Mobile Price Prediction

Submitted By

Ashwini Kantode
(EBEON0622608624)

Abstract

The aim of this research is to develop a model to predict the price of a mobile when the specifications of a mobile are given and to find the ML algorithm that predicts the price most accurately. The usage of archival data to accurately forecast forthcoming instances is the essence of Predictive Analytics. One of the ways Predictive Analytics can be performed is by using Machine Learning. Predictive Machine Learning works by taking in data as input to develop and train a prediction model and the trained model is used to predict the outcome of future data instances. Supervised Machine Learning algorithms make use of data that contains a pre-defined class label, which is the attribute that needs to be predicted. The class label is the price of a mobile in our case. The Mobile Price Class dataset sourced from the Kaggle data science community website (https://www.kaggle.com/datasets/faisalmohmmed/mobile-price-prediction) that categorizes mobiles into price ranges was used to train the prediction model. Python is used due to its readily accessible ML libraries. Various classification algorithms were used to train the model to try and find the algorithm that is able to predict the mobile price class most accurately. Metrics like accuracy score, confusion matrix, etc. are were used to evaluate the trained model to determine the algorithm most suitable among the ones used.

Table of Contents

Chapter 1

Introduction

Price is the most important component in the marketing of any product and is often the definitive factor in its sale to a consumer. In a constantly evolving and volatile market, the price is often the factor that makes or breaks a product. Setting an optimal price before the release of a product is imperative for any company. A tool that gives the estimated price of a product after weighing in the features it provides can come in handy and can help the company in making an informed decision while setting the market price for a product. Such a tool can also be used by a consumer to get an estimated price based on the features they are looking for in the product.

Nowadays, a Mobile is an essential accessory of a person. It is the fastest evolving and moving product in the technology market space. New mobiles with updated versions and new features are introduced into the market at a rapid pace. Thousands of mobiles are sold each day. In such a fast-paced and volatile market, a mobile company needs to set optimal prices to compete with its rivals. The first step in fixing a price is to estimate the price based on the features. The objective this research is to develop an ML model capable of estimating the price of a mobile phone based on its features. A potential buyer can also make use of the model to estimate the price of a mobile by inputting just the features they require into the tool. The same approach to create a prediction model can be used to develop a price estimation model for most products that have similar independent variable parameters. The price of a mobile is dependent on many features for example, the processor, battery capacity, camera quality, display size and thickness, etc. These features can be used to classify phones into various categories like entry-level, mid-range, flagship, premium, etc. Supervised ML algorithms are used in this paper as the dataset used has a definitive class label for price range.

Scenario:

We are going to work on a dataset which consist information about the various kinds mobile data and its price and other features. When we work on these sorts of data, we need to see which column is important for us and which is not. Our main is to make a model which can give us a good prediction on the price range.

Goals:

- Predict the price of the mobile dataset based on each features. This is ordinal outcome.
  0 = low price
  1 = medium price
  2 = high price
  3 = very high price
- Experiment with various Classification model and see which greatest accuracy.
- Determine which features are important to predict on the price of the mobile.

Chapter 2
Features and Predictors

Predictor (Y is 0, 1, 2 or 3 Price range of mobiles) is determine by 20 features(X):

1. **battery power** :Total energy a battery can store in one time measured in mAh
2. **blue**: Has Bluetooth or not (Binary)
3. **clock_speed**: speed at which microprocessor executes instructions
4. **dual_sim**: Has dual sim support or not (Binary)
5. **fc**: Front Camera mega pixels
6. **four_g**: Has 4G or not (Binary)
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 Megabytes
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 (Binary)
19. **touch_screen**: Has touch screen or not (Binary)
20. **wifi**: Has wifi or not (Binary)

Data has 3 types of data:
Continuous: Which is quantitative data that can be measured.
Ordinal data: Categorical data that has an order to it (0, 1, 2, 3, etc.).
Binary Data: Data whose unit can take on only two possible states (0&1).

Chapter 3
Methodology

1. Importing Python Libraries

Python has created several open-source libraries. A library is an initially merged collection of code scripts that can be used iteratively to save time. A Python library is a group of interconnected modules. It contains code bundles that can be reused in a variety of programs. Python libraries such as:

i. NumPy:

NumPy is one of the most widely used open-source Python packages, focusing on mathematical and scientific computation. It has built-in mathematical functions for convenient computation and facilitates large matrices and multidimensional data. It can be used for various things, including linear algebra, as an N-dimensional container for all types of data. The NumPy Array Python object defines an N-dimensional array with rows and columns.

ii. Pandas:

Pandas is an open source library. In the domain of data science, this well-known library is widely used. They're mostly used for analysis, manipulation, and cleaning of data, among other things. Pandas allows us to perform simple data modelling and analysis without having to swap.

iii. Matplotlib:

The plotting of numerical data is the responsibility of this library. It's for this reason that it's used in analysis of data. It's an open-source library that plots high-definition figures such as pie charts, scatterplots, boxplots, and graphs, among other things.

iv. Scikit learn:

Scikit learn is an open source machine learning library that supports supervised and unsupervised learning. It also provides various tools for model fitting, data preprocessing, model selection, model evaluation, and many other utilities.

2. Data collection:

The Mobile Price Class dataset sourced from the Kaggle data science community website that categorizes mobiles into price ranges used to train the prediction model.

Use: This type of estimation will help companies to predict the price of mobiles to give tough competition to another mobile manufacturer. It is also helpful for the customers who are investing money to buy mobile at the best price with the best features.

3. Data cleaning and Preprocessing

Data cleaning is fixing or removing incorrect, corrupted, incorrectly formatted, duplicate, or incomplete data within a dataset. If data is incorrect, outcomes and algorithms are unreliable, even though they may look correct. When combining multiple data sources, there are many opportunities for data to be duplicated or mislabelled. Data cleaning reduces errors and improves data quality. Correcting errors in data and eliminating bad records can be a time-consuming.

4. Machine Learning Algorithms

A Machine Learning system learns from historical data, builds the prediction models, and whenever it receives new data, predicts the output for it**.** The accuracy of predicted output depends upon the amount of data, as the huge amount of data helps to build a better model which predicts the output more accurately. We used classification model like as KNN, random forest, decision tree, and svm. These are supervised machine learning algorithm below:

a) K Nearest Neighbour (KNN):

K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique. KNN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories. KNN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm. KNN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems. KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

b) Random Forest:

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model. The random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

c) Decision Tree:

Decision Tree is a supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome. In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches. The decisions or the test are performed on the basis of features of the given dataset. It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.

d) SVM (Support Vector Machine):

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning. The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane. SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine.

e) Gradient Boosting
Gradient boosting is a machine learning technique used in regression and classification tasks, among others. It gives a prediction model in the form of an ensemble of weak prediction models, which are typically decision trees.

4. Implementation:

As we already discussed in the methodology section about some of the implementation details. So, the language used in this project is Python programming. We're running python code in anaconda navigator's Jupyter notebook. Jupyter notebook is much faster than Python IDE tools like PyCharm or Visual studio for implementing ML algorithms. The advantage of Jupyter notebook is that while writing code, it's really helpful for Data visualization and plotting some graphs like bar, pie and heatmap of correlated matrices. Implementation steps:
a) Dataset collection.
b) Importing Libraries: Numpy, Pandas, Scikit-learn, Matplotlib and Seaborn libraries were used.
c) Exploratory data analysis: For getting more insights about data.
d) Data cleaning and preprocessing: Checked for null and junk values using isnull().sum() functions of python. In preprocessing phase, we did feature engineering on our dataset. As we have numerical variables. And rename column name.
e) Model selection: We first separated x from y. x's are features or input variables of our datasets and y's are dependent or target variables which are crucial for predicting disease. Then using by the importing model selection function of the sklearn library, we splitted our x's and y's into train and test split using train_test_split () function of sklearn. We splitted 80% of our data for training and 20% for testing.
f) Applied ML models and created a confusion matrix of all models.
g) Deployment of the model which gave the best accuracy.

Analysis of Dataset

1. Libraries:

- import numpy as np
- import pandas as pd
- import matplotlib.pyplot as plt
- import seaborn as sns
- import warnings warnings.filterwarnings ('ignore')
- from sklearn.model_selection import train_test_split
- from sklearn.neighbors import KNeighborsClassifier
- from sklearn.ensemble import RandomForestClassifier
- from sklearn.tree import DecisionTreeClassifier
- from sklearn.svm import SVC

Data Wrangling:

mobile = pd.read_csv("mobile_price.csv")
mobile.head()

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

5 rows × 21 columns

```
mobile.shape # print shape of data
```

```
(2000, 21)
```

```
mobile.columns # print columns of data
```

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

```
mobile.isnull().sum() # sum of null values each column
```

```
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
```

```
mobile.describe() # summarize the count, standard deviation, min, max for numeric variable
```

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

8 rows × 21 columns

Observation:

1. Shape of dataset 2000 rows and 21 column.
2. 20 features and 1 (price range is predictor).
3. There are no missing values in the data.
4. The describe() method returns description of the data in the DataFrame.
5. All of the column types are in int or float.
6. The nunique () method returns the number of unique values for each column. In data has 3 types of data continuous(can be measured), ordinal(0,1,2..) and binary data(0,1).

Feature Engineering

Rename Features

```
mobile.rename(columns = {'blue':'bluetooth','fc':'front_camera','pc':'primary_camera','m_dep':'mobile_depth',
                         'mobile_wt':'mobile_weight','sc_h':'screen_height','sc_w':'screen_width'},inplace=True)
```

Pivot table of numeric data values

```
pd.pivot_table(mobile, index='price_range', values=['battery_power',  'clock_speed' , 'front_camera','int_memory',
                                    'mobile_weight', 'primary_camera', 'px_height','px_width','ram', 'screen_height',
                                                'screen_width', 'talk_time'])
```

price_range	battery_power	clock_speed	front_camera	int_memory	mobile_weight	primary_camera	px_height	px_width	ram	screen_height	screen_widt
0	1116.902	1.5502	4.084	31.174	140.552	9.574	536.408	1150.270	785.314	12.324	5.68
1	1228.868	1.4886	4.340	32.116	140.510	9.924	666.892	1251.908	1679.490	12.212	5.54
2	1228.320	1.5298	4.498	30.920	143.614	10.018	632.284	1234.046	2582.816	12.010	5.71
3	1379.984	1.5204	4.316	33.976	136.320	10.150	744.848	1369.838	3449.232	12.680	6.12

Pivot table of binary data values

```
pd.pivot_table(mobile,index='price_range',values=['bluetooth','dual_sim', 'four_g','three_g','touch_screen', 'wifi'])
```

price_range	bluetooth	dual_sim	four_g	three_g	touch_screen	wifi
0	0.486	0.500	0.518	0.746	0.524	0.496
1	0.490	0.510	0.524	0.756	0.522	0.504
2	0.486	0.498	0.494	0.774	0.470	0.504
3	0.518	0.530	0.550	0.770	0.496	0.524

find average of battery and talk time power with each price range

```
mobile.groupby('price_range')['talk_time','battery_power'].mean()
```

price_range	talk_time	battery_power
0	10.612	1116.902
1	11.362	1228.868
2	10.972	1228.320
3	11.098	1379.984

front and primary camera

```
mobile.groupby('price_range')['front_camera','primary_camera'].mean()
```

price_range	front_camera	primary_camera
0	4.084	9.574
1	4.340	9.924
2	4.498	10.018
3	4.316	10.150

Pixel height and width

```
mobile.groupby('price_range')['px_height', 'px_width'].mean()
```

price_range	px_height	px_width
0	536.408	1150.270
1	666.892	1251.908
2	632.284	1234.046
3	744.848	1369.838

talk time, battery and ram

```
mobile.groupby('price_range')['talk_time','battery_power','ram'].mean()
```

price_range	talk_time	battery_power	ram
0	10.612	1116.902	785.314
1	11.362	1228.868	1679.490
2	10.972	1228.320	2582.816
3	11.098	1379.984	3449.232

Value counts of each price range

```
mobile['price_range'].value_counts()
```

```
1    500
2    500
3    500
0    500
Name: price_range, dtype: int64
```

Features observation:

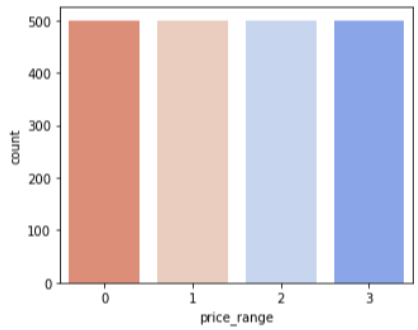
1. Rename columns of dataset.
2. Draw pivot table of numerical and binary values.
3. Group by columns: calculating average of battery power, talk time, front and primary camera, pixel height or width with each price range.
4. Price range: 500 count values of each price range (0, 1, 2. 3).

Exploratory Data Analysis

Price_range

Plot Count of each price range

```
plt.figure(figsize=(5,4))  # visualization of price range
sns.countplot(x='price_range',data=mobile,palette='coolwarm_r')
plt.show()
```

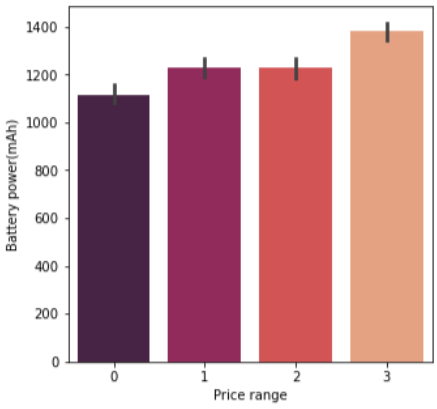


Observation:
There are mobile phones in 4 price ranges and count of mobile price range similar.

Battery

To check mobile battery power with price range

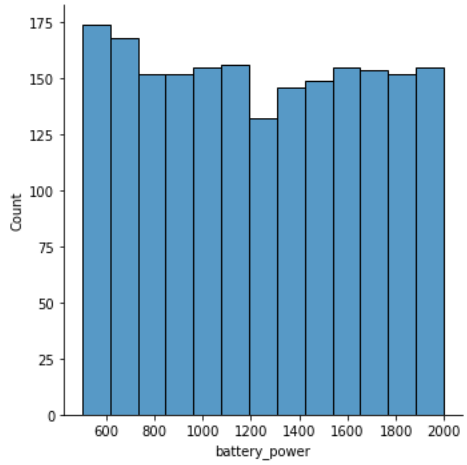
```
plt.figure(figsize=(5,5))
sns.barplot(x="price_range", y="battery_power", data=mobile,palette='rocket')
plt.ylabel('Battery power(mAh)')                    # y axis labels
plt.xlabel('Price range')                           # x ais labels
plt.show()
```



Observation:
Mobiles with battery power more than 1300 mAh has very high cost. And Mobiles with battery power between 1200 and 1300 mAh falls under medium and high cost category.

Count of battery power

```
sns.displot(data=mobile["battery_power"])
plt.show()
```



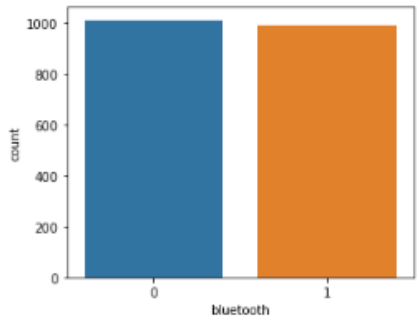
Observation:

This plot shows how the battery mAh is spread. there is a gradual increase as the price range increases.

Bluetooth

How many mobiles have bluetooth or not ?

```
plt.figure(figsize=(5,4))
sns.countplot(x='bluetooth',data=mobile)
plt.show()
```



Observation:

0 = mobile does not support bluetooth.

1 = mobile support bluetooth.

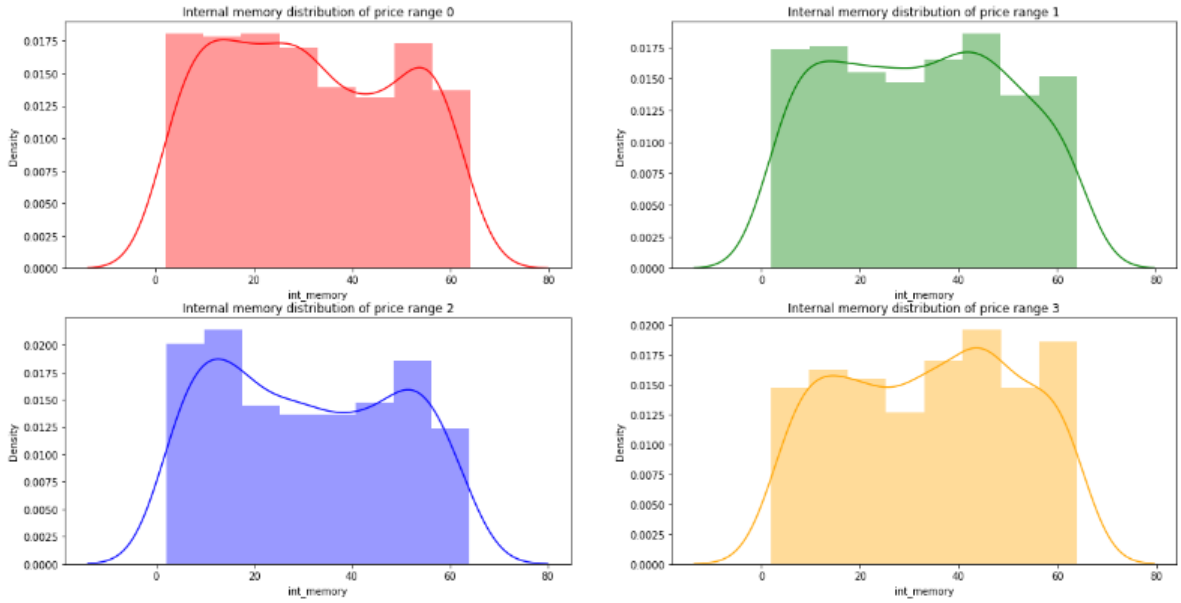
So, we can see that half the devices have Bluetooth, and half don't.

Internal Memory

Internal memory distribution per price range

```
mob_price_0 = mobile[mobile['price_range']==0] # low cost price range
mob_price_1 = mobile[mobile['price_range']==1] # medium cost price range
mob_price_2 = mobile[mobile['price_range']==2] # high cost price range
mob_price_3 = mobile[mobile['price_range']==3] # very high cost price range

plt.figure(figsize = (20, 10))
plt.subplot(2,2,1)
sns.distplot(mob_price_0['int_memory'],color='r')
plt.title('Internal memory distribution of price range 0')
plt.subplot(2,2,2)
sns.distplot(mob_price_1['int_memory'],color='g')
plt.title('Internal memory distribution of price range 1')
plt.subplot(2,2,3)
sns.distplot(mob_price_2['int_memory'],color='b')
plt.title('Internal memory distribution of price range 2')
plt.subplot(2,2,4)
sns.distplot(mob_price_3['int_memory'],color='orange')
plt.title('Internal memory distribution of price range 3')
plt.show()
```



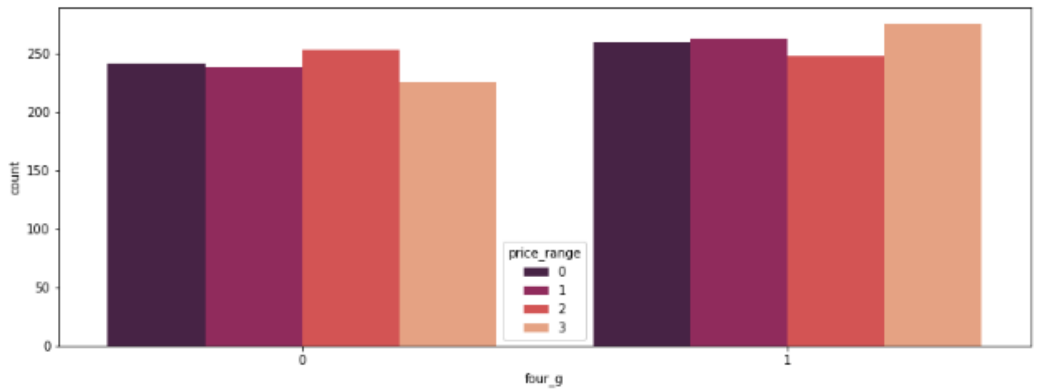
Observation:

We can see that internal memory sizes doesn't have many difference when it comes to distribution per price range.

4 g connectivity

How many mobiles have 4g or not based on price range ?

```
plt.figure(figsize = (14, 5))
sns.countplot(mobile['four_g'], hue = mobile['price_range'],palette='rocket')
plt.show()
```



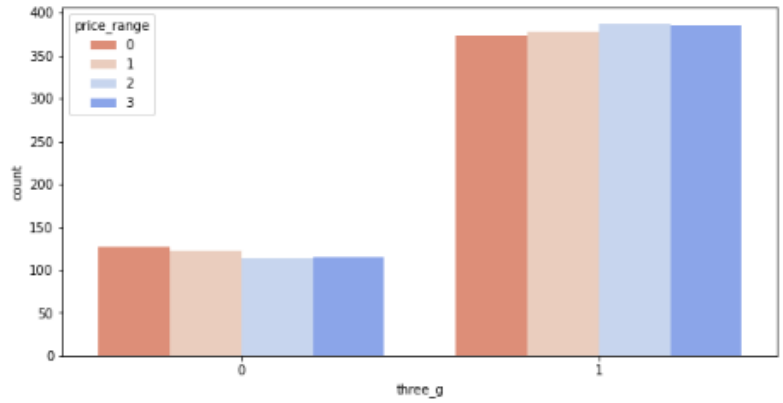
Observation:

0 is non 4g and 1 is 4g connectivity.
There are slightly more mobile with 4g compared to 4g line.
Mobile with price range 3 has the most 4gline among all group.

3 g connectivity

How many mobiles have 3g or not based on price range ?

```
plt.figure(figsize = (10, 5))
sns.countplot(mobile['three_g'], hue = mobile['price_range'],palette='coolwarm_r')
plt.show()
```



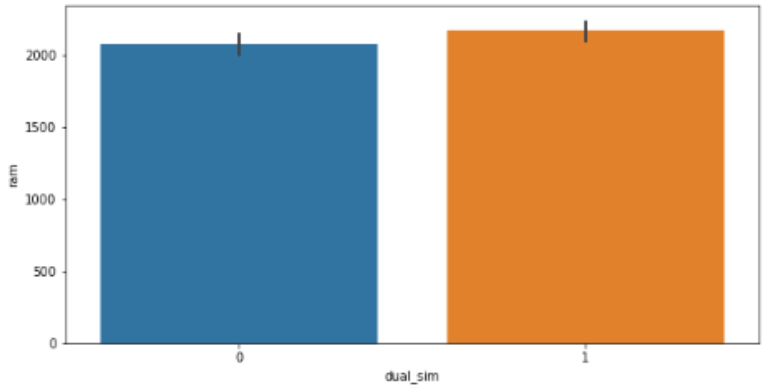
Observation:

0 is non 3g and 1 is 3g connectivity.
There are more 3g mobile as compared to non 3g.
Mobile with price range 2 & 3 have more 3g line.

Dual SIM

Compare between dual sim and RAM

```
plt.figure(figsize = (10, 5))
sns.barplot(mobile['dual_sim'], y = mobile['ram'])
plt.show()
```

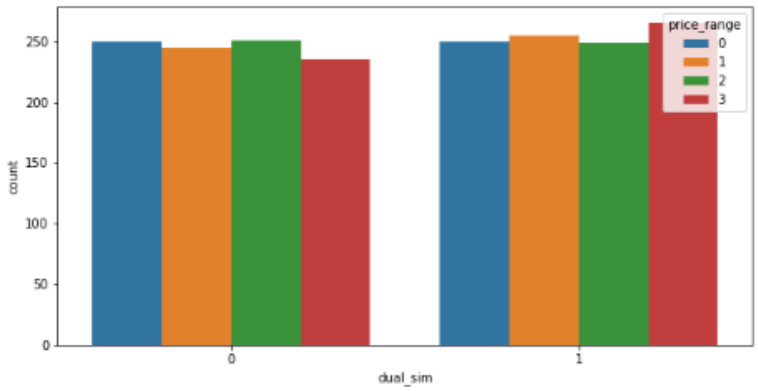


Observation:

- mobile with dual sim have a slightly higher ram compared to non dual sim.
- 0 = device does not support dual sim
- 1 = device support dual sim

How many mobiles have dual sim based on price range ?

```
plt.figure(figsize = (10, 5))
sns.countplot(mobile['dual_sim'], hue = mobile['price_range'])
plt.show()
```



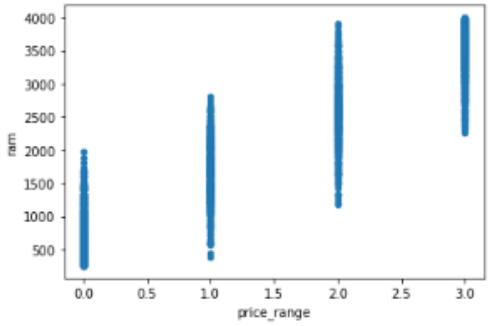
Observation:

- There is no major difference in term of number of mobiles that has dual sim in every price range.

RAM

Scatter plot of RAM based on price range

```
mobile.plot(x='price_range',y='ram',kind='scatter')
plt.show()
```



Observation:

- RAM has continuous increase with price range while moving from Low cost(0) to Very high cost(3).

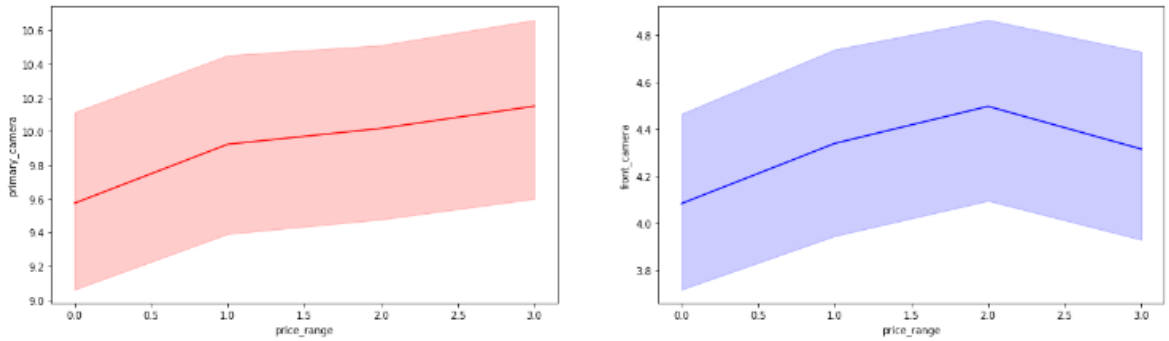
Primary and Front camera

To check camera feature based on price range

```
plt.figure(figsize = (20, 12))

plt.subplot(2,2,1)
sns.lineplot(x='price_range',y='primary_camera',data=mobile,color='r')

plt.subplot(2,2,2)
sns.lineplot(x='price_range',y='front_camera',data=mobile,color='b')
plt.show()
```



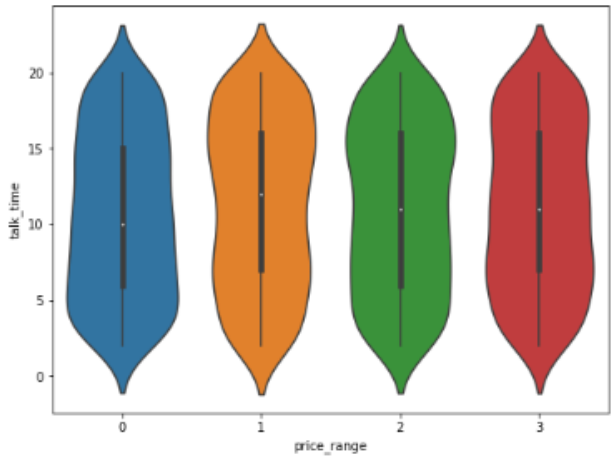
Observation:

- We can see that there's a slightly increase in average primary camera megapixel when the price range increase. However the average front camera megapixel drop when the price range goes from 2 to 3.

Talk Time

Compare Talktime with price range

```
plt.figure(figsize=(8,6))
sns.violinplot(y="talk_time", x="price_range", data=mobile)
plt.show()
```



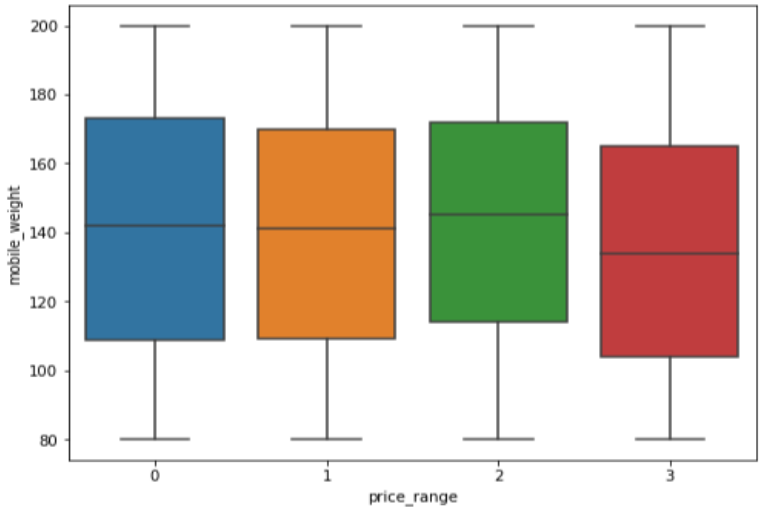
Observation:

- We can see talk time doesn't really affect much on the price range of a mobiles.

Mobile weight

To check mobiles weight depend upon price range

```
plt.figure(figsize=(8,6))
sns.boxplot(y="mobile_weight", x="price_range", data=mobile)
plt.show()
```



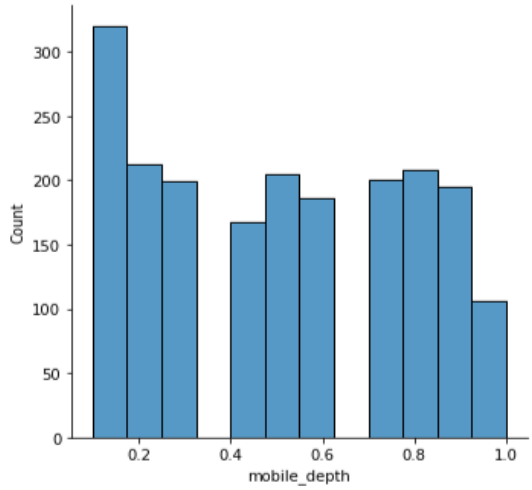
Observation:

- We can see that low price range of mobiles are heavy and costly mobiles are lighter.

Mobile depth

To check mobiles depth based on price range

```
sns.displot(data=mobile["mobile_depth"])
plt.show()
```



Observation:

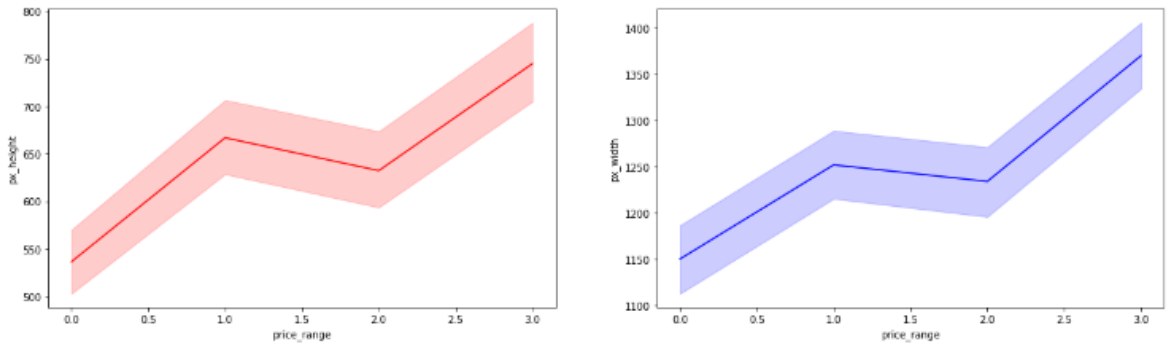
- A few mobiles are very thin and a few ones are almost a thick.

Pixel Height and Width

Plot pixcel height and with based on price range

```
plt.figure(figsize = (20, 12))

plt.subplot(2,2,1)
sns.lineplot(x ='price_range', y ='px_height', data = mobile,color='r')
plt.subplot(2,2,2)
sns.lineplot(x ='price_range', y ='px_width', data = mobile,color='b')
plt.show()
```



Observation:

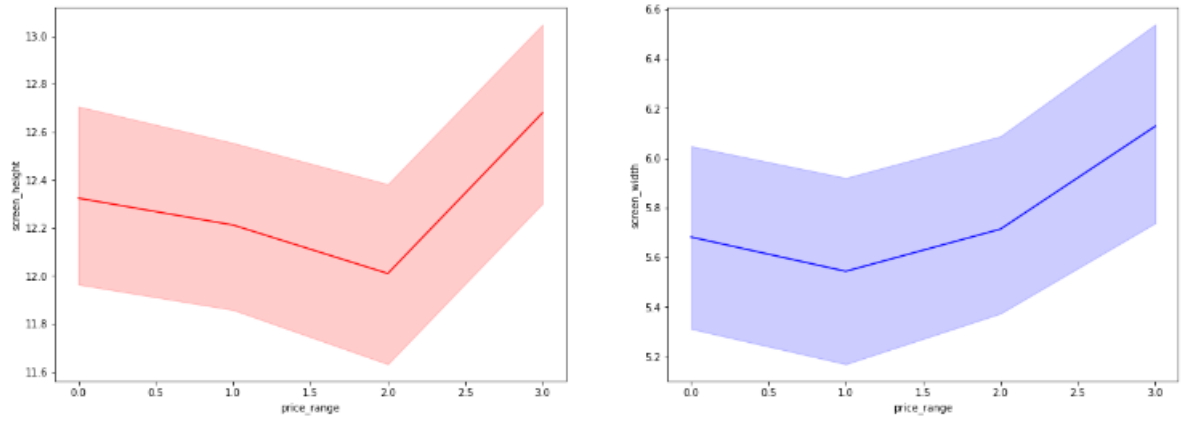
- We can see that both pixel width and height value generally goes up as the price range increase.

Screen Height and Width

To check mobile screen height and width based on price range ¶

```
plt.figure(figsize = (20, 15))

plt.subplot(2,2,1)
sns.lineplot(x ='price_range', y ='screen_height', data = mobile,color='r')
plt.subplot(2,2,2)
sns.lineplot(x ='price_range', y ='screen_width', data = mobile,color='b')
plt.show()
```



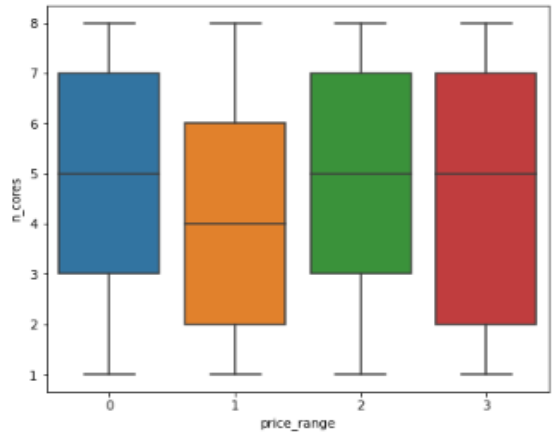
Observation:

- Mobiles having max screen height and width falls in very high price category.
- We can see in line chart of screen width and screen height from class 2 screen width and height starts increasing with price. Similar case is with px_height and px_width.As resolution of screen increases the price also increases.

Number of core processor

To check price range depend on numder of core processor

```
plt.figure(figsize = (7, 6))
sns.boxplot(x ='price_range', y ='n_cores', data = mobile)
plt.show()
```

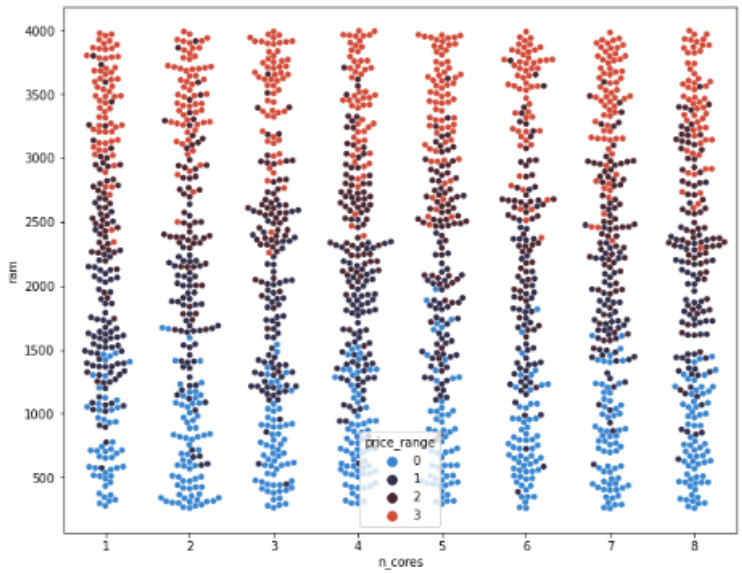


Observation:

- There are very few mobiles price range 1 with less number of cores. Most of the mobiles price range 2 and 3 are with high number of cores.

Relation between core processor, ram with price range

```
plt.figure(figsize = (10, 8))
sns.swarmplot(mobile['n_cores'], mobile['ram'], hue = mobile['price_range'], palette = 'icefire')
plt.show()
```

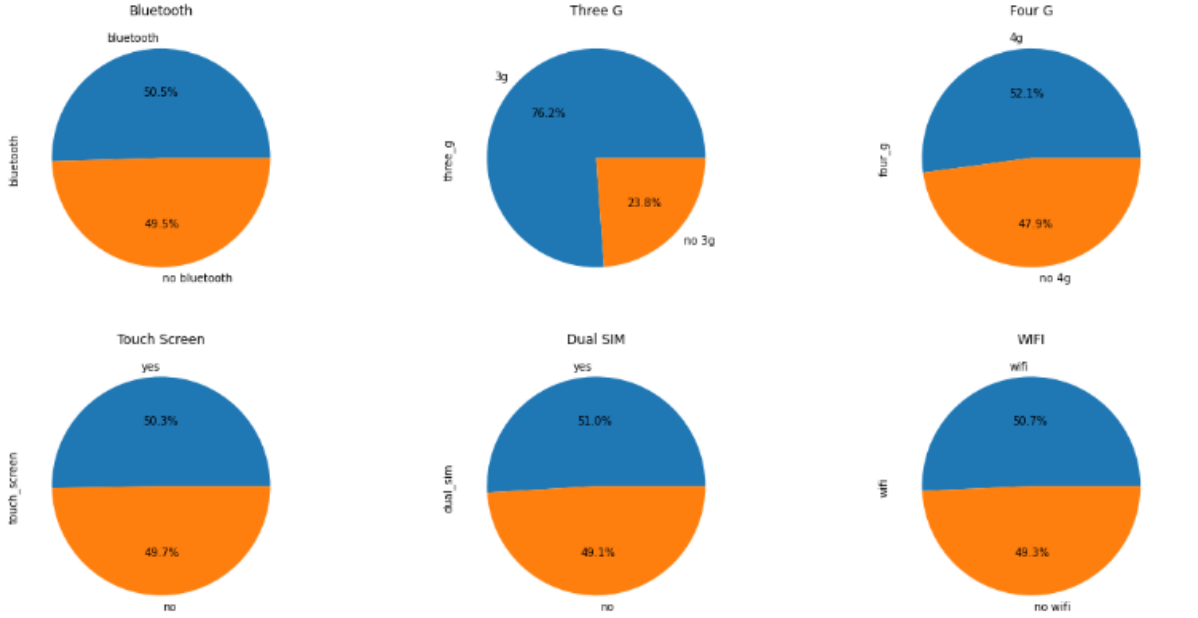


Observation:

- This graph shows that number of cores could vary in different size of ram. We have phone in price range 0 with 8 number of cores or phone with price range of 3 with 1 core.

Pie chart of binary category features of mobiles (Bluetooth, 3G, 4G, touch screen, dual sim, and wifi)

```
plt.figure(figsize = (20, 10))
plt.subplot(2,3,1)
mobile['bluetooth'].value_counts().plot.pie(autopct='%1.1f%%',labels=['bluetooth','no bluetooth'])
labels_blue=['no','yes']
plt.title('Bluetooth')
plt.subplot(2,3,2)
mobile['three_g'].value_counts().plot.pie(autopct='%1.1f%%',labels=['3g','no 3g'])
plt.title('Three G')
plt.subplot(2,3,3)
mobile['four_g'].value_counts().plot.pie(autopct='%1.1f%%',labels=['4g','no 4g'])
plt.title('Four G')
plt.subplot(2,3,4)
mobile['touch_screen'].value_counts().plot.pie(autopct='%1.1f%%',labels=['yes','no'])
plt.title('Touch Screen')
plt.subplot(2,3,5)
mobile['dual_sim'].value_counts().plot.pie(autopct='%1.1f%%',labels=['yes','no'])
plt.title('Dual SIM')
plt.subplot(2,3,6)
mobile['wifi'].value_counts().plot.pie(autopct='%1.1f%%',labels=['wifi','no wifi'])
plt.title('WIFI')
plt.show()
```

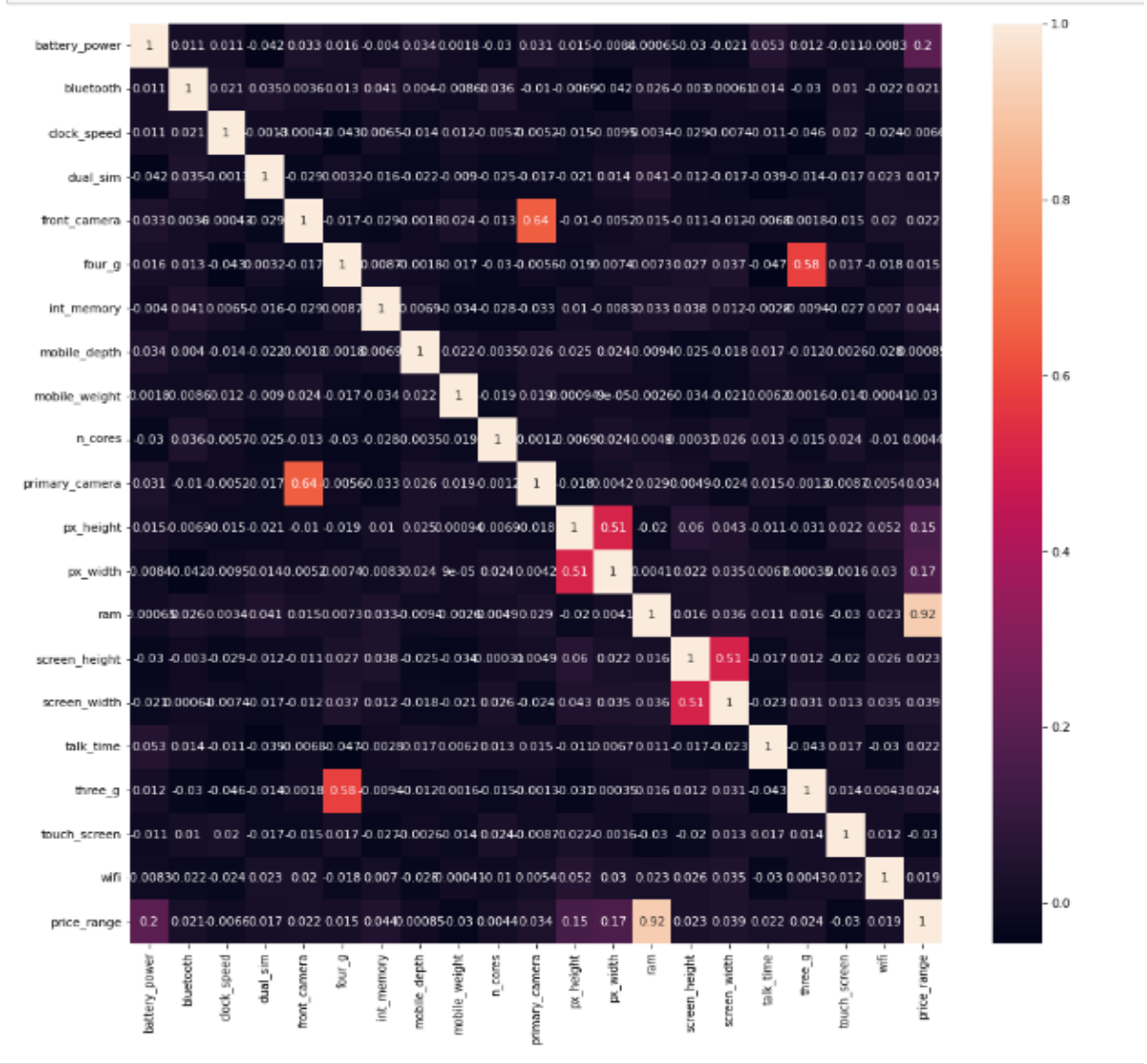


Observation:

- Percentage Distribution of Mobiles having bluetooth, dual sim, 4G, Wi-Fi and touchscreen are almost 50 %.
- Very few mobiles (23.8%) do not have 3g.

Heatmap

```
correlation = mobile.corr()
plt.figure(figsize = [15, 15])
sns.heatmap(correlation,annot = True)
plt.show()
```



Observation:

- RAM and price range shows high correlation which is a good, it signifies that RAM will play major deciding factor in estimating the price range.
- There is some collinearity in feature pairs ('pc', 'fc') and ('px_width', 'px_height'). Both correlations are justified since there are good chances that if front camera of a phone is good, the back camera would also be good.

Machine Learning Model

Use Classification model: The Supervised Machine Learning algorithm can be broadly classified into Regression and Classification Algorithms. In classification algorithms, we have predict the categorical values, we need Classification algorithms.

Why used classification model?
Thus our target variable (price range = 0, 1, 2, 3) has 4 categories so basically it is a classification problem.

Split data into train and test

```
from sklearn.model_selection import train_test_split
x = mobile.drop(['price_range'],axis=1)
y = mobile['price_range']
```

```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.2)
print("shape of X Train :",(x_train.shape))
print("shape of X Test :",(x_test.shape))
print("shape of Y Train :",(y_train.shape))
print("shape of Y Test :",(y_test.shape))
```

```
shape of X Train : (1500, 20)
shape of X Test : (500, 20)
shape of Y Train : (1500,)
shape of Y Test : (500,)
```

Observation:

- We split data into train (80%) and test size 0.2 (20%).

Model 1. KNN (K-Nearest Neighbour)

```
from sklearn.neighbors import KNeighborsClassifier
```

```
knn = KNeighborsClassifier(n_neighbors=10)  # get instance model
```

```
knn.fit(x_train,y_train) # fit/train model
```

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

```
acc_knn = knn.score(x_test,y_test)*100   #  X/Y test accuracy
acc_knn
```

```
92.0
```

```
y_predict = knn.predict(x_test)
```

```
from sklearn.metrics import classification_report
print('KNeighborsClassifier classification report\n' )
print(classification_report(y_test,y_predict))
```

```
KNeighborsClassifier classification report

              precision    recall  f1-score   support

           0       0.97      0.98      0.97        92
           1       0.89      0.96      0.92        96
           2       0.86      0.90      0.88       106
           3       0.98      0.86      0.91       106

    accuracy                           0.92       400
   macro avg       0.92      0.92      0.92       400
weighted avg       0.92      0.92      0.92       400
```

- Accuracy of KNN classifier is 92%.

Model 2. Random Forest

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=300)  # get instace of model
```

```
rf.fit(x_train,y_train)    # fit/train model
```

```
RandomForestClassifier(n_estimators=300)
```

```
acc_rf = rf.score(x_test,y_test)*100   # x/y test accuracy
acc_rf
```

```
88.0
```

```
y_predict = rf.predict(x_test)
```

```
from sklearn.metrics import classification_report
print('RandomForestClassifier classification report\n' )
print(classification_report(y_test,y_predict))   # output accuracy
```

```
RandomForestClassifier classification report

              precision    recall  f1-score   support

           0       0.92      0.95      0.93        92
           1       0.83      0.84      0.84        96
           2       0.81      0.87      0.84       106
           3       0.98      0.87      0.92       106

    accuracy                           0.88       400
   macro avg       0.88      0.88      0.88       400
weighted avg       0.88      0.88      0.88       400
```

- Accuracy of random forest is 88%.

Model 3. Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
```

```
dt = DecisionTreeClassifier()   # get instance of model
```

```
dt.fit(x_train,y_train)  # train/fit model
```

```
DecisionTreeClassifier()
```

```
acc_dt = dt.score(x_test,y_test)*100 # x/y test accuracy
acc_dt
```

```
84.75
```

```
y_predict = dt.predict(x_test) # get y prediction
```

```
from sklearn.metrics import classification_report
print('Decision Tree Classifier classification report\n' )
print(classification_report(y_test,y_predict))   # output accuracy
```

```
Decision Tree Classifier classification report

              precision    recall  f1-score   support

           0       0.91      0.90      0.91        92
           1       0.77      0.85      0.81        96
           2       0.82      0.75      0.78       106
           3       0.90      0.89      0.90       106

    accuracy                           0.85       400
   macro avg       0.85      0.85      0.85       400
weighted avg       0.85      0.85      0.85       400
```

- Accuracy of decision tree is 85%.

Model 4. SVM (Support vector machine)

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

```
svm = SVC() # get instance of model
```

```
svm.fit(x_train,y_train) # train/fit model
```

```
SVC()
```

```
acc_svm = svm.score(x_test,y_test)*100
acc_svm
```

```
94.19999999999999
```

```
y_predict = svm.predict(x_test) # get y predictions
```

```
from sklearn.metrics import classification_report
print('SVM Classifier classification report\n' )
print(classification_report(y_test,y_predict)) # output accuracy
```

```
SVM Classifier classification report

              precision    recall  f1-score   support

           0       0.99      0.98      0.98       127
           1       0.90      0.98      0.94       124
           2       0.96      0.85      0.90       136
           3       0.92      0.96      0.94       113

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

- Accuracy of SVM is 94%.

Model 5. Gradient Boosting

```
from sklearn.ensemble import GradientBoostingClassifier
gr =  GradientBoostingClassifier(n_estimators=50,learning_rate = 0.1)
```

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

```
GradientBoostingClassifier(n_estimators=50)
```

```
acc_gr = gr.score(x_test, y_test) * 100
acc_gr
```

```
86.8
```

```
y_predict = gr.predict(x_test)
```

```
from sklearn.metrics import classification_report
print('Gradient Boosting Classifier classification report\n' )
print(classification_report(y_test,y_predict)) # output accuracy
```

```
Gradient Boosting Classifier classification report

              precision    recall  f1-score   support

           0       0.96      0.94      0.95       127
           1       0.80      0.86      0.83       124
           2       0.82      0.78      0.80       136
           3       0.90      0.90      0.90       113

    accuracy                           0.87       500
   macro avg       0.87      0.87      0.87       500
weighted avg       0.87      0.87      0.87       500
```

- Accuracy of gradient boosting is 87%.
- From comparing the 5 models, we can conclude that Model 4: SVM yields the highest accuracy. With an accuracy of 94%.
- We have precision, recall, f1-score and support:
- Precision : be "how many are correctly classified among that class"
- Recall : "how many of this class you find over the whole number of element of this class"
- F1-score: harmonic mean of precision and recall values. F1 score reaches its best value at 1 and worst value at 0. F1 Score = 2 x ((precision x recall) / (precision + recall))
- Support: of samples of the true response that lie in that class.

Making the Confusion Matrix:

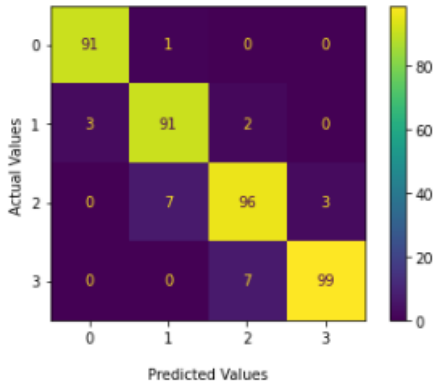
Confusion Matrix is a tool to determine the performance of classifier. It contains information about actual and predicted classifications.

```
from sklearn.metrics import confusion_matrix
svm_matrix = confusion_matrix(y_test,y_predict)
svm_matrix
```

```
array([[91,  1,  0,  0],
       [ 3, 91,  2,  0],
       [ 0,  7, 96,  3],
       [ 0,  0,  7, 99]], dtype=int64)
```

Confusion matrix of SVM

```
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(svm,x_test,y_test)
plt.xlabel('\nPredicted Values')
plt.ylabel('Actual Values ')
plt.show()
```



Chapter 4
Analysis of Result

Models table

```
Models = pd.DataFrame({'Model': ['K Nearest neighbor','Random Forest','Decision Tree','SVM','Gradient Boosting'],
                       'Accuracy':[acc_knn,acc_rf,acc_dt,acc_svm,acc_gr]})
Models
```

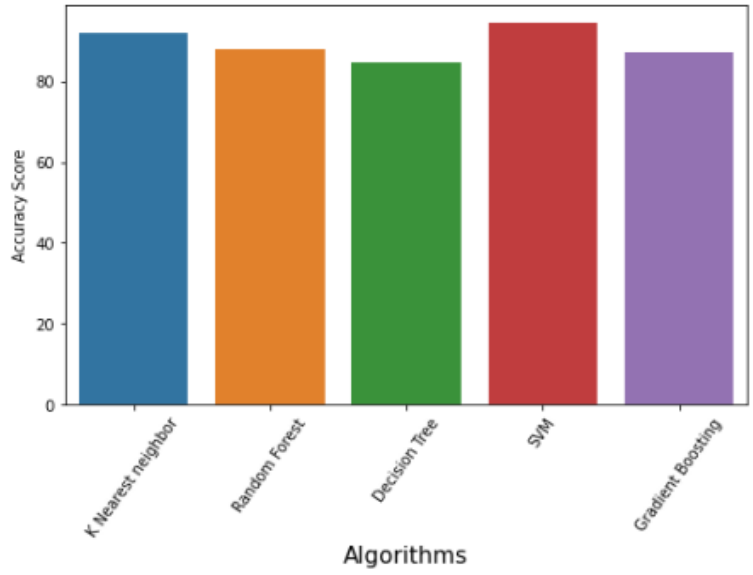
	Model	Accuracy
0	K Nearest neighbor	92.00
1	Random Forest	88.00
2	Decision Tree	84.75
3	SVM	94.25
4	Gradient Boosting	87.00

Observation:

- In the table we have seen Accuracy scored of our model is very good.

Bar plot of algorithm models

```
plt.figure(figsize=(8,5))
sns.barplot(x='Model',y='Accuracy',data=Models)
plt.xticks(rotation=55)
plt.xlabel("Algorithms",fontsize=15)
plt.ylabel("Accuracy Score")
plt.show()
```



Observation:

After analysis our dataset with five different model, we conclude that KNN and SVM is best model for our dataset. (the highest accuracy score = 94.2%).

Chapter 5
Conclusion

- From EDA we can see that here are mobile phones in 4 price ranges. The number of elements is almost similar.
- Half the devices have Bluetooth, and half don’t.
- There is a gradual increase in battery as the price range increases.
- Ram has continuous increase with price range while moving from Low cost to Very high cost.
- Costly phones are lighter.
- RAM, battery power, pixels played more significant role in deciding the price range of mobile phone.
- Form all the above experiments we can conclude that SVM classifier, gradient boosting and, KNN with confusion matrix we got good accuracy.

Reference

- Manisha-sirsat.blogspot.com, 'What is Confusion Matrix and Advanced Classification Metrics?' .Available: https://manisha-sirsat.blogspot.com/2019/04/confusion-matrix.html

- 'Mobile Price Prediction through Four Classification Algorithms '.Available: https://www.analyticsvidhya.com/blog/2022/02/learn-mobile-price-prediction-through-four-classification-algorithms/

- Python library: https://scikit-learn.org/stable/supervised_learning.html#supervised-learning

- Muhammad Asim, Zafar Khan, "Mobile Price Class prediction using Machine Learning Techniques", International Journal of Computer Applications (0975 – 8887) Volume 179 – No.29, March 2018, doi: 10.5120/ijca2018916555. [3]. P. Arora, S. Srivastava and B. Garg, "MOBILE PRICE PREDICTION USING WEKA", 2020.

- https://en.wikipedia.org/wiki/Python_(programming_language)

- Dataset from kaggle :https://www.kaggle.com/datasets/faisalmohmmed/mobile-price-prediction