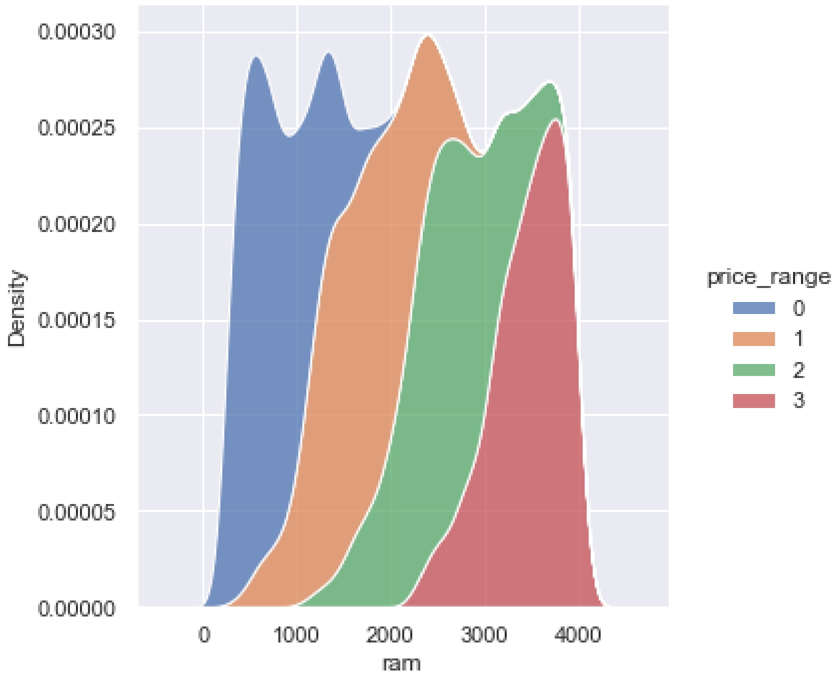

```
--------------------------------------------------------------------------------------------------------------------
Feature : ram
--------------------------------------------------------------------------------------------------------------------
```



175
150
125
100
75
50
25
0
Count
500
1000
1500
2000
2500
3000
3500
4000
ram

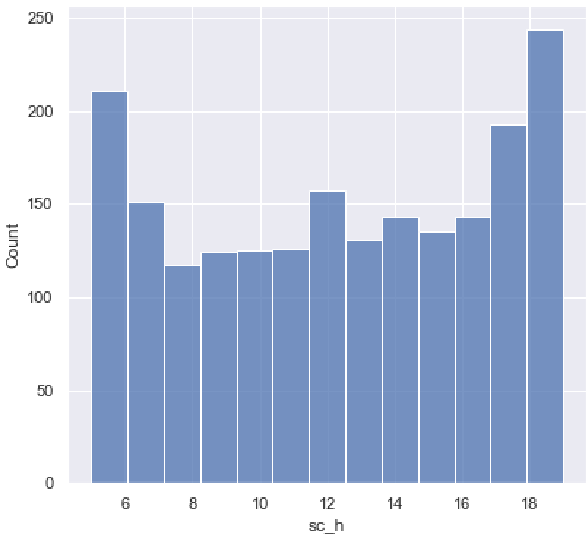


4000
3500
3000
2500
2000
1500
1000
500
ram
0
1
2
3
price_range

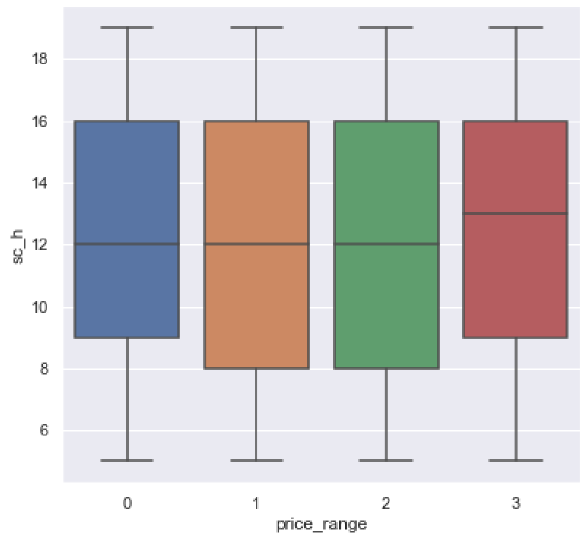


0.00030
0.00025
0.00020
0.00015
0.00010
0.00005
0.00000
Density
0
1000
2000
3000
4000
ram
price_range
0
1
2
3

Feature : **sc_h**



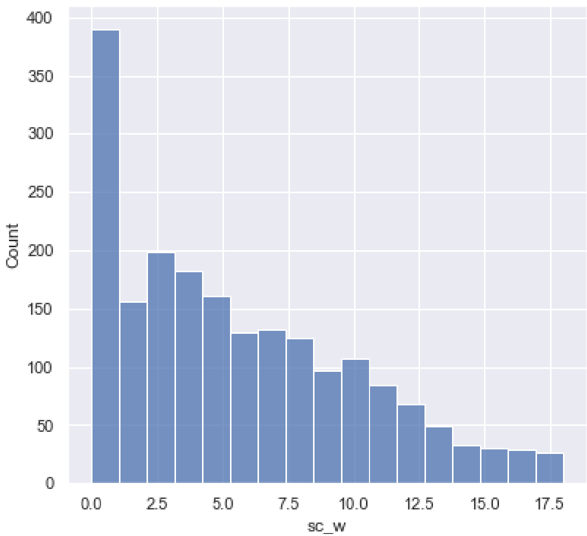
250
200
150
100
50
0
Count
6
8
10
12
14
16
18
sc_h

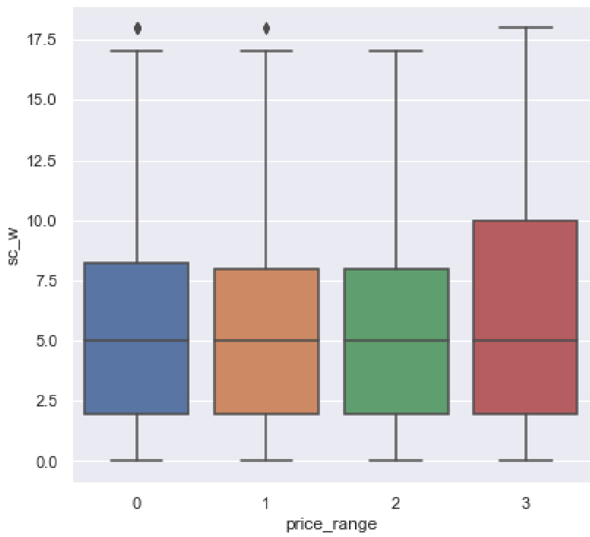


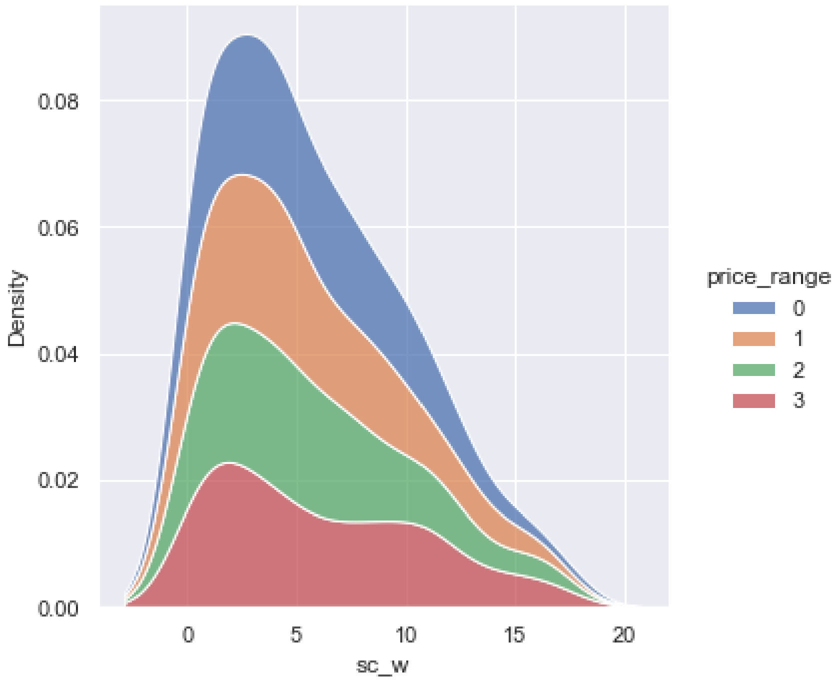
18
16
14
12
10
8
6
sc_h
0
1
2
3
price_range


Density
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07
5
10
15
20
sc_h
price_range
0
1
2
3

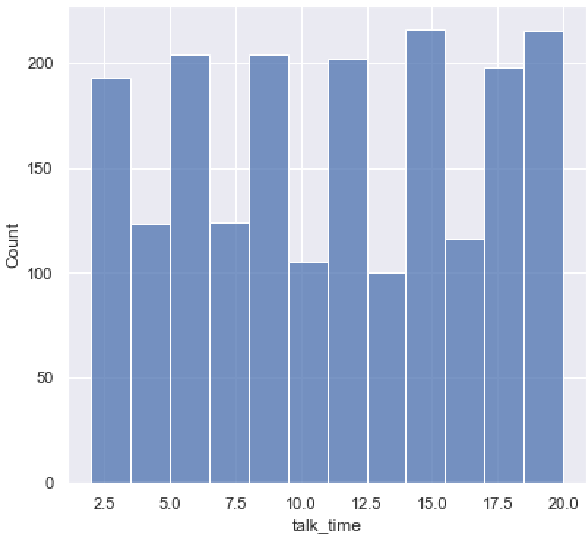
Feature : **sc_w**

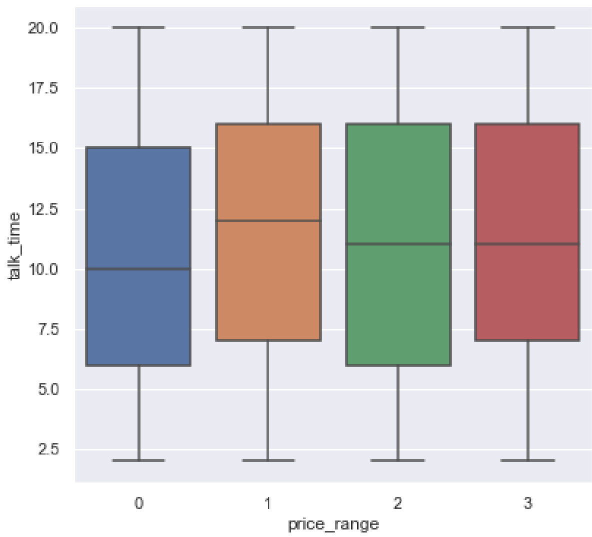

Count
0
50
100
150
200
250
300
350
400
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
sc_w

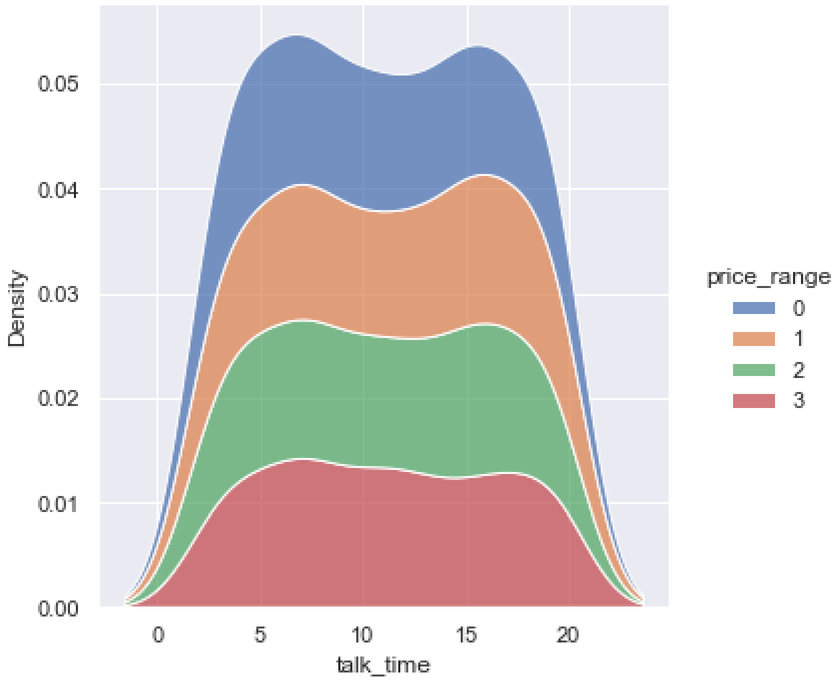

sc_w
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0
1
2
3
price_range



Feature : **talk_time**







Note:

- **battery_power**, **int_memory**, **m_depth**, **mobile_wt**, **pc**, **px_width**, **ram**, **sc_h** and **talktime** are features with data concentrated toward the center and their extremes are less in quantity.
- **sc_w**, **px_h**, **fc** and **clock_speed** features are right skewed i.e mean > median > mode

Correlation

In [19]:

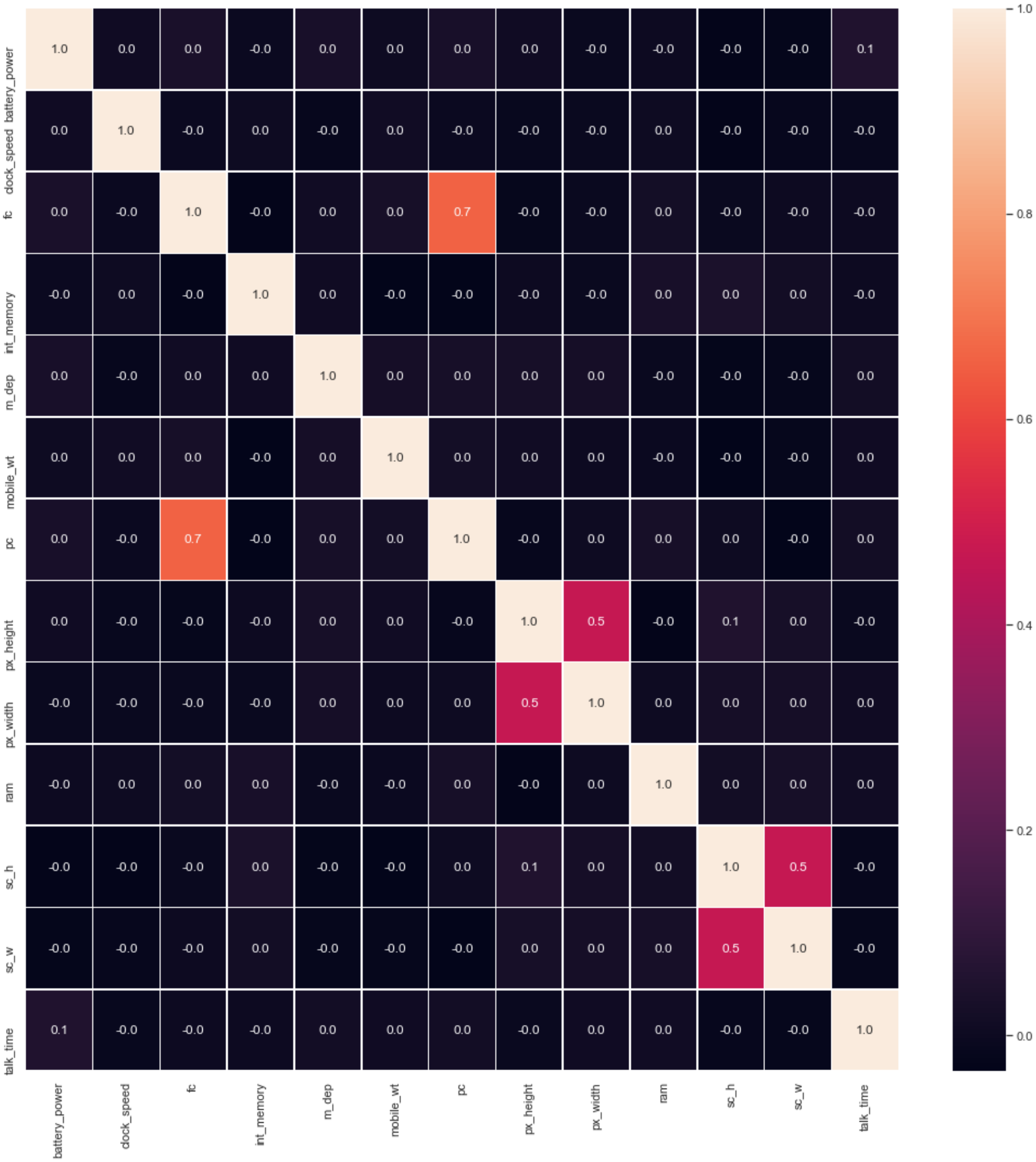
```
# check correlation
corr = data.corr(method = 'spearman')
corr
```

Out[19]:

	battery_power	clock_speed	fc	int_memory	m_dep	mobile_wt	pc
battery_power	1.000000	0.009161	0.034931	-0.003748	0.033412	0.001752	0.030757
clock_speed	0.009161	1.000000	-0.005288	0.005447	-0.014712	0.010773	-0.005925
fc	0.034931	-0.005288	1.000000	-0.027282	0.012780	0.027134	0.659161
int_memory	-0.003748	0.005447	-0.027282	1.000000	0.007380	-0.034259	-0.033373
m_dep	0.033412	-0.014712	0.012780	0.007380	1.000000	0.022438	0.027605
mobile_wt	0.001752	0.010773	0.027134	-0.034259	0.022438	1.000000	0.019011
pc	0.030757	-0.005925	0.659161	-0.033373	0.027605	0.019011	1.000000
px_height	0.009490	-0.013043	-0.020919	-0.001568	0.026156	0.011230	-0.015187
px_width	-0.009040	-0.008619	-0.009170	-0.008511	0.023180	0.000783	0.003462
ram	-0.001285	0.004119	0.019897	0.033061	-0.010398	-0.002731	0.028860
sc_h	-0.029283	-0.030092	-0.009578	0.040244	-0.023964	-0.033955	0.005105
sc_w	-0.026544	-0.015129	-0.001169	0.015987	-0.019489	-0.018952	-0.034842
talk_time	0.052730	-0.012699	-0.001404	-0.002436	0.016665	0.006343	0.014256

In [93]:

```python
# correlation map
f,ax = plt.subplots(figsize=(18, 18))
sns_plot = sns.heatmap(corr, annot=True, linewidths=.5, fmt= '.1f', ax=ax)

# Save the plot
f.savefig("correlation.png")
```



Note: Features **fc**, **pc** are highly correlated

4.4.2. Categorical Features

Analysis of categorical features

...goto toc

In [21]:

```python
# Get only categorical features for analysis
categorical_data = data[categorical_features]
```

```
categorical_data.head()
```

Out[21]:

	blue	dual_sim	four_g	n_cores	three_g	touch_screen	wifi	price_range
0	0	0	0	2	0	0	1	1
1	1	1	1	3	1	1	0	2
2	1	1	1	5	1	1	0	2
3	1	0	0	6	1	0	0	2
4	1	0	1	2	1	1	0	1

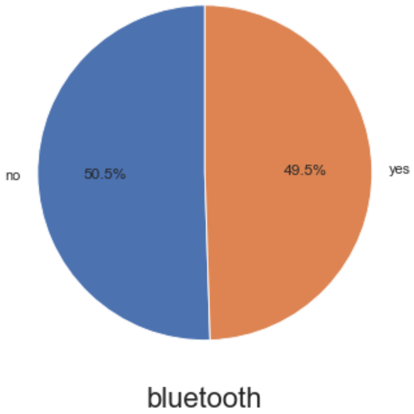
In [96]:

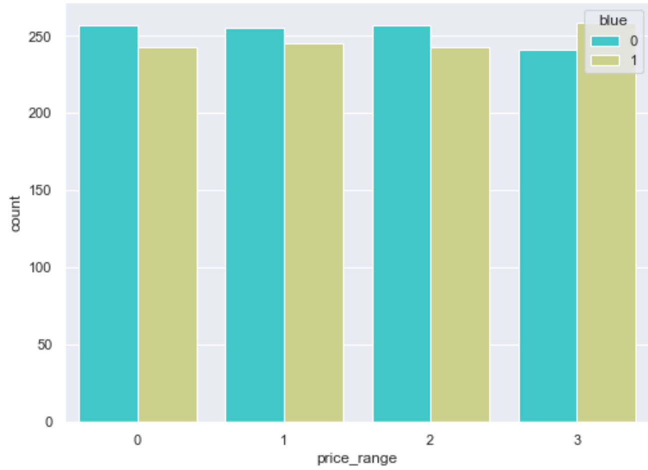
```
def plot_categorical_features(data, categorical_features):

    names, count  = {'blue': 'bluetooth', 'dual_sim': "Dual Sim", "four_g":"4G", "n_
         "three_g":"3G", "touch_screen":"Touch Screen", "wifi":"WiFi"}, 1

    for feature in categorical_features:

        if feature == "price_range":
            continue

        print("-"*150)
        print(f"Feature : \033[4m\033[1m{names[feature]}\033[0m\033[0m")
        print("-"*150)

        labels = ["no", "yes"] if feature != "n_cores" else data[feature].unique().t

        # Create subplots figure
        fig, axes = plt.subplots(1, 2, figsize=(18, 6))

        if count % 2 == 0:
            # Plot countplot of feature with respect to target
            sns.countplot(x = 'price_range', data = data, hue=feature, ax = axes[0],

            # Plot pie chart to show distribution of feature
            axes[1].pie(data[feature].value_counts().values, labels = labels, autopc
            axes[1].set_xlabel(names[feature], size=22)

        else:
            # Plot pie chart to show distribution of feature
            axes[0].pie(data[feature].value_counts().values, labels = labels, autopc
            axes[0].set_xlabel(names[feature], size=22)

            # Plot countplot of feature with respect to target
            sns.countplot(x = 'price_range', data = data, hue=feature, ax = axes[1],

        # Increase the counter
        count += 1

        # Save features
        fig.savefig(f"{feature}_feature.png")

        # Show all plots
        plt.show()
```

In [97]:

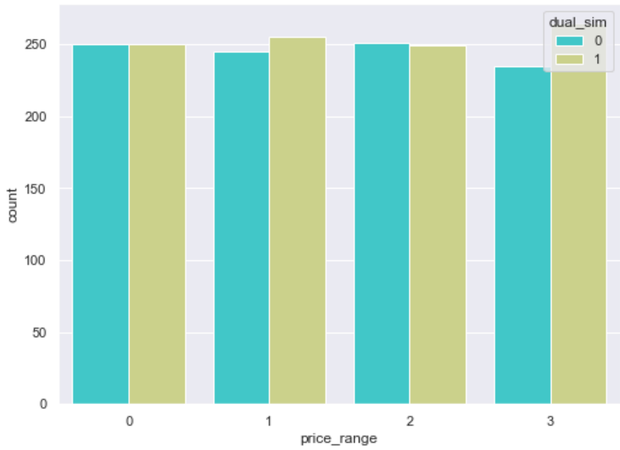
```
plot_categorical_features(categorical_data, categorical_features)
```

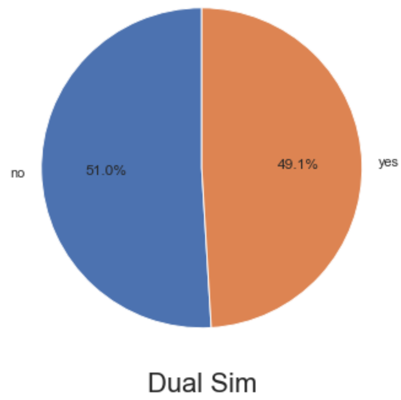

Feature : **bluetooth**



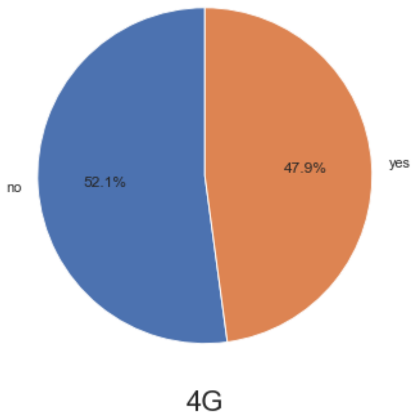


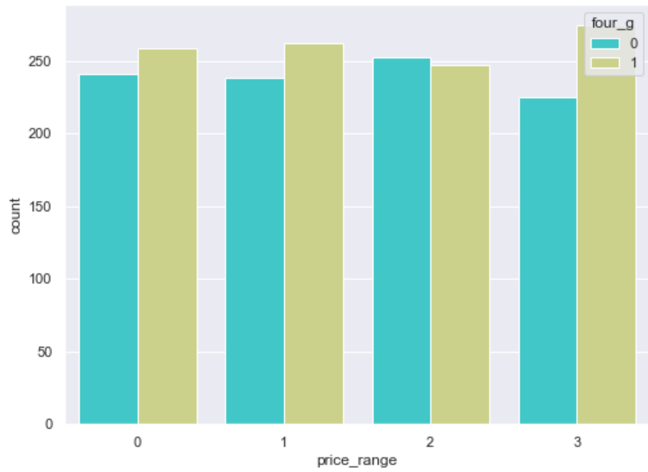
Feature : **Dual Sim**



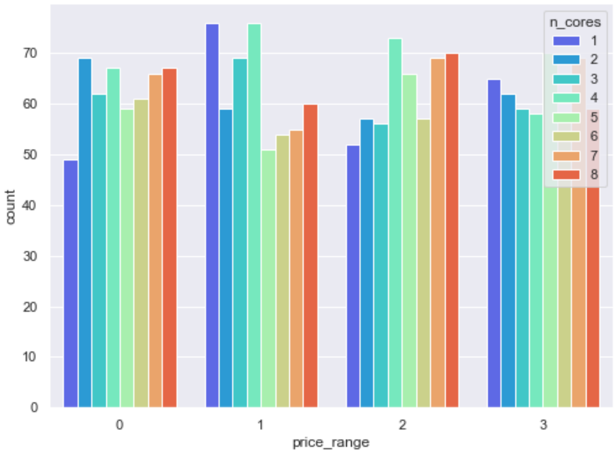


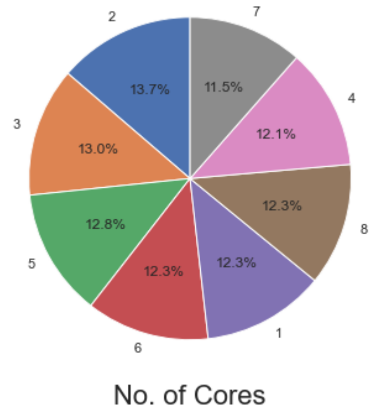
Feature : **4G**



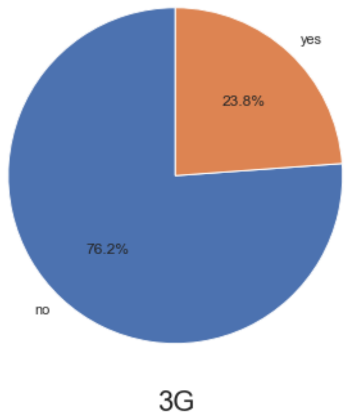


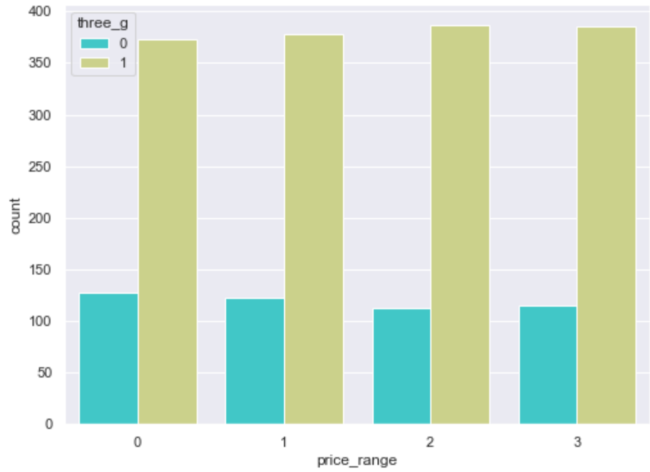
Feature : **No. of Cores**



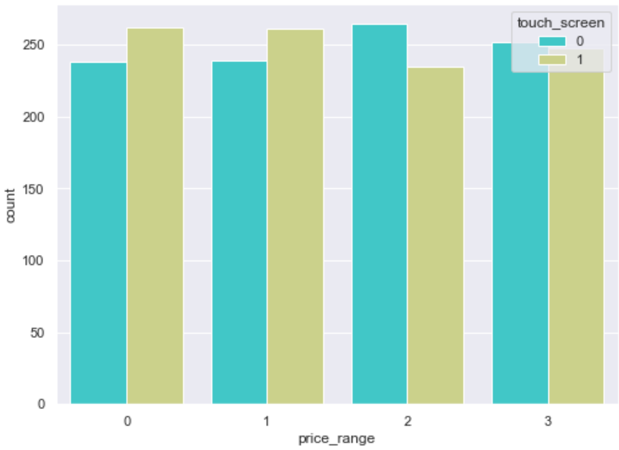


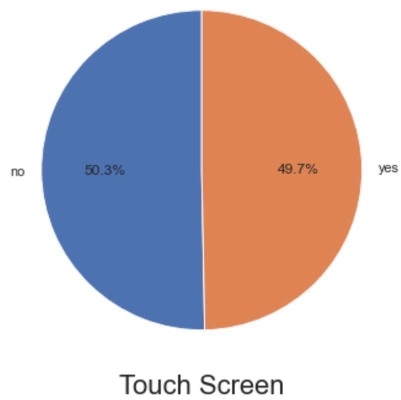
Feature : 3G



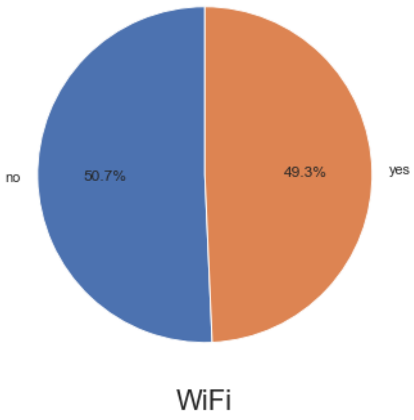


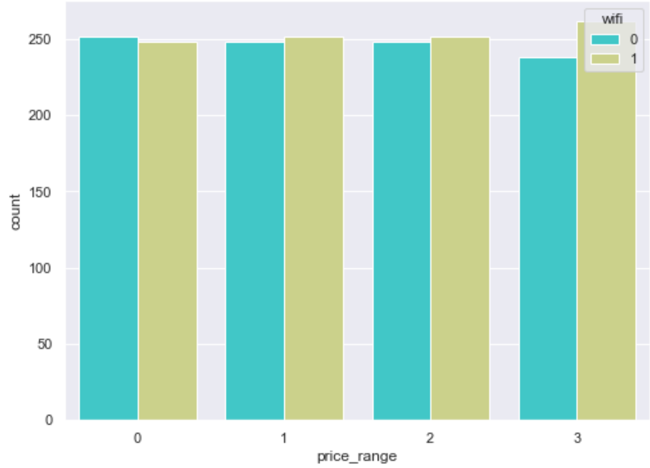
Feature : Touch Screen





Feature : WiFi

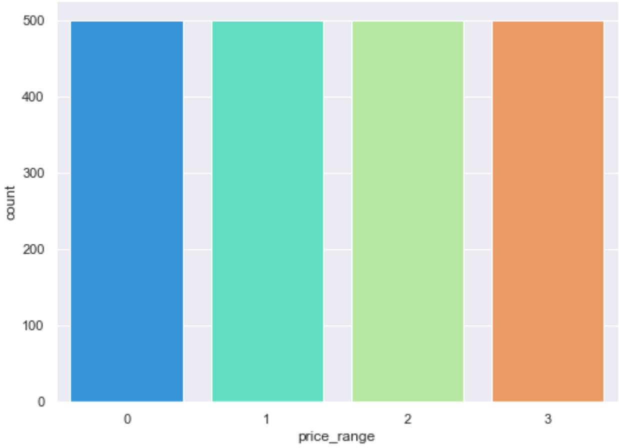


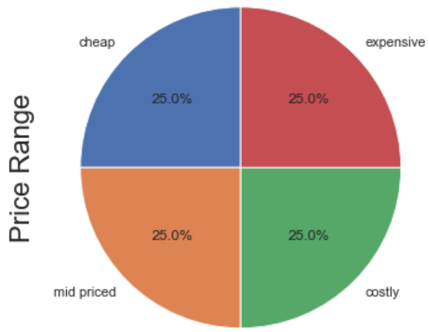


Analyzing target feature

In [120…

```
# Create subplots figure
fig, axes = plt.subplots(1, 2, figsize=(18, 6))

# Plot countplot of feature with respect to target
sns.countplot(x = 'price_range', data = data, ax = axes[0], palette='rainbow')

# Plot pie chart to show distribution of feature

labels = ['cheap', 'mid priced', 'costly', 'expensive']

axes[1].pie(categorical_data.price_range.value_counts().values, labels = labels, aut
axes[1].set_ylabel('Price Range', size=22)

# Save plot
fig.savefig('target_feature.png')

# Show all plots
plt.show()
```





4.2.3. Analysis Report

...goto toc

Analysis Report

Number of Instances	Number of Attributes	Numeric Features	Categorical Features	Target Feature	Missing Values
2000	21	13	8	price_range	Null

Data Types

Sr.No.	Column	Data type
1	battery_power	int64
2	blue	category
3	clock_speed	float64
4	dual_sim	category
5	fc	int64
6	four_g	category
7	int_memory	int64
8	m_dep	float64
9	mobile_wt	int64
10	n_cores	category
11	pc	int64
12	px_height	int64
13	px_width	int64
14	ram	int64
15	sc_h	int64
16	sc_w	int64
17	talk_time	int64
18	three_g	category
19	touch_screen	category
20	wifi	category
21	price_range	category

Exploratory Data Analysis

Numeric Features

- Generally cheaper phones have low front camera mega pixels as compared to others.
- Costly and expensive phones generally have higher battery and internal memory.
- Additionaly, they are lighter as compared to others.
- Pixel resolution increases as price range increases but clock speed doesnot show much deviation with respect to price.
- There is no significant variation in phones price with respect to screen width and height.
- front camera and primary camera mega pixels are higly correlated to each other.

- **RAM** is the most important feature to predict price among all the features which is also true in practical scenario.

Categorical Features

- Features like **blue**, **dual_sim**, **four_g**, **three_g**, **touch_screen**, **wifi** are binary in nature and are equally distributed
  - 1 : Yes
  - 0 : No
- Number of cores are ranging from 1 to 8
- Majority of expensive phone have features like wifi, bluetooth, dual sim and 4G.
- 3G features is available in all types of phones. But in the given dataset it is biased that 70% of phones don't have 3G service.
- Target feature price_range has four different categories
  - 0 : cheap
  - 1 : mid-priced
  - 2 : costly
  - 3 : expensive
- Additionally target feature is balanced

Note:

There is no need to encode categorical features as they are encoded by defualt. We just need to change there datatype using pandas function called *pd.to_numeric()*. But feature **n_cores** need to **one-hot encoded** as it has eight different categories.

In [25]:

```
# One-hot encode n_cores feature
no_cores = pd.get_dummies(data['n_cores'], prefix='cores', drop_first=True)

# Convert features to numeric
data_preprocessed = data.drop('n_cores', axis = 1).apply(pd.to_numeric, axis = 1)

# Concatenate with original feature
data_preprocessed = data_preprocessed.join(no_cores)
data_preprocessed.head()
```

Out[25]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	pc
0	842.0	0.0	2.2	0.0	1.0	0.0	7.0	0.6	188.0	2.0
1	1021.0	1.0	0.5	1.0	0.0	1.0	53.0	0.7	136.0	6.0
2	563.0	1.0	0.5	1.0	2.0	1.0	41.0	0.9	145.0	6.0
3	615.0	1.0	2.5	0.0	0.0	0.0	10.0	0.8	131.0	9.0
4	1821.0	1.0	1.2	0.0	13.0	1.0	44.0	0.6	141.0	14.0

5 rows × 27 columns

4.3. Feature Selection

Since there are all together **27** independent features we will perform feature selection to eliminate curse of dimensionality.

We will be using **f_classif as feature selector** because our **features** are **quantitative** i.e numeric and **target** feature is **categorical**. f_classif is a feature selector that computes the **ANOVA F-value** between each feature and the target vector.

...goto toc

In [27]:

```
# Seperate independent features and target feature
X, y = data_preprocessed.drop(['price_range'], axis = 1), data_preprocessed['price_r
```

In [28]:

```
# Calculate f-score and p-value
f_statistic, p_values = f_classif(X, y)

# Create a dataframe to record score
d = pd.DataFrame()

# Save features
d['feature'] = X.columns

# record F-score
d['fscore'] = f_statistic

# record p-values
d['pvalue'] = p_values

# Sort based of f-score
d.sort_values(by = 'fscore', ascending=False)
```

Out[28]:

	feature	fscore	pvalue
12	ram	3520.110824	0.000000e+00
0	battery_power	31.598158	5.948688e-20
11	px_width	22.620882	2.116911e-14
10	px_height	19.484842	1.886085e-12
8	mobile_wt	3.594318	1.311739e-02
6	int_memory	2.922996	3.277694e-02
13	sc_h	2.225984	8.324991e-02
14	sc_w	1.671000	1.712146e-01
15	talk_time	1.628811	1.806686e-01
7	m_dep	1.500682	2.124595e-01
17	touch_screen	1.293302	2.750433e-01
22	cores_5	1.291585	2.756251e-01
21	cores_4	1.065282	3.626575e-01
5	four_g	1.059525	3.651552e-01
9	pc	0.825446	4.797489e-01
24	cores_7	0.784417	5.025467e-01

	feature	fscore	pvalue
4	fc	0.772182	5.095042e-01
20	cores_3	0.574106	6.320524e-01
25	cores_8	0.513034	6.733232e-01
19	cores_2	0.509003	6.760970e-01
2	clock_speed	0.493708	6.866752e-01
1	blue	0.476768	6.984831e-01
16	three_g	0.457320	7.121507e-01
3	dual_sim	0.428239	7.327869e-01
18	wifi	0.284940	8.363070e-01
23	cores_6	0.163473	9.209778e-01

Note:

We can see that **RAM** has the **highest F-score** and **minimum p-value** which is expected. From *22* features we will be selecting top *10* features based on their *f-score* using *SelectKBest* method.

In [29]:

```
# Perform feature selection we will select best 10 features
fvalue_Best = SelectKBest(f_classif, k = 10)

# Fit and transform feature selector on given dataset
X_best = fvalue_Best.fit_transform(X, y)
```

In [30]:

```
print(f'Original dataset have \033[4m\033[1m{X.shape[1]}\033[0m\033[0m features.\nAf
```

```
Original dataset have 26 features.
After feature selection dataset have 10 features.
```

In [31]:

```
# The list of your K best features
mask = fvalue_Best.get_support()

# Get list of selected features
selected_features = [feature for bool_val, feature in zip(mask, X.columns.values.tol

# print best features
print("Selected features are : ", selected_features)
```

```
Selected features are :  ['battery_power', 'int_memory', 'm_dep', 'mobile_wt', 'px_h
eight', 'px_width', 'ram', 'sc_h', 'sc_w', 'talk_time']
```

4.4. Data Transformation

...goto toc

4.4.1 Normalization

Normalization is used to scale the data of an attribute so that it falls in a smaller range, such as -1.0 to 1.0 or 0.0 to 1.0. It is generally useful for classification algorithms.

We will use *Standard Scaler* to perform normalization.

...goto toc

In [33]:

```
# Initilize scaler
scaler = StandardScaler()

# fit the scaler
scaler.fit(X_best)
```

Out[33]:

```
StandardScaler()
```

In [34]:

```
# Transform the dataset
X_normal = scaler.fit_transform(X_best)
```

4.4.2. Split dataset

We will be splitting the dataset into train and test set with **70-30** split

...goto toc

In [36]:

```
# let us now split the dataset into train & test
X_train, X_test, y_train, y_test = train_test_split(X_normal, y, test_size = 0.3, ra

# print the shape of 'x_train'
print("X_train : ",X_train.shape)

# print the shape of 'x_test'
print("X_test : ",X_test.shape)

# print the shape of 'y_train'
print("y_train : ",y_train.shape)

# print the shape of 'y_test'
print("y_test : ",y_test.shape)
```

```
X_train :  (1400, 10)
X_test :  (600, 10)
y_train :  (1400,)
y_test :  (600,)
```

5. Model Development

We will be training different classification model and choose the one with best performance

...goto toc

5.1. KNN

*To find optimal value of **K** we will be performing hyperparameter tuning using **Grid Search Cross Validation**.*

...goto toc

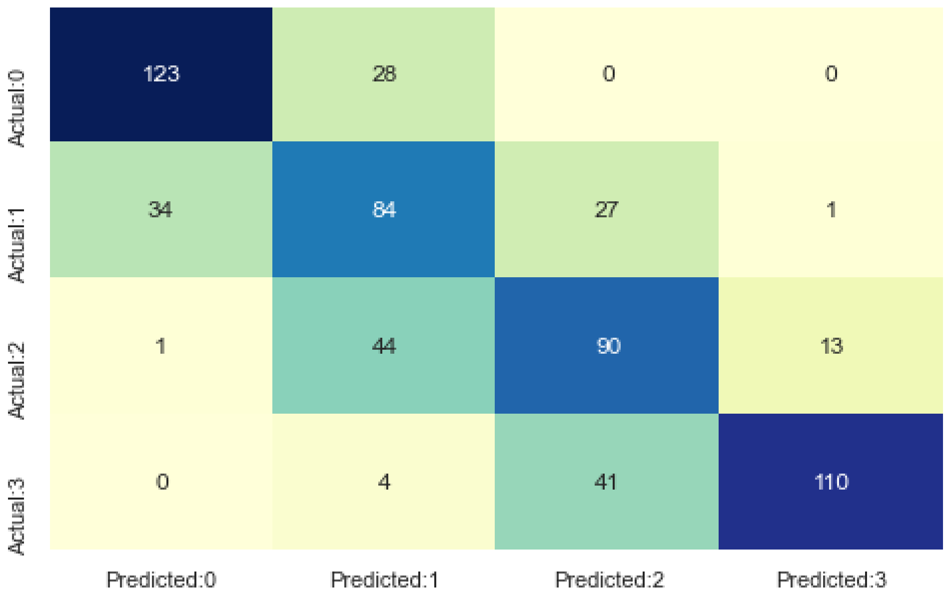
In [39]:
```
# Hyperparameter tuning

# Initialize a knn object
knn = KNeighborsClassifier()

# Create a dictionary of all values we want to test for n_neighbors
param_grid = {'n_neighbors': np.arange(2, 6)}
```

In [40]:
```
# Perform gridsearch
knn_gscv = GridSearchCV(knn, param_grid, cv=5)

# fit the data
knn_gscv.fit(X_train, y_train)
```

Out[40]:
```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(),
             param_grid={'n_neighbors': array([2, 3, 4, 5])})
```

In [41]:
```
# predict the values
y_pred_knn  = knn_gscv.predict(X_test)
```

In [42]:
```
# compute the confusion matrix
cm = confusion_matrix(y_test, y_pred_knn)

# label the confusion matrix
conf_matrix = pd.DataFrame(data=cm,columns=['Predicted:0','Predicted:1','Predicted:2

# set sizeof the plot
plt.figure(figsize = (8,5))

# plot a heatmap
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap="YlGnBu", cbar=False)
plt.show()
```



In [44]:
```
# Generate classiffication report

# accuracy measures by classification_report()
result = classification_report(y_test, y_pred_knn)

# print the result
print(result)
```

```
              precision    recall  f1-score   support

         0.0       0.78      0.81      0.80       151
         1.0       0.53      0.58      0.55       146
         2.0       0.57      0.61      0.59       148
         3.0       0.89      0.71      0.79       155

    accuracy                           0.68       600
   macro avg       0.69      0.68      0.68       600
weighted avg       0.69      0.68      0.68       600
```

In [51]:

```
# Tabulate the result

# create a list of column names
cols = ['Model', 'Precision Score', 'Recall Score','Accuracy Score','f1-score']

# creating an empty dataframe of the colums
result_tabulation = pd.DataFrame(columns = cols)

# compiling the required information
knn_estimator = pd.Series({'Model': "KNN",
                 'Precision Score': metrics.precision_score(y_test, y_pred_knn, aver
                 'Recall Score': metrics.recall_score(y_test, y_pred_knn, average="m
                 'Accuracy Score': metrics.accuracy_score(y_test, y_pred_knn),
                  'f1-score':metrics.f1_score(y_test, y_pred_knn, average="macro")})



# appending our result table
result_tabulation = result_tabulation.append(knn_estimator , ignore_index = True)

# view the result table
result_tabulation
```

Out[51]:

	Model	Precision Score	Recall Score	Accuracy Score	f1-score
0	KNN	0.69005	0.676924	0.678333	0.680475

5.2 Random Forest

...goto toc

In [53]:

```
# Fitting Random Forest Classification to the Training set
classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random
classifier.fit(X_train, y_train)
```

Out[53]: RandomForestClassifier(criterion='entropy', n_estimators=10, random_state=42)

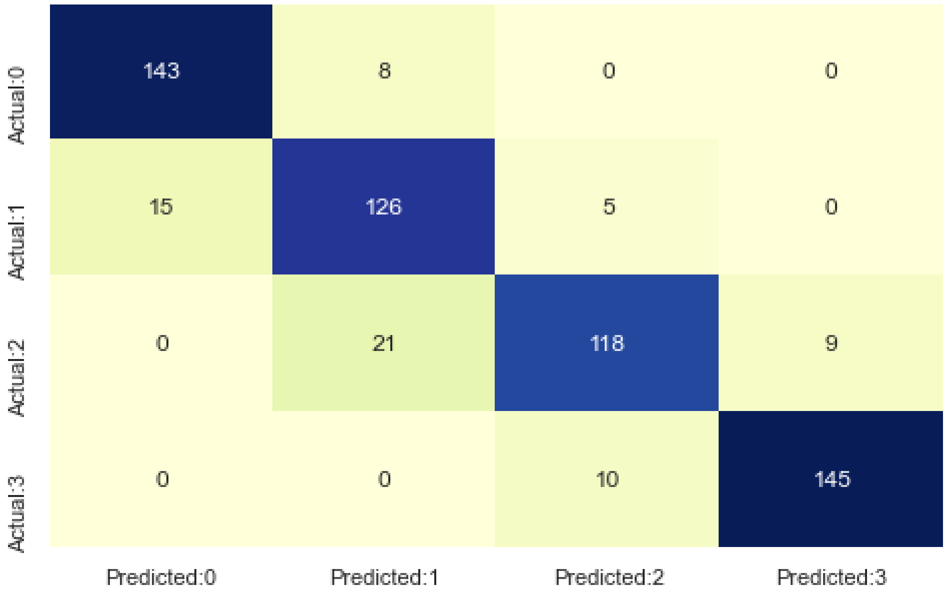
In [54]:

```
# Predicting the Test set results
y_pred_random = classifier.predict(X_test)
```

In [55]:

```
# compute the confusion matrix
cm = confusion_matrix(y_test, y_pred_random)

# label the confusion matrix
conf_matrix = pd.DataFrame(data=cm,columns=['Predicted:0','Predicted:1','Predicted:2
```

```python
# set sizeof the plot
plt.figure(figsize = (8,5))

# plot a heatmap
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap="YlGnBu", cbar=False)
plt.show()
```



In [57]:

```python
# Generate classification report

# accuracy measures by classification_report()
result = classification_report(y_test, y_pred_random)

# print the result
print(result)
```

```
              precision    recall  f1-score   support

         0.0       0.91      0.95      0.93       151
         1.0       0.81      0.86      0.84       146
         2.0       0.89      0.80      0.84       148
         3.0       0.94      0.94      0.94       155

    accuracy                           0.89       600
   macro avg       0.89      0.89      0.89       600
weighted avg       0.89      0.89      0.89       600
```

In [60]:

```python
# create the result table for all scores
random_forest_metrics = pd.Series({'Model': "Random Forest",
                 'Precision Score': metrics.precision_score(y_test, y_pred_random, a
                 'Recall Score': metrics.recall_score(y_test, y_pred_random, average
                 'Accuracy Score': metrics.accuracy_score(y_test, y_pred_random),
                  'f1-score':metrics.f1_score(y_test, y_pred_random, average="macro"



# appending our result table
result_tabulation = result_tabulation.append(random_forest_metrics , ignore_index =

# view the result table
result_tabulation
```

Out[60]:

	Model	Precision Score	Recall Score	Accuracy Score	f1-score
0	KNN	0.690050	0.676924	0.678333	0.680475
1	Random Forest	0.886686	0.885704	0.886667	0.885286

5.3 Naive Bayes

...goto toc

In [61]:

```
# build the model
GNB = GaussianNB()

# fit the model
GNB.fit(X_train, y_train)
```

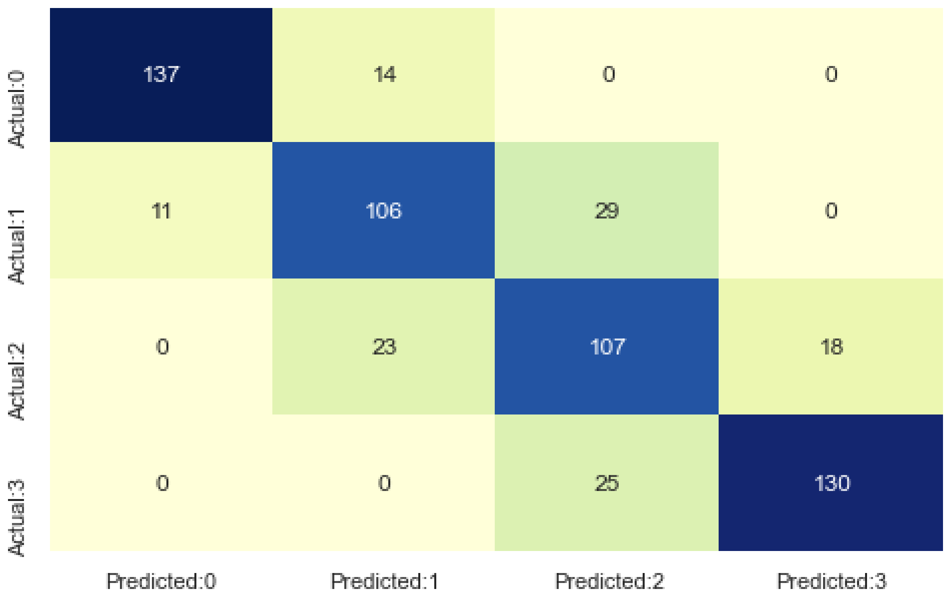
Out[61]: GaussianNB()

In [62]:

```
# predict the values
y_pred_GNB  = GNB.predict(X_test)
```

In [63]:

```
# compute the confusion matrix
cm = confusion_matrix(y_test, y_pred_GNB)

# label the confusion matrix
conf_matrix = pd.DataFrame(data=cm,columns=['Predicted:0','Predicted:1','Predicted:2

# set sizeof the plot
plt.figure(figsize = (8,5))

# plot a heatmap
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap="YlGnBu", cbar=False)
plt.show()
```



In [64]:

```
# Generate classiffication report

# accuracy measures by classification_report()
```

```
result = classification_report(y_test, y_pred_GNB)

# print the result
print(result)
```

```
              precision    recall  f1-score   support

         0.0       0.93      0.91      0.92       151
         1.0       0.74      0.73      0.73       146
         2.0       0.66      0.72      0.69       148
         3.0       0.88      0.84      0.86       155

    accuracy                           0.80       600
   macro avg       0.80      0.80      0.80       600
weighted avg       0.80      0.80      0.80       600
```

In [65]:

```
# create the result table for all scores
GNB_metrices = pd.Series({'Model': "Naive Bayes",
                 'Precision Score': metrics.precision_score(y_test, y_pred_GNB, aver
                 'Recall Score': metrics.recall_score(y_test, y_pred_GNB, average="m
                 'Accuracy Score': metrics.accuracy_score(y_test, y_pred_GNB),
                  'f1-score':metrics.f1_score(y_test, y_pred_GNB, average="macro")})



# appending our result table
result_tabulation = result_tabulation.append(GNB_metrices , ignore_index = True)

# view the result table
result_tabulation
```

Out[65]:

	Model	Precision Score	Recall Score	Accuracy Score	f1-score
0	KNN	0.690050	0.676924	0.678333	0.680475
1	Random Forest	0.886686	0.885704	0.886667	0.885286
2	Naive Bayes	0.802477	0.798749	0.800000	0.800149

5.4 Gradient Boosting

We will be performing hyperparameter tuning

...goto toc

In [68]:

```
# Choose the best Hyperparameters
# We have chosen, learning_rate, max_depth and the n_estimators.

# Define hyperparameters
parameters = {
    "n_estimators":[5,50,250,500],
    "max_depth":[1,3,5,7,9],
    "learning_rate":[0.01,0.1,1,10,100]
}

# Call the Boosting classifier constructor
gbc = GradientBoostingClassifier()
```

In [69]:

```python
# Use the GridSearhCV() for the cross -validation
cv = GridSearchCV(gbc,parameters,cv=5)

# Fit the data
cv.fit(X_train, y_train)
```

Out[69]:

```
GridSearchCV(cv=5, estimator=GradientBoostingClassifier(),
             param_grid={'learning_rate': [0.01, 0.1, 1, 10, 100],
                         'max_depth': [1, 3, 5, 7, 9],
                         'n_estimators': [5, 50, 250, 500]})
```

In [70]:

```python
# Function to display best parameters
def display(results):
    print(f'Best parameters are: {results.best_params_}')
    print("\n")
    mean_score = results.cv_results_['mean_test_score']
    std_score = results.cv_results_['std_test_score']
    params = results.cv_results_['params']
    for mean,std,params in zip(mean_score,std_score,params):
        print(f'{round(mean,3)} + or -{round(std,3)} for the {params}')
```

In [71]:

```python
# Display best parameters
display(cv)
```

```
Best parameters are: {'learning_rate': 1, 'max_depth': 1, 'n_estimators': 500}


0.734 + or -0.026 for the {'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 5}
0.754 + or -0.023 for the {'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 5
0}
0.76 + or -0.032 for the {'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 25
0}
0.781 + or -0.024 for the {'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 50
0}
0.784 + or -0.024 for the {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 5}
0.794 + or -0.023 for the {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 5
0}
0.836 + or -0.019 for the {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 25
0}
0.876 + or -0.011 for the {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 50
0}
0.841 + or -0.018 for the {'learning_rate': 0.01, 'max_depth': 5, 'n_estimators': 5}
0.857 + or -0.009 for the {'learning_rate': 0.01, 'max_depth': 5, 'n_estimators': 5
0}
0.879 + or -0.016 for the {'learning_rate': 0.01, 'max_depth': 5, 'n_estimators': 25
0}
0.885 + or -0.014 for the {'learning_rate': 0.01, 'max_depth': 5, 'n_estimators': 50
0}
0.834 + or -0.02 for the {'learning_rate': 0.01, 'max_depth': 7, 'n_estimators': 5}
0.846 + or -0.019 for the {'learning_rate': 0.01, 'max_depth': 7, 'n_estimators': 5
0}
0.868 + or -0.026 for the {'learning_rate': 0.01, 'max_depth': 7, 'n_estimators': 25
0}
0.877 + or -0.021 for the {'learning_rate': 0.01, 'max_depth': 7, 'n_estimators': 50
0}
0.824 + or -0.022 for the {'learning_rate': 0.01, 'max_depth': 9, 'n_estimators': 5}
0.832 + or -0.021 for the {'learning_rate': 0.01, 'max_depth': 9, 'n_estimators': 5
0}
0.852 + or -0.02 for the {'learning_rate': 0.01, 'max_depth': 9, 'n_estimators': 25
0}
0.86 + or -0.025 for the {'learning_rate': 0.01, 'max_depth': 9, 'n_estimators': 50
0}
0.754 + or -0.023 for the {'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 5}
0.781 + or -0.024 for the {'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 50}
0.871 + or -0.014 for the {'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 25
```

```
0}
0.901 + or -0.01 for the {'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 500}
0.792 + or -0.025 for the {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 5}
0.88 + or -0.007 for the {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 50}
0.894 + or -0.012 for the {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 25
0}
0.897 + or -0.013 for the {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 50
0}
0.852 + or -0.004 for the {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 5}
0.889 + or -0.012 for the {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 50}
0.899 + or -0.014 for the {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 25
0}
0.896 + or -0.015 for the {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 50
0}
0.842 + or -0.021 for the {'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 5}
0.88 + or -0.017 for the {'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 50}
0.892 + or -0.02 for the {'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 250}
0.895 + or -0.016 for the {'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 50
0}
0.834 + or -0.022 for the {'learning_rate': 0.1, 'max_depth': 9, 'n_estimators': 5}
0.854 + or -0.023 for the {'learning_rate': 0.1, 'max_depth': 9, 'n_estimators': 50}
0.878 + or -0.018 for the {'learning_rate': 0.1, 'max_depth': 9, 'n_estimators': 25
0}
0.884 + or -0.017 for the {'learning_rate': 0.1, 'max_depth': 9, 'n_estimators': 50
0}
0.784 + or -0.017 for the {'learning_rate': 1, 'max_depth': 1, 'n_estimators': 5}
0.899 + or -0.012 for the {'learning_rate': 1, 'max_depth': 1, 'n_estimators': 50}
0.917 + or -0.013 for the {'learning_rate': 1, 'max_depth': 1, 'n_estimators': 250}
0.921 + or -0.011 for the {'learning_rate': 1, 'max_depth': 1, 'n_estimators': 500}
0.86 + or -0.018 for the {'learning_rate': 1, 'max_depth': 3, 'n_estimators': 5}
0.893 + or -0.018 for the {'learning_rate': 1, 'max_depth': 3, 'n_estimators': 50}
0.897 + or -0.011 for the {'learning_rate': 1, 'max_depth': 3, 'n_estimators': 250}
0.899 + or -0.012 for the {'learning_rate': 1, 'max_depth': 3, 'n_estimators': 500}
0.856 + or -0.011 for the {'learning_rate': 1, 'max_depth': 5, 'n_estimators': 5}
0.891 + or -0.014 for the {'learning_rate': 1, 'max_depth': 5, 'n_estimators': 50}
0.895 + or -0.006 for the {'learning_rate': 1, 'max_depth': 5, 'n_estimators': 250}
0.895 + or -0.01 for the {'learning_rate': 1, 'max_depth': 5, 'n_estimators': 500}
0.866 + or -0.016 for the {'learning_rate': 1, 'max_depth': 7, 'n_estimators': 5}
0.893 + or -0.018 for the {'learning_rate': 1, 'max_depth': 7, 'n_estimators': 50}
0.886 + or -0.011 for the {'learning_rate': 1, 'max_depth': 7, 'n_estimators': 250}
0.87 + or -0.018 for the {'learning_rate': 1, 'max_depth': 7, 'n_estimators': 500}
0.868 + or -0.011 for the {'learning_rate': 1, 'max_depth': 9, 'n_estimators': 5}
0.887 + or -0.005 for the {'learning_rate': 1, 'max_depth': 9, 'n_estimators': 50}
0.876 + or -0.011 for the {'learning_rate': 1, 'max_depth': 9, 'n_estimators': 250}
0.864 + or -0.01 for the {'learning_rate': 1, 'max_depth': 9, 'n_estimators': 500}
0.121 + or -0.017 for the {'learning_rate': 10, 'max_depth': 1, 'n_estimators': 5}
0.121 + or -0.017 for the {'learning_rate': 10, 'max_depth': 1, 'n_estimators': 50}
0.121 + or -0.017 for the {'learning_rate': 10, 'max_depth': 1, 'n_estimators': 250}
0.121 + or -0.017 for the {'learning_rate': 10, 'max_depth': 1, 'n_estimators': 500}
0.221 + or -0.142 for the {'learning_rate': 10, 'max_depth': 3, 'n_estimators': 5}
0.221 + or -0.142 for the {'learning_rate': 10, 'max_depth': 3, 'n_estimators': 50}
0.221 + or -0.142 for the {'learning_rate': 10, 'max_depth': 3, 'n_estimators': 250}
0.221 + or -0.142 for the {'learning_rate': 10, 'max_depth': 3, 'n_estimators': 500}
0.362 + or -0.12 for the {'learning_rate': 10, 'max_depth': 5, 'n_estimators': 5}
0.369 + or -0.137 for the {'learning_rate': 10, 'max_depth': 5, 'n_estimators': 50}
0.346 + or -0.106 for the {'learning_rate': 10, 'max_depth': 5, 'n_estimators': 250}
0.284 + or -0.148 for the {'learning_rate': 10, 'max_depth': 5, 'n_estimators': 500}
0.799 + or -0.033 for the {'learning_rate': 10, 'max_depth': 7, 'n_estimators': 5}
0.605 + or -0.228 for the {'learning_rate': 10, 'max_depth': 7, 'n_estimators': 50}
0.709 + or -0.108 for the {'learning_rate': 10, 'max_depth': 7, 'n_estimators': 250}
0.619 + or -0.158 for the {'learning_rate': 10, 'max_depth': 7, 'n_estimators': 500}
0.838 + or -0.02 for the {'learning_rate': 10, 'max_depth': 9, 'n_estimators': 5}
0.808 + or -0.037 for the {'learning_rate': 10, 'max_depth': 9, 'n_estimators': 50}
0.731 + or -0.077 for the {'learning_rate': 10, 'max_depth': 9, 'n_estimators': 250}
0.772 + or -0.037 for the {'learning_rate': 10, 'max_depth': 9, 'n_estimators': 500}
0.103 + or -0.018 for the {'learning_rate': 100, 'max_depth': 1, 'n_estimators': 5}
0.103 + or -0.018 for the {'learning_rate': 100, 'max_depth': 1, 'n_estimators': 50}
0.103 + or -0.018 for the {'learning_rate': 100, 'max_depth': 1, 'n_estimators': 25
0}
```

```
0.103 + or -0.018 for the {'learning_rate': 100, 'max_depth': 1, 'n_estimators': 50
0}
0.177 + or -0.08 for the {'learning_rate': 100, 'max_depth': 3, 'n_estimators': 5}
0.177 + or -0.08 for the {'learning_rate': 100, 'max_depth': 3, 'n_estimators': 50}
0.177 + or -0.08 for the {'learning_rate': 100, 'max_depth': 3, 'n_estimators': 250}
0.176 + or -0.08 for the {'learning_rate': 100, 'max_depth': 3, 'n_estimators': 500}
0.289 + or -0.106 for the {'learning_rate': 100, 'max_depth': 5, 'n_estimators': 5}
0.335 + or -0.12 for the {'learning_rate': 100, 'max_depth': 5, 'n_estimators': 50}
0.289 + or -0.098 for the {'learning_rate': 100, 'max_depth': 5, 'n_estimators': 25
0}
0.31 + or -0.13 for the {'learning_rate': 100, 'max_depth': 5, 'n_estimators': 500}
0.64 + or -0.186 for the {'learning_rate': 100, 'max_depth': 7, 'n_estimators': 5}
0.501 + or -0.228 for the {'learning_rate': 100, 'max_depth': 7, 'n_estimators': 50}
0.584 + or -0.183 for the {'learning_rate': 100, 'max_depth': 7, 'n_estimators': 25
0}
0.538 + or -0.211 for the {'learning_rate': 100, 'max_depth': 7, 'n_estimators': 50
0}
0.794 + or -0.072 for the {'learning_rate': 100, 'max_depth': 9, 'n_estimators': 5}
0.786 + or -0.052 for the {'learning_rate': 100, 'max_depth': 9, 'n_estimators': 50}
0.721 + or -0.185 for the {'learning_rate': 100, 'max_depth': 9, 'n_estimators': 25
0}
0.775 + or -0.08 for the {'learning_rate': 100, 'max_depth': 9, 'n_estimators': 500}
```

Note:

Best parameters are:

- learning_rate = 1
- max_depth = 1
- n_estimators = 500

In [101…

```
# Train the classifier
GBM = GradientBoostingClassifier(learning_rate=1, max_depth=1, n_estimators=500)

# fit on data
GBM.fit(X_train, y_train)
```

Out[101…

```
GradientBoostingClassifier(learning_rate=1, max_depth=1, n_estimators=500)
```

In [112…

```
# Create a dataframe to store importance of features
feature_importance = pd.DataFrame()
feature_importance['feature'] = selected_features
feature_importance['Importance'] = GBM.feature_importances_

# Sort in decreasing order
feature_importance = feature_importance.sort_values(ascending = False, by = 'Importa
feature_importance
```

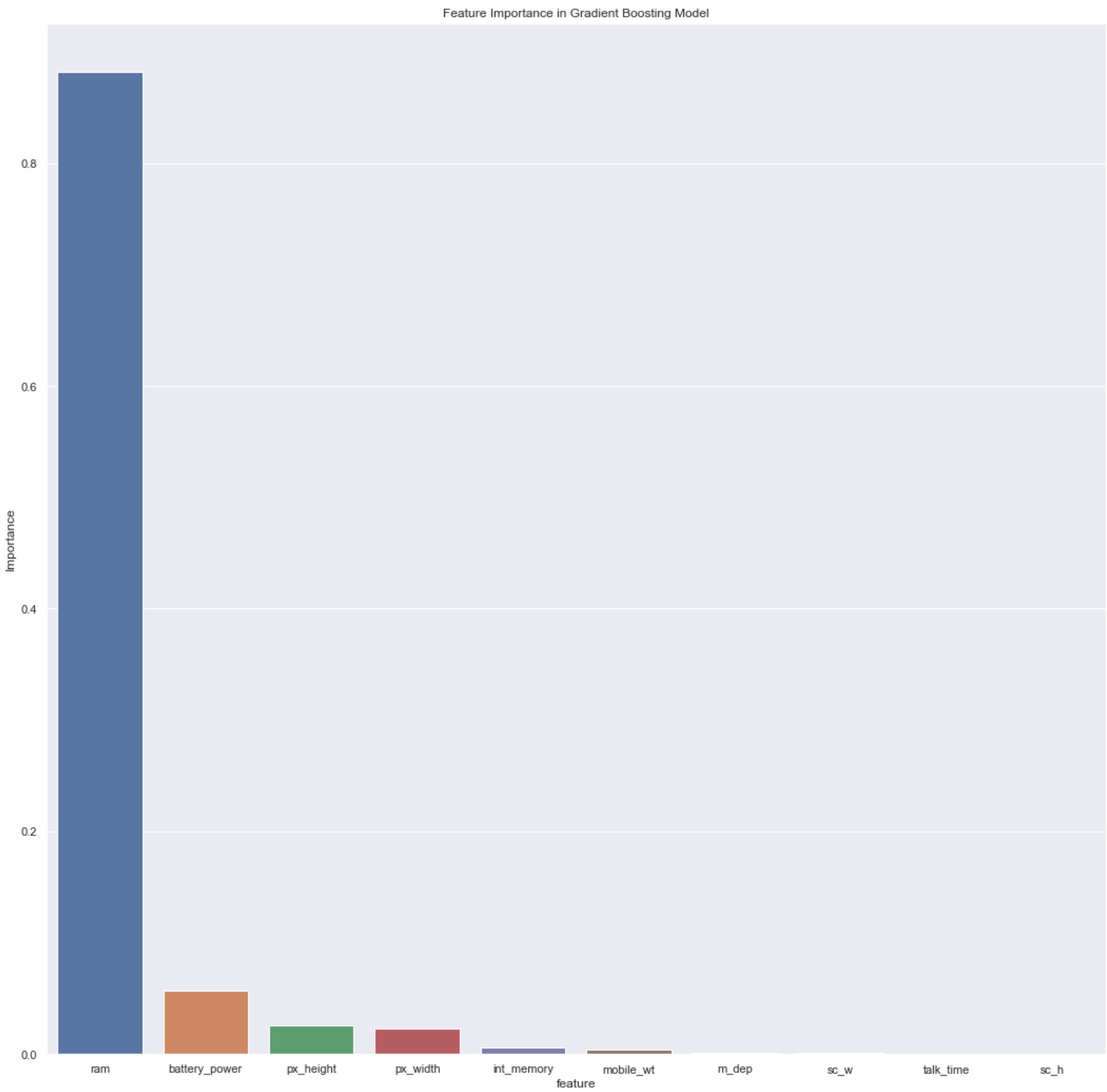
Out[112…

	feature	Importance
0	ram	0.881613
1	battery_power	0.057333
2	px_height	0.025328
3	px_width	0.022853
4	int_memory	0.005798
5	mobile_wt	0.003682
6	m_dep	0.001509

	feature	Importance
7	sc_w	0.001049
8	talk_time	0.000602
9	sc_h	0.000232

In [119…

```
# Plot feature importance
fig, ax = plt.subplots(figsize = (18,18))

# Barplot
sns.barplot(x = 'feature' , y = 'Importance', data = feature_importance, ax = ax)

# Add title
ax.set_title("Feature Importance in Gradient Boosting Model")

# Save the plot
fig.savefig('Feature_importance.png')

# Show the plot
plt.show()
```



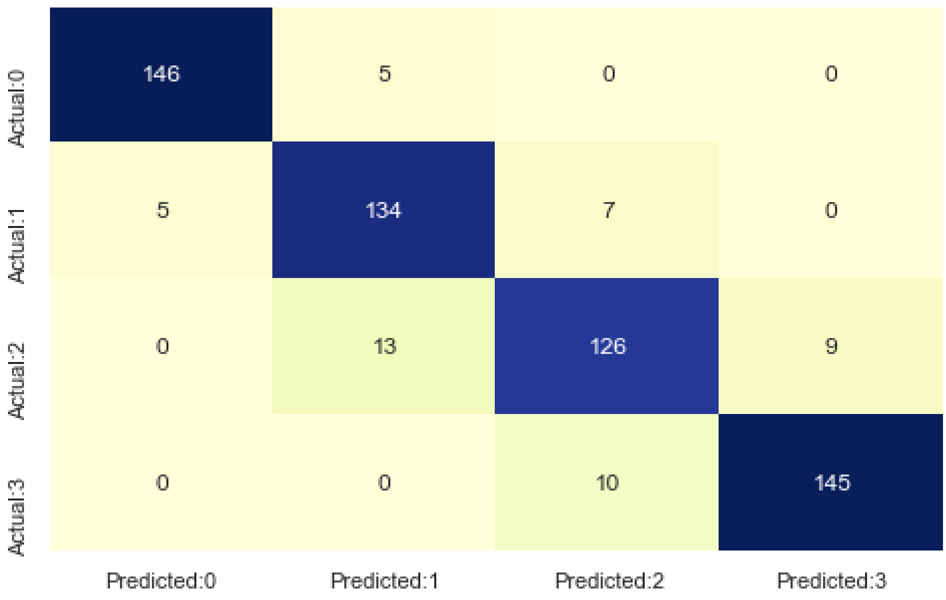
In [103…

```
# predict the values
```

```python
y_pred_gbm  = GBM.predict(X_test)
```

In [81]:

```python
# compute the confusion matrix
cm = confusion_matrix(y_test, y_pred_gbm)

# label the confusion matrix
conf_matrix = pd.DataFrame(data=cm,columns=['Predicted:0','Predicted:1','Predicted:2

# set sizeof the plot
plt.figure(figsize = (8,5))

# plot a heatmap
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap="YlGnBu", cbar=False)
plt.show()
```



In [82]:

```python
# Generate classification report

# accuracy measures by classification_report()
result = classification_report(y_test, y_pred_gbm)

# print the result
print(result)
```

```
              precision    recall  f1-score   support

         0.0       0.97      0.97      0.97       151
         1.0       0.88      0.92      0.90       146
         2.0       0.88      0.85      0.87       148
         3.0       0.94      0.94      0.94       155

    accuracy                           0.92       600
   macro avg       0.92      0.92      0.92       600
weighted avg       0.92      0.92      0.92       600
```

In [77]:

```python
# create the result table for all scores
GBM_metrices = pd.Series({'Model': "Gradient Boosting",
                 'Precision Score': metrics.precision_score(y_test, y_pred_gbm, aver
                 'Recall Score': metrics.recall_score(y_test, y_pred_gbm, average="m
                 'Accuracy Score': metrics.accuracy_score(y_test, y_pred_gbm),
                  'f1-score':metrics.f1_score(y_test, y_pred_gbm, average="macro")})
```

```python
# appending our result table
result_tabulation = result_tabulation.append(GBM_metrices , ignore_index = True)

# view the result table
result_tabulation
```

Out[77]:

	Model	Precision Score	Recall Score	Accuracy Score	f1-score
0	KNN	0.690050	0.676924	0.678333	0.680475
1	Random Forest	0.886686	0.885704	0.886667	0.885286
2	Naive Bayes	0.802477	0.798749	0.800000	0.800149
3	Gradient Boosting	0.917786	0.917883	0.918333	0.917677

6. Model Comparision

...goto toc

In [78]:

```python
result_tabulation
```

Out[78]:

	Model	Precision Score	Recall Score	Accuracy Score	f1-score
0	KNN	0.690050	0.676924	0.678333	0.680475
1	Random Forest	0.886686	0.885704	0.886667	0.885286
2	Naive Bayes	0.802477	0.798749	0.800000	0.800149
3	Gradient Boosting	0.917786	0.917883	0.918333	0.917677

Note: We can see that Gradient Boosting Method has outperformed.

In [84]:

```python
best_model = GBM
```

Save the model

In [121...

```python
pickle.dump(best_model, open("mobile_price_predictor.sav", "wb"))
```

...goto toc

Conclusion

During the analysis the given problem of predicting the price range we found that the problem is a multiclass classiffication problem and the dataset given is balanced with respect different categories of target feature (price_range).

We built different classifier for the given problem like KNN, Naive Bayes, Random Forest but **Gradient Boosting Classifier** outperformed other classifiers with test accuracy of **91.8%** and a f1-score of **0.917**.

Additionaly, while extracting feature importance from trained gradient boosting classifier we found that the feature **RAM** is the most important feature to predict price among all the features which is also true in practical scenario.

Best Model

Model	Precision Score	Recall Score	Accuracy Score	f1-score
Gradient Boosting	0.917786	0.917883	0.918333	0.917677

In [ ]: