```
import numpy as np
import pandas as pd

# daata visualization
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')

df = pd.DataFrame(pd.read_csv('dataset.csv'))

df.head()
```

```
   battery_power  blue  clock_speed  dual_sim  fc  four_g  int_memory
m_dep  \
0            500     0          1.3         1  13       0          64
0.9
1            500     0          1.4         0   0       0          28
0.4
2            501     0          2.3         0  12       1          54
0.3
3            501     1          0.5         1  14       0          22
0.5
4            502     0          1.5         1   7       0          37
0.2

   mobile_wt  n_cores  ...  px_height  px_width   ram  sc_h  sc_w
talk_time  \
0        134        4  ...       1384      1912  2807    19     7
7
1        128        3  ...       1010      1854  3460     9     3
3
2        131        4  ...        504      1089  2346    13    12
2
3        174        6  ...        239      1636  3077    17     3
17
4        199        2  ...        705      1810  1649     6     1
14

   three_g  touch_screen  wifi  price_range
0        0             0     1            1
1        1             1     1            3
2        1             0     1            1
3        0             0     0            2
4        0             1     0            1

[5 rows x 21 columns]
```

```
#phase1
```

Task 1

1. Import the relevant packages
2. Import the datasets
3. Walkthrough the characteristics of the dataset
4. Check out if there is any gap

```
df.info() # we got 21 cols
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000 entries, 0 to 2999
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   battery_power  3000 non-null   int64  
 1   blue           3000 non-null   int64  
 2   clock_speed    3000 non-null   float64
 3   dual_sim       3000 non-null   int64  
 4   fc             3000 non-null   int64  
 5   four_g         3000 non-null   int64  
 6   int_memory     3000 non-null   int64  
 7   m_dep          3000 non-null   float64
 8   mobile_wt      3000 non-null   int64  
 9   n_cores        3000 non-null   int64  
 10  pc             3000 non-null   int64  
 11  px_height      3000 non-null   int64  
 12  px_width       3000 non-null   int64  
 13  ram            3000 non-null   int64  
 14  sc_h           3000 non-null   int64  
 15  sc_w           3000 non-null   int64  
 16  talk_time      3000 non-null   int64  
 17  three_g        3000 non-null   int64  
 18  touch_screen   3000 non-null   int64  
 19  wifi           3000 non-null   int64  
 20  price_range    3000 non-null   int64  
dtypes: float64(2), int64(19)
memory usage: 492.3 KB
```

```
df.isnull().sum() # so there is no null 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
```

Task 2

1. Perform the data cleaning actions
2. Rename the attributes with below values for further operations;

```
df.rename(columns={"battery_power":"BatteryPower",
                   "blue":"Bluetooth",
                  "clock_speed":"ClockSpeed",
                  "dual_sim":"DualSim",
                  "fc":"FrontCameraMP",
                   "pc":"PrimaryCameraMP",
                  "four_g":"4G",
                  "int_memory":"InternalMemory",
                  "m_dep":"MobileDepth",
                  "mobile_wt":"Weight",
                  "n_cores":"Numberofcores",
                  "px_height":"PixelR.Height",
                  "px_width":"PixelR.Width",
                  "ram":"Ram(MB)",
                  "sc_h":"ScreenHeight",
                  "sc_w":"ScreenWidth",
                  "talk_time":"LongestBatteryCharge",
                  "three_g":"3G",
                  "touch_screen":"TouchScreen",
                  "wifi":"WIFI",
                  "price_range":"PriceRange"},inplace=True)

df.head(1)
```

```
   BatteryPower  Bluetooth  ClockSpeed  DualSim  FrontCameraMP  4G  \
0           500          0         1.3        1             13   0

   InternalMemory  MobileDepth  Weight  Numberofcores  ...
PixelR.Height  \
0              64          0.9     134              4  ...
1384
```

```
   PixelR.Width  Ram(MB)  ScreenHeight  ScreenWidth
LongestBatteryCharge  3G  \
0          1912     2807            19            7
7   0

   TouchScreen  WIFI  PriceRange
0            0     1           1

[1 rows x 21 columns]
```

```
unique_vals = df.nunique()

print(unique_vals)
```

```
BatteryPower            1275
Bluetooth                  2
ClockSpeed                26
DualSim                    2
FrontCameraMP             20
4G                         2
InternalMemory            63
MobileDepth               10
Weight                   121
Numberofcores              8
PrimaryCameraMP           21
PixelR.Height           1333
PixelR.Width            1313
Ram(MB)                 2070
ScreenHeight              15
ScreenWidth               19
LongestBatteryCharge      19
3G                         2
TouchScreen                2
WIFI                       2
PriceRange                 4
dtype: int64
```

```
diff = df['LongestBatteryCharge'].mean() -
df['LongestBatteryCharge'].min()
print("Approximate difference between min and avg talk time:", diff)
```

```
Approximate difference between min and avg talk time:
9.035666666666666
```

```
avg_ram = df["Ram(MB)"].mean()

print("Average RAM for most mobiles:", avg_ram)
```

```
Average RAM for most mobiles: 2129.1413333333335
```

```
commin = df['InternalMemory'].mode()[0]
print(commin)
```

```
44
```

```
most_common = df['your_column'].mode()[0]
```

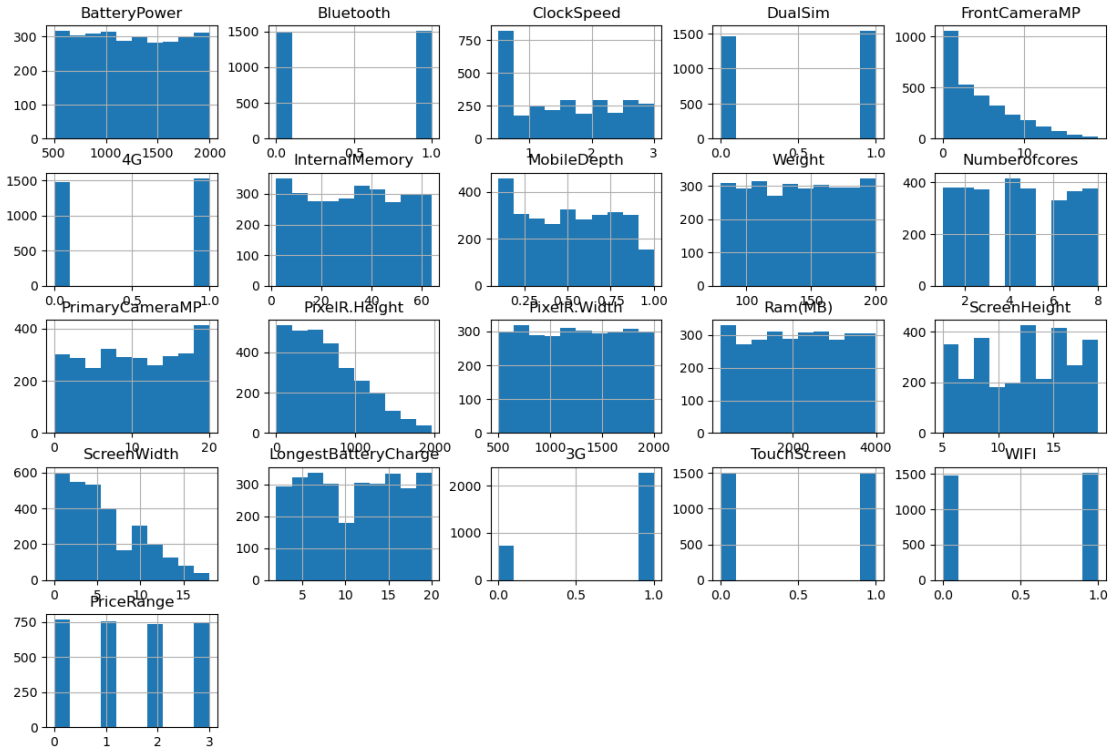
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000 entries, 0 to 2999
Data columns (total 21 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   BatteryPower          3000 non-null   int64  
 1   Bluetooth             3000 non-null   int64  
 2   ClockSpeed            3000 non-null   float64
 3   DualSim               3000 non-null   int64  
 4   FrontCameraMP         3000 non-null   int64  
 5   4G                    3000 non-null   int64  
 6   InternalMemory        3000 non-null   int64  
 7   MobileDepth           3000 non-null   float64
 8   Weight                3000 non-null   int64  
 9   Numberofcores         3000 non-null   int64  
 10  PrimaryCameraMP       3000 non-null   int64  
 11  PixelR.Height         3000 non-null   int64  
 12  PixelR.Width          3000 non-null   int64  
 13  Ram(MB)               3000 non-null   int64  
 14  ScreenHeight          3000 non-null   int64  
 15  ScreenWidth           3000 non-null   int64  
 16  LongestBatteryCharge  3000 non-null   int64  
 17  3G                    3000 non-null   int64  
 18  TouchScreen           3000 non-null   int64  
 19  WIFI                  3000 non-null   int64  
 20  PriceRange            3000 non-null   int64
```

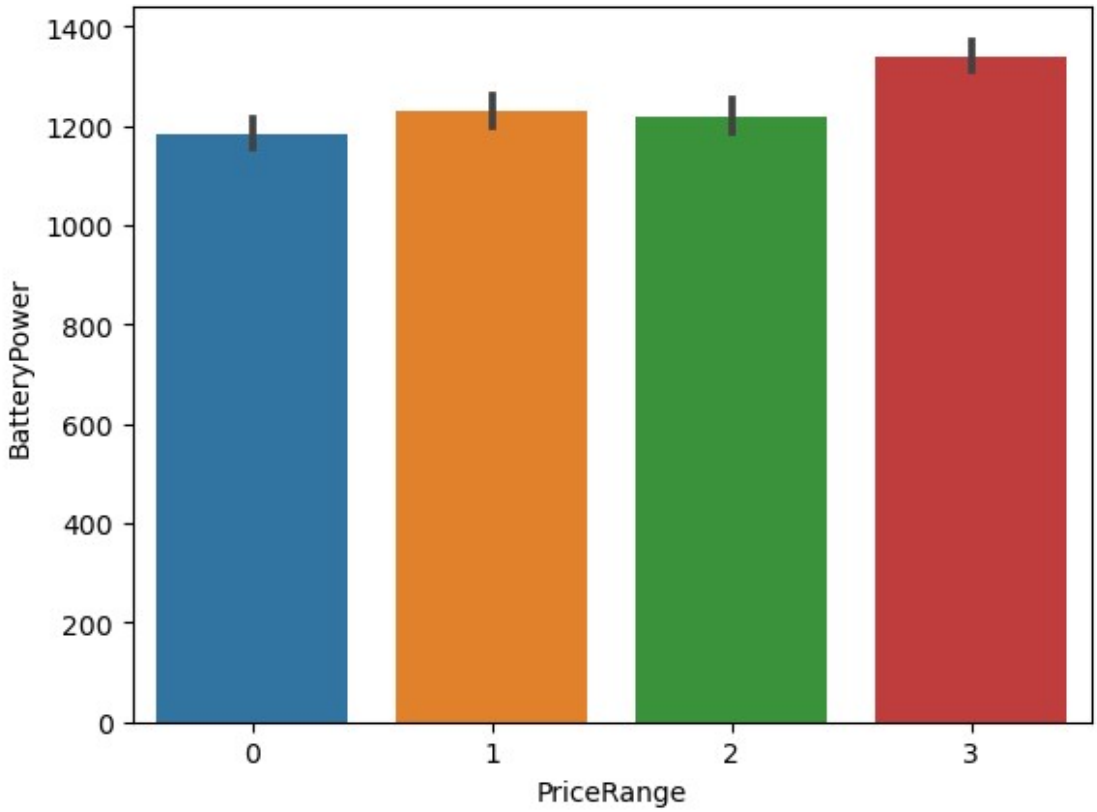
```
dtypes: float64(2), int64(19)
memory usage: 492.3 KB
```

Phase 2

```
df.hist(figsize=(15,10))
plt.show()
```

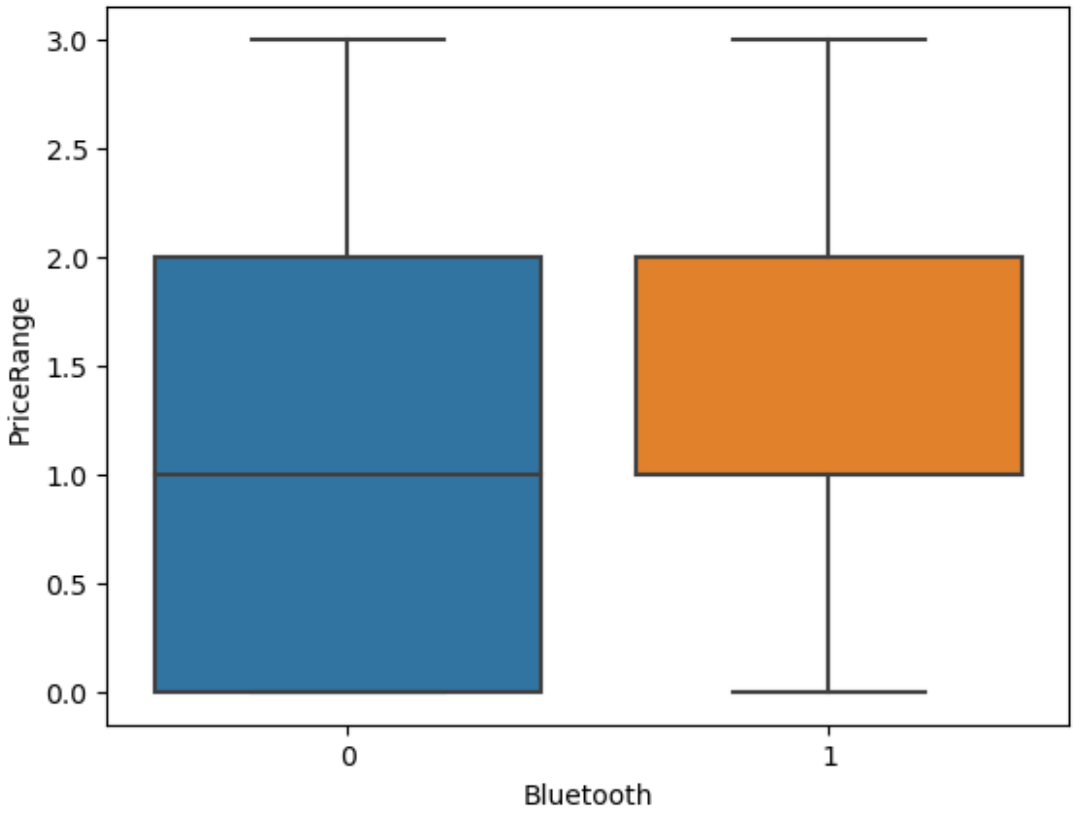



```
sns.barplot(x='PriceRange', y='BatteryPower', data=df)
plt.show()
```



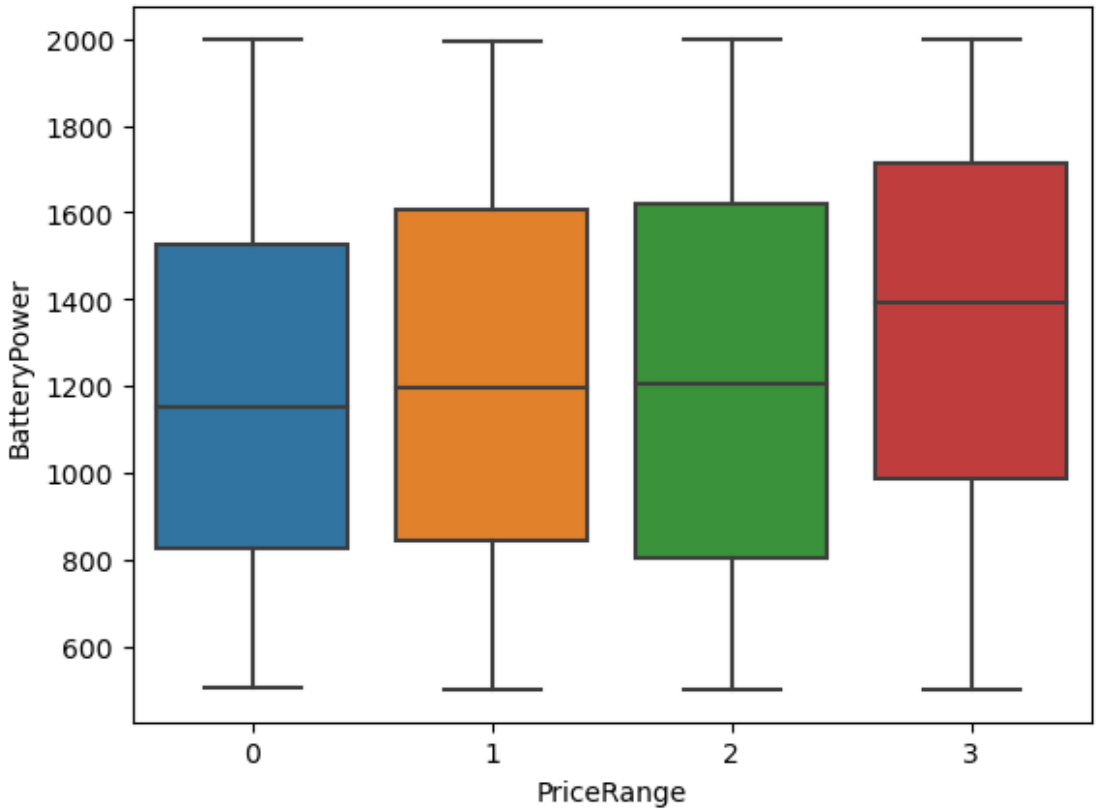

```
sns.boxplot(x="Bluetooth", y="PriceRange", data=df)
```

```
<AxesSubplot:xlabel='Bluetooth', ylabel='PriceRange'>
```



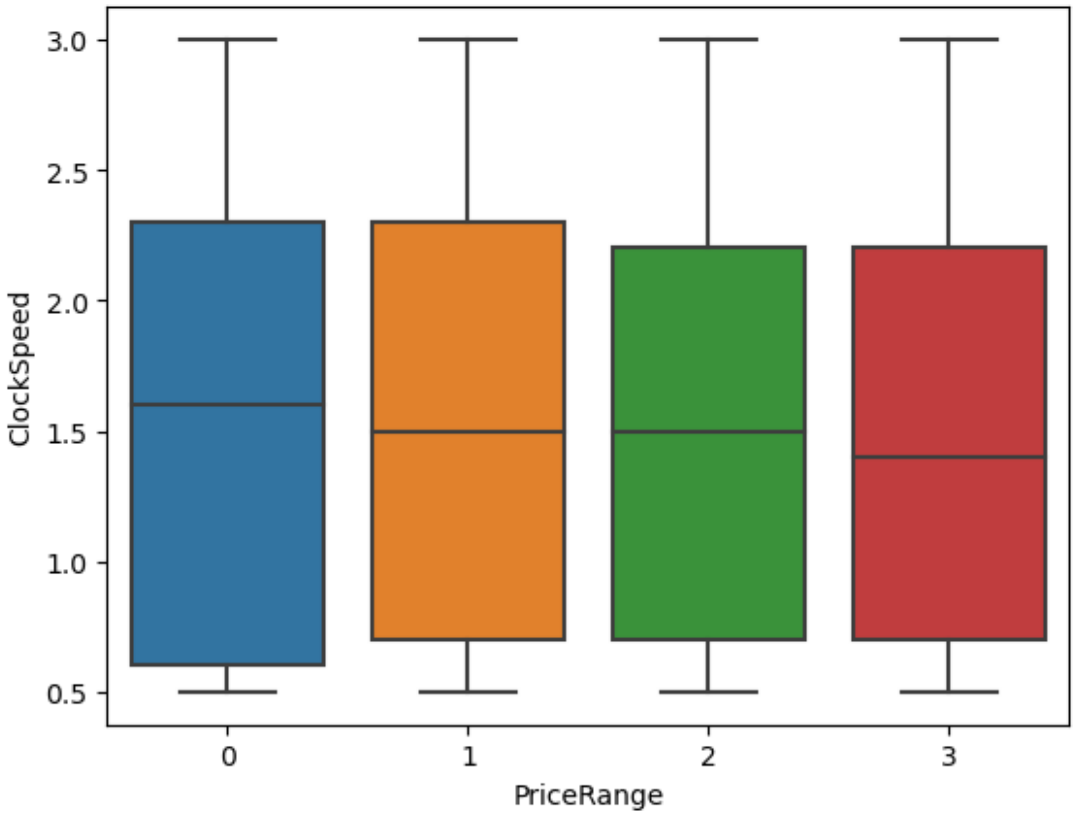
so the phones having bluetooth start with price range of 1

```
sns.boxplot(x="PriceRange", y="BatteryPower", data=df)
plt.show()
```

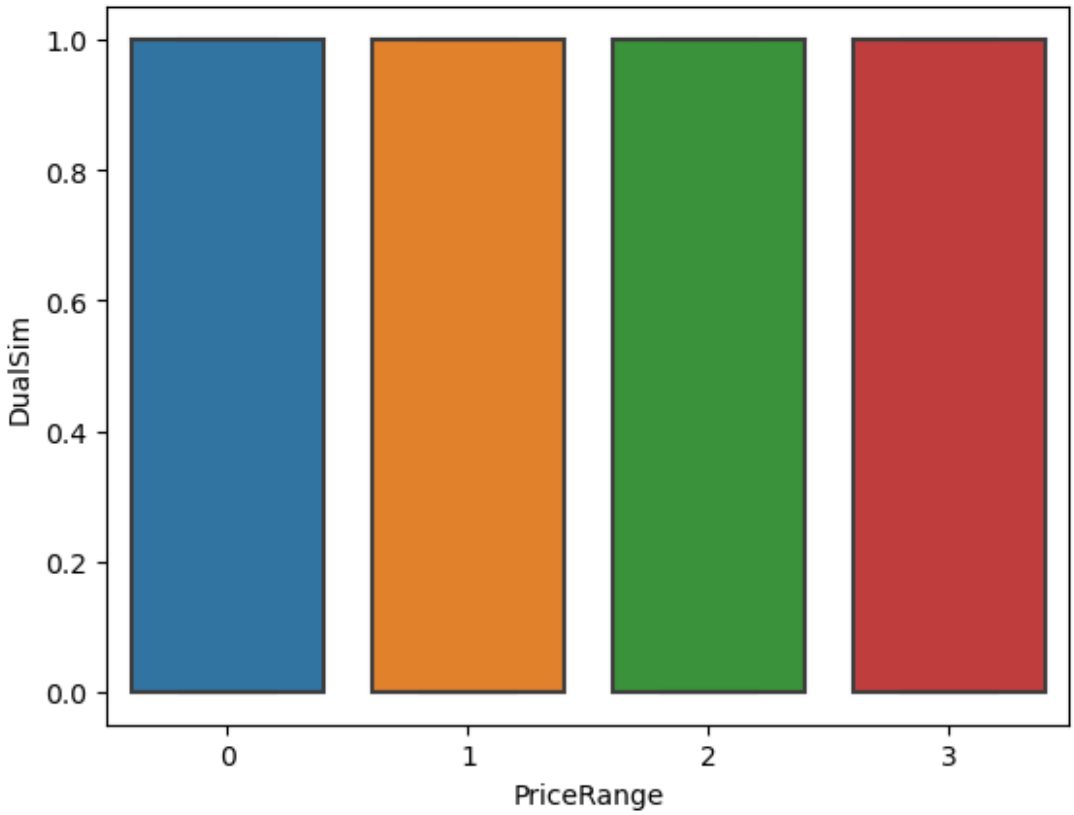


phones with high price range have more battery power

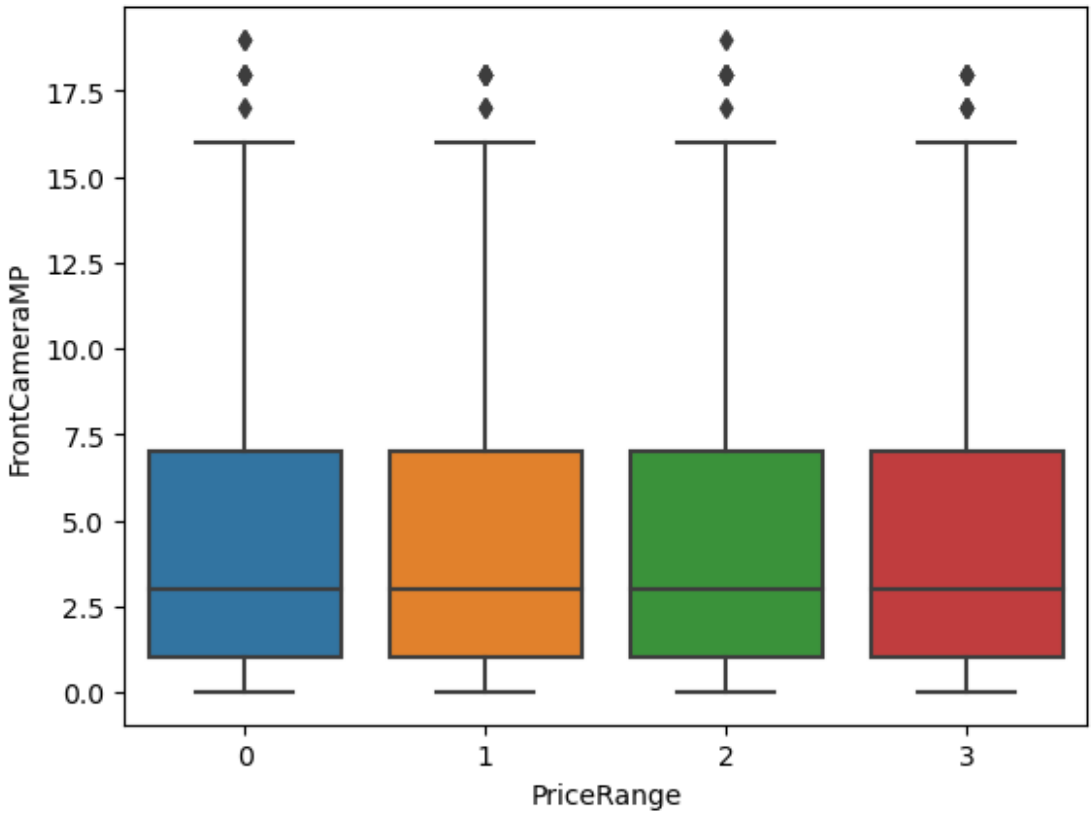
```
sns.boxplot(x='PriceRange', y='ClockSpeed', data=df)
plt.show()
```



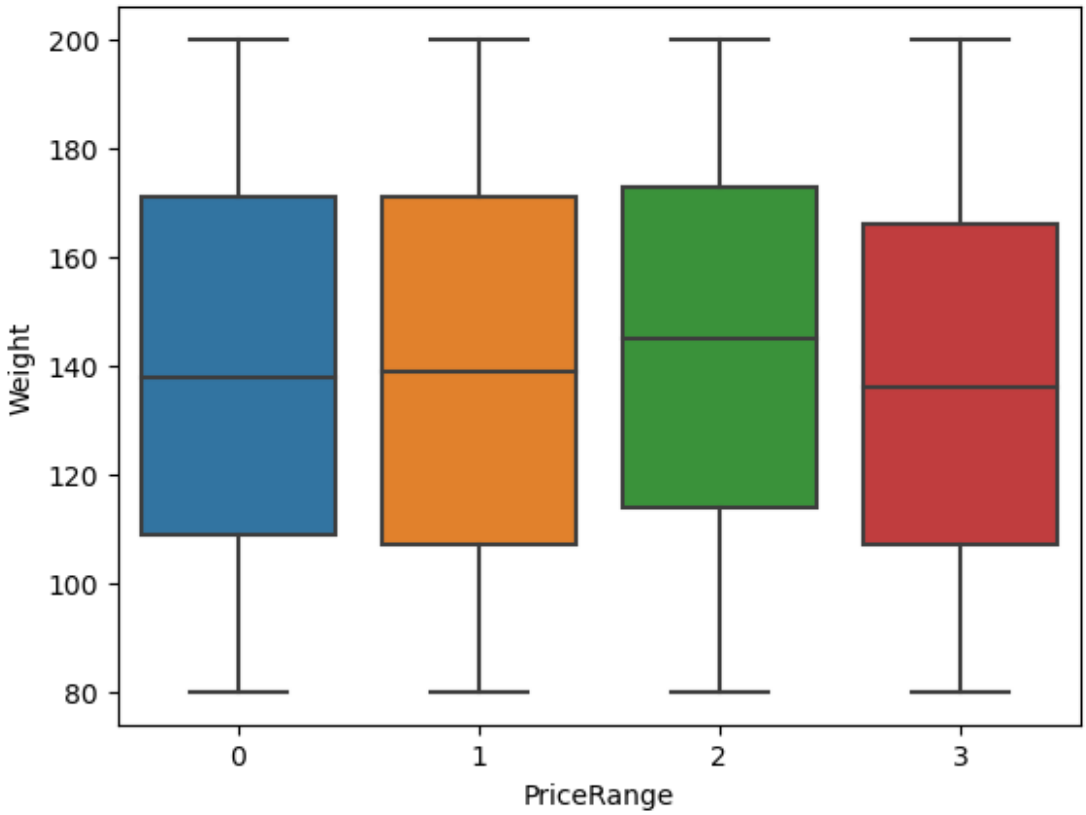

```
sns.boxplot(x='PriceRange', y='DualSim', data=df)
plt.show()
```




```
sns.boxplot(x='PriceRange', y='FrontCameraMP', data=df)
plt.show()
```

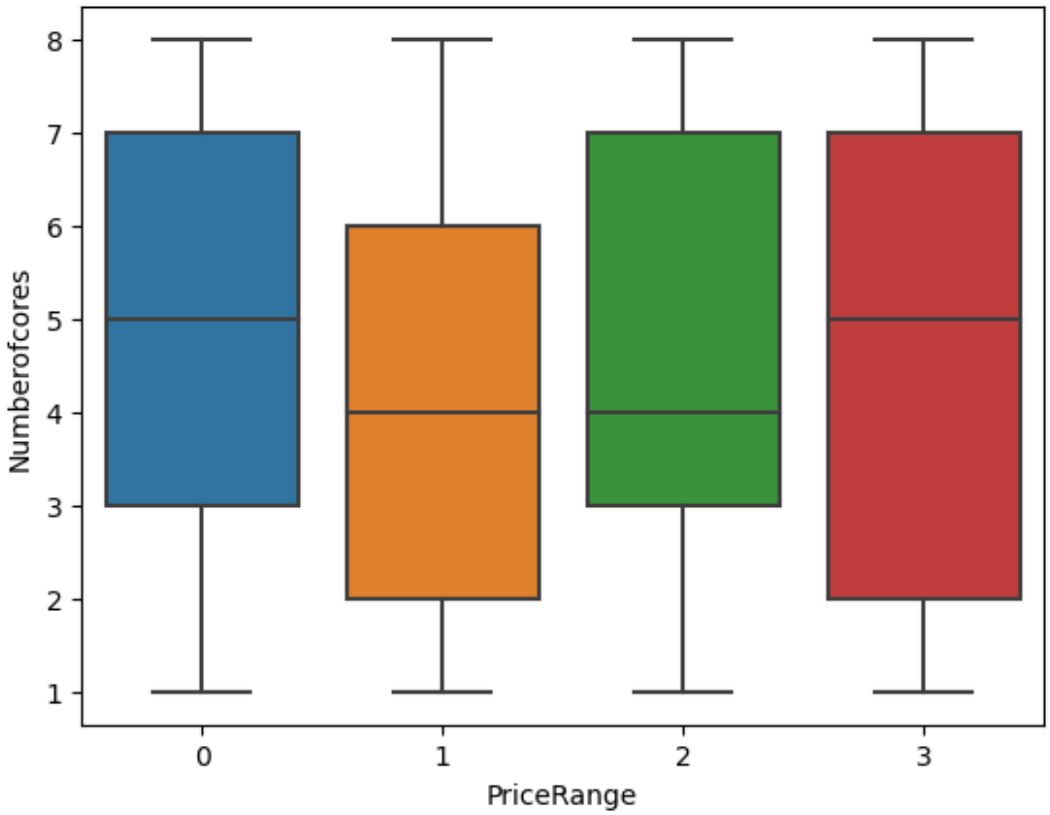



```
sns.boxplot(x='PriceRange', y='Weight', data=df)
plt.show()
```



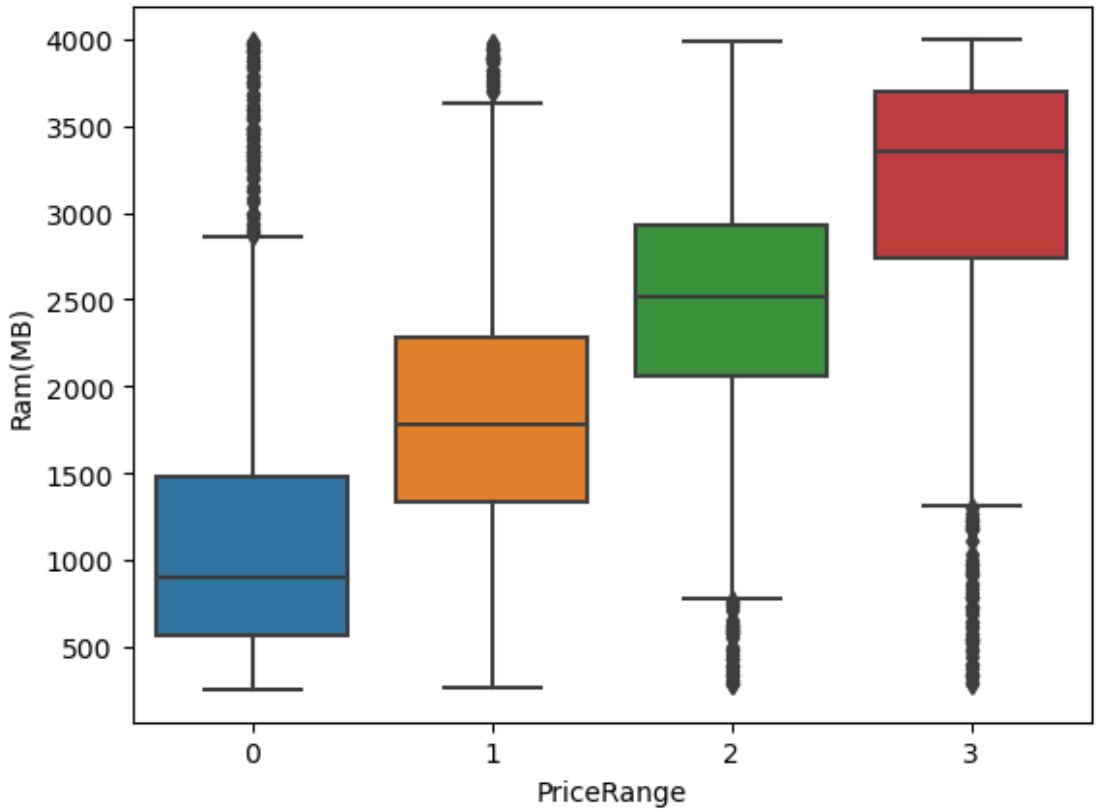
so higher the price range light weight is the phone also with price range 1 avg is the weight

```
sns.boxplot(x='PriceRange', y='Numberofcores', data=df)
plt.show()
```



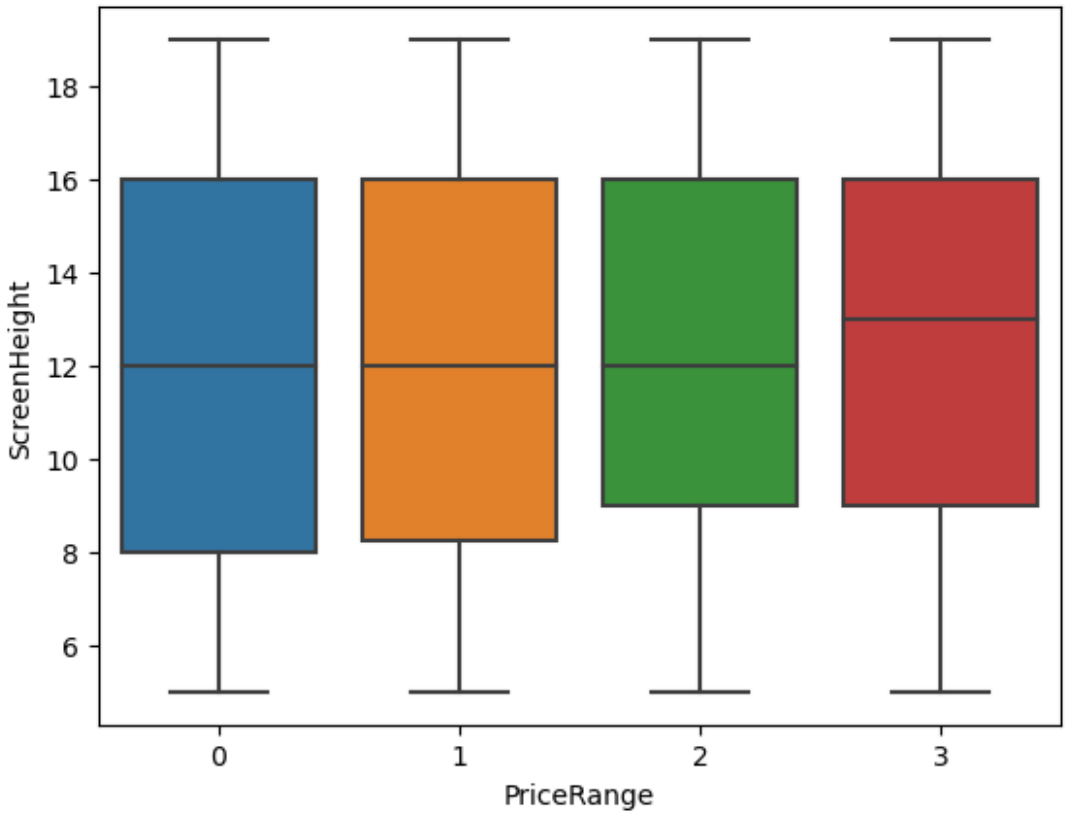
phones with price range of 1 have less number of cores

```
sns.boxplot(x='PriceRange', y='Ram(MB)', data=df)
plt.show()
```

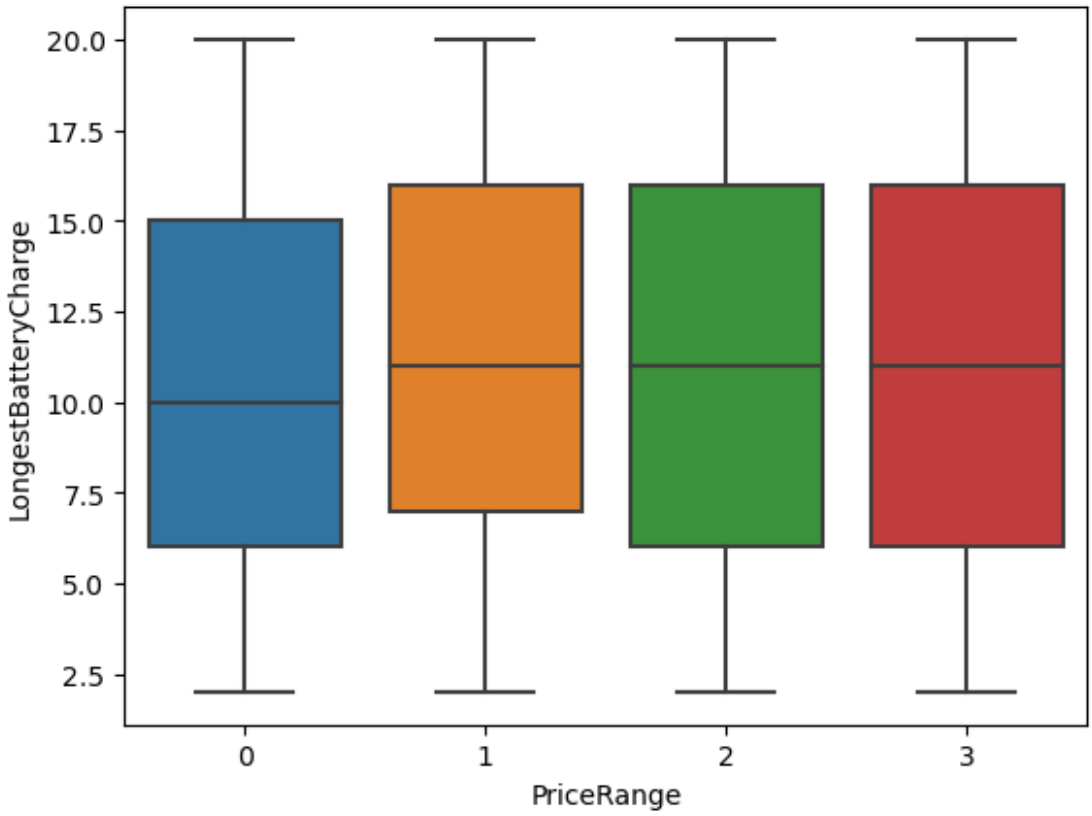


phones with price range 2-3 have good ram and price range with 1 have avg ram

```
sns.boxplot(x='PriceRange', y='ScreenHeight', data=df)
plt.show()
```




```
sns.boxplot(x='PriceRange', y='LongestBatteryCharge', data=df)
plt.show()
```



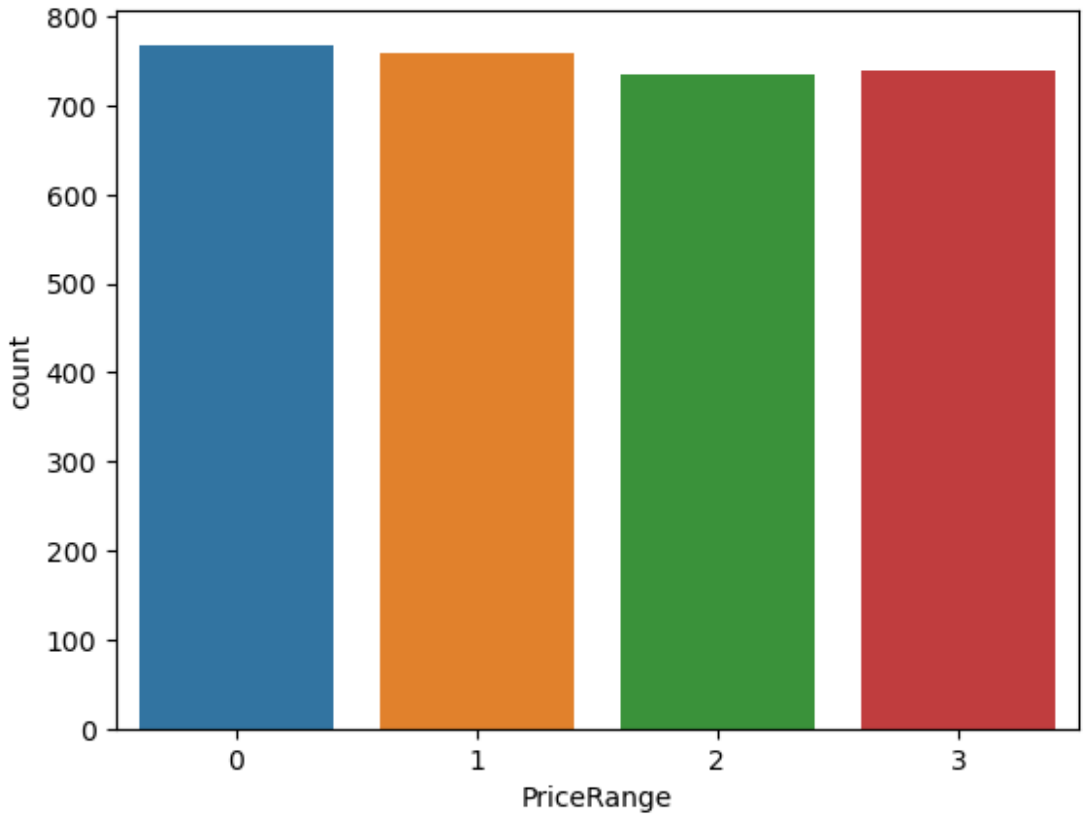
phones with price range of 2-3 have good battery life

```
df.head(1)
```

```
   BatteryPower  Bluetooth  ClockSpeed  DualSim  FrontCameraMP  4G  \
0           500          0         1.3        1             13   0

   InternalMemory  MobileDepth  Weight  Numberofcores  ...
PixelR.Height  \
0              64          0.9     134              4  ...
1384

   PixelR.Width  Ram(MB)  ScreenHeight  ScreenWidth
LongestBatteryCharge  3G  \
0          1912     2807            19            7
7   0

   TouchScreen  WIFI  PriceRange
0            0     1           1

[1 rows x 21 columns]
```

```
# distribution of variables
```

```
sns.countplot(x='PriceRange', data=df)
plt.show()
```



Phase 3

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
```

```
X = df.drop('PriceRange', axis=1)
y = df['PriceRange']
```

```
X.head(1)
```

```
   BatteryPower  Bluetooth  ClockSpeed  DualSim  FrontCameraMP  4G  \
0           500          0         1.3        1             13   0

   InternalMemory  MobileDepth  Weight  Numberofcores  PrimaryCameraMP
\
0              64          0.9     134              4               17
```

```
   PixelR.Height  PixelR.Width  Ram(MB)  ScreenHeight  ScreenWidth  \
0           1384          1912     2807            19            7

   LongestBatteryCharge  3G  TouchScreen  WIFI
0                     7   0            0     1
```

```
X_train , X_test ,y_train , y_test =
train_test_split(X,y,test_size=0.3,random_state=51)
```

```
X_train.shape
```

```
(2100, 20)
```

```
y_train.shape
```

```
(2100,)
```

```
X_test.shape
```

```
(900, 20)
```

```
# desclaring all the models

logreg = LogisticRegression()
knn = KNeighborsClassifier()
svc = SVC()
dt = DecisionTreeClassifier(random_state=51)
rf = RandomForestClassifier(random_state=51)


logreg.fit(X_train, y_train)
knn.fit(X_train, y_train)
svc.fit(X_train, y_train)
dt.fit(X_train, y_train)
rf.fit(X_train, y_train)
```

```
RandomForestClassifier(random_state=51)
```

```
logreg_pred = logreg.predict(X_test)
knn_pred = knn.predict(X_test)
svc_pred = svc.predict(X_test)
dt_pred = dt.predict(X_test)
rf_pred = rf.predict(X_test)

logreg_acc = accuracy_score(y_test, logreg_pred)
knn_acc = accuracy_score(y_test, knn_pred)
svc_acc = accuracy_score(y_test, svc_pred)
dt_acc = accuracy_score(y_test, dt_pred)
rf_acc = accuracy_score(y_test, rf_pred)

print(logreg_acc)
```

```
0.5
```

```
print(knn_acc)
```

```
0.6477777777777778
```

```
print(svc_acc)
```

```
0.7077777777777777
```

```
print(dt_acc)
```

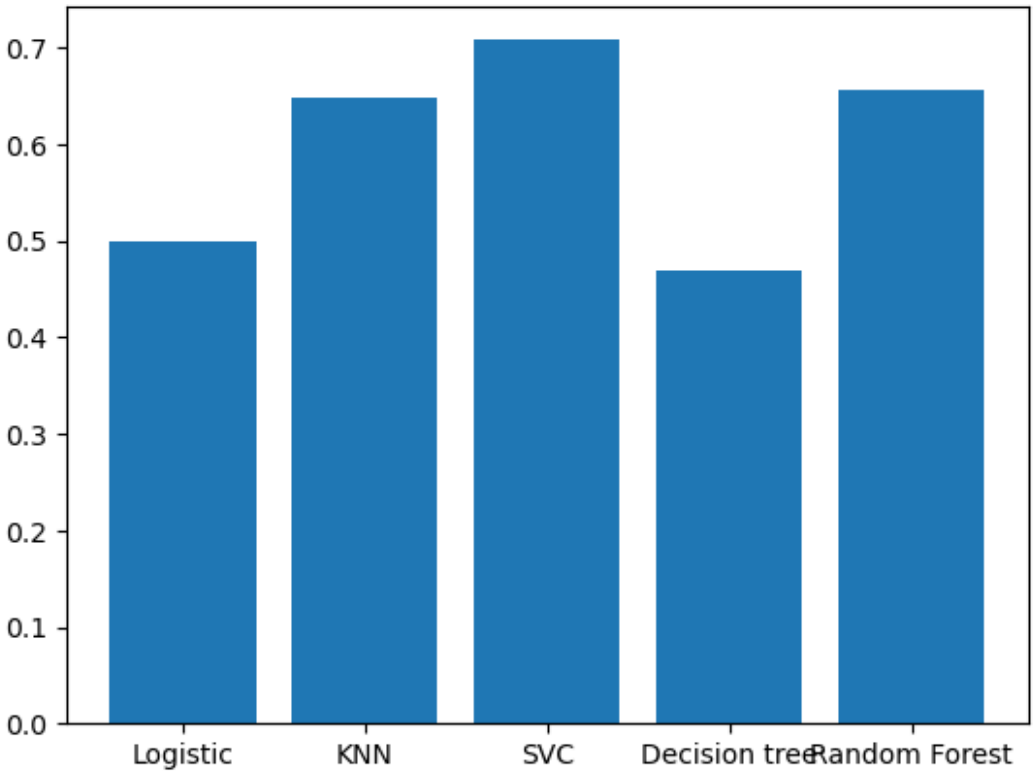
```
0.47
```

```
print(rf_acc)
```

```
0.6555555555555556
```

```
models = ['Logistic','KNN','SVC','Decision tree','Random Forest']
accuracy = [logreg_acc,knn_acc,svc_acc,dt_acc,rf_acc]
plt.bar(models,accuracy)

best_model = models[accuracy.index(max(accuracy))]
print(f'The best model is {best_model} with an accuracy score of
{max(accuracy)}.')
```

```
The best model is SVC with an accuracy score of 0.7077777777777777.
```



Phase 4

```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

param_grid = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf'],
    'gamma': ['scale', 'auto']
}

svc = SVC(random_state=51)

grid_search = GridSearchCV(estimator=svc, param_grid=param_grid, cv=5,
n_jobs=-1)
grid_search.fit(X_train, y_train)


print("Best parameters:", grid_search.best_params_)
print("Best score:", grid_search.best_score_)
```

```
Best parameters: {'C': 10, 'gamma': 'scale', 'kernel': 'rbf'}
Best score: 0.7138095238095239
```

```
new_data = [[1800, 1, 1.0, 0, 5, 3, 5, 0.3, 64, 4, 5, 400, 1280, 1280,
0, 5, 3, 5, 1, 0,1,0]]
x_d = pd.DataFrame(new_data)
X_subset = x_d.iloc[:, :20]
predicted_price_range = grid_search.best_estimator_.predict(X_subset)
print("Predicted price range:", predicted_price_range)
```

```
Predicted price range: [1]
```

```
#?????????????

df.head(1)
```

```
   BatteryPower  Bluetooth  ClockSpeed  DualSim  FrontCameraMP  4G  \
0           500          0         1.3        1             13   0

   InternalMemory  MobileDepth  Weight  Numberofcores  ...
PixelR.Height  \
0              64          0.9     134              4  ...
1384

   PixelR.Width  Ram(MB)  ScreenHeight  ScreenWidth
LongestBatteryCharge  3G  \
0          1912     2807            19            7
7   0

   TouchScreen  WIFI  PriceRange
0            0     1           1
```

```
[1 rows x 21 columns]

# Take user input for each feature
BatteryPower = int(input("Enter Battery Power (in mAh): "))
Bluetooth = int(input("Does the phone have Bluetooth? (0 for No, 1 for
Yes): "))
ClockSpeed = float(input("Enter Clock Speed (in GHz) value in decimal:
"))
DualSim = int(input("Does the phone have Dual SIM? (0 for No, 1 for
Yes): "))
FrontCameraMP = float(input("Enter Front Camera Resolution (in
Megapixels): "))
FourG = int(input("Does the phone have 4G? (0 for No, 1 for Yes): "))
InternalMemory = int(input("Enter Internal Memory (in GB): "))
MobileDepth = float(input("Enter Mobile Depth (in cm) in decimal: "))
Weight = float(input("Enter Mobile Weight (in grams): "))
Numberofcores = int(input("Enter Number of Cores: "))
PrimaryCameraMP = float(input("Enter Primary Camera Resolution (in
Megapixels): "))
PixelR_Height = int(input("Enter Pixel Resolution Height in px: "))
PixelR_Width = int(input("Enter Pixel Resolution Width in px: "))
Ram = int(input("Enter RAM (in MB): "))
ScreenHeight = int(input("Enter Screen Height (in cm): "))
ScreenWidth = int(input("Enter Screen Width (in cm): "))
LongestBatteryCharge = int(input("Enter Longest Battery Charge (in
hours): "))
ThreeG = int(input("Does the phone have 3G? (0 for No, 1 for Yes): "))
TouchScreen = int(input("Does the phone have Touch Screen? (0 for No,
1 for Yes): "))
WiFi = int(input("Does the phone have WiFi? (0 for No, 1 for Yes): "))

# Make a prediction based on the user input
new_data = [[BatteryPower, Bluetooth, ClockSpeed, DualSim,
FrontCameraMP, FourG, InternalMemory, MobileDepth,
             Weight, Numberofcores, PrimaryCameraMP, PixelR_Height,
PixelR_Width, Ram, ScreenHeight, ScreenWidth,
             LongestBatteryCharge, ThreeG, TouchScreen, WiFi]]

x_d = pd.DataFrame(new_data)
X_subset = x_d.iloc[:, :20]
predicted_price_range = grid_search.best_estimator_.predict(X_subset)


print("Predicted price range:", predicted_price_range)

Enter Battery Power (in mAh): 3500
Does the phone have Bluetooth? (0 for No, 1 for Yes): 1
Enter Clock Speed (in GHz) value in decimal: 2.1
Does the phone have Dual SIM? (0 for No, 1 for Yes): 1
```

```
Enter Front Camera Resolution (in Megapixels): 5
Does the phone have 4G? (0 for No, 1 for Yes): 1
Enter Internal Memory (in GB): 64
Enter Mobile Depth (in cm) in decimal: 7.7
Enter Mobile Weight (in grams): 174
Enter Number of Cores: 8
Enter Primary Camera Resolution (in Megapixels): 12
Enter Pixel Resolution Height in px: 1125
Enter Pixel Resolution Width in px: 2436
Enter RAM (in MB): 3
Enter Screen Height (in cm): 13
Enter Screen Width (in cm): 6
Enter Longest Battery Charge (in hours): 7
Does the phone have 3G? (0 for No, 1 for Yes): 1
Does the phone have Touch Screen? (0 for No, 1 for Yes): 1
Does the phone have WiFi? (0 for No, 1 for Yes): 1
Predicted price range: [0]
```