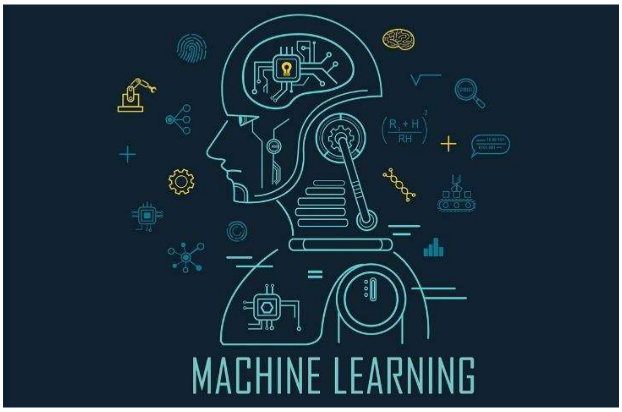


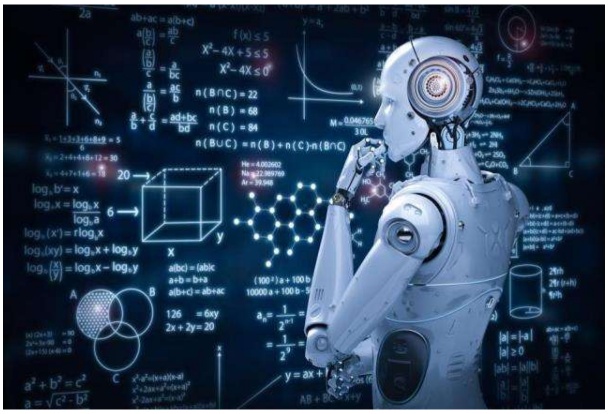


TABLE OF CONTENTS

Mobile Price Classification

ROLE OF EACH STUDENT :

1). Narendra Rathod - Analysis of Dataset, Splitting of Train, Model Training using SVM Linear Kernel Classifier, Model Training using Gradient Boosting Classifier, Bias and Variance Calculations, Confusion Matrix.

2). Nayan Kumar - Test dataset, Feature Scaling, Model Training using KNN Classifier, Model Training XGBClassifier, Model Training using Ada Boost Classifier , Theory and Accuracy.

DATA SET

This dataset contain information about many mobiles and variables about it.

DATA SOURCE LINK - CLICK ON THE LINK

DESCRIPTION OF PROJECT:

Price is the most effective attribute of marketing and business. The very first question of costumer is about the price of items. All the costumers are first worried and thinks "If he would be able to purchase something with given specifications or not".

The price of a product is the most important attribute of marketing that product. One of those products where price matters a lot is a smartphone because it comes with a lot of features so that a company thinks a lot about how to price this mobile which can justify the features and also cover the marketing and manufacturing costs of the mobile. In this article, I will walk you through the task of mobile price classification with Machine Learning using Python.

PROBLEM STATEMENT:

Narendra 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.

Narendra 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.

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

OBJECTIVE:

The objective of the data source is to classify activities into one of the three activities performed*.*
The three activities performed were as follows:

- Cleaning Data and Feature Selection
- Training and Evaluate Model
- Save Model for Future Use

APPROACH ADOPTED:

With the given data set to get better idea of mobile price, we adopted various classifications/clustering algorithms and try to find out which classifier gives the best accuracy/score according to given data.

PROGRAM SOURCE CODE:

```
 import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV
import warnings
warnings.filterwarnings('ignore')
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

df=pd.read_csv("train.csv")
df.head(10)
```

```
   battery_power  blue  clock_speed  dual_sim  fc  four_g  int_memory
m_dep  \
0            842     0          2.2         0   1       0           7
0.6
1           1021     1          0.5         1   0       1          53
0.7
2            563     1          0.5         1   2       1          41
0.9
3            615     1          2.5         0   0       0          10
0.8
4           1821     1          1.2         0  13       1          44
0.6
5           1859     0          0.5         1   3       0          22
0.7
6           1821     0          1.7         0   4       1          10
0.8
7           1954     0          0.5         1   0       0          24
0.8
8           1445     1          0.5         0   0       0          53
0.7
9            509     1          0.6         1   2       1           9
0.1

   mobile_wt  n_cores  ...  px_height  px_width   ram  sc_h  sc_w
talk_time  \
0        188        2  ...         20       756  2549     9     7
19
```

```
1        136         3  ...        905       1988  2631    17      3
7
2        145         5  ...       1263       1716  2603    11      2
9
3        131         6  ...       1216       1786  2769    16      8
11
4        141         2  ...       1208       1212  1411     8      2
15
5        164         1  ...       1004       1654  1067    17      1
10
6        139         8  ...        381       1018  3220    13      8
18
7        187         4  ...        512       1149   700    16      3
5
8        174         7  ...        386        836  1099    17      1
20
9         93         5  ...       1137       1224   513    19     10
12

   three_g  touch_screen  wifi  price_range
0        0             0     1            1
1        1             1     0            2
2        1             1     0            2
3        1             0     0            2
4        1             1     0            1
5        1             0     0            1
6        1             0     1            3
7        1             1     1            0
8        1             0     0            0
9        1             0     0            0

[10 rows x 21 columns]
```

```
df.shape
```

```
(2000, 21)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   battery_power  2000 non-null   int64  
 1   blue           2000 non-null   int64  
 2   clock_speed    2000 non-null   float64
 3   dual_sim       2000 non-null   int64  
 4   fc             2000 non-null   int64  
 5   four_g         2000 non-null   int64  
 6   int_memory     2000 non-null   int64  
```

```
 7   m_dep           2000 non-null   float64
 8   mobile_wt       2000 non-null   int64
 9   n_cores         2000 non-null   int64
 10  pc              2000 non-null   int64
 11  px_height       2000 non-null   int64
 12  px_width        2000 non-null   int64
 13  ram             2000 non-null   int64
 14  sc_h            2000 non-null   int64
 15  sc_w            2000 non-null   int64
 16  talk_time       2000 non-null   int64
 17  three_g         2000 non-null   int64
 18  touch_screen    2000 non-null   int64
 19  wifi            2000 non-null   int64
 20  price_range     2000 non-null   int64
dtypes: float64(2), int64(19)
memory usage: 328.2 KB
```

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

```
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
```

```
df.describe()
```

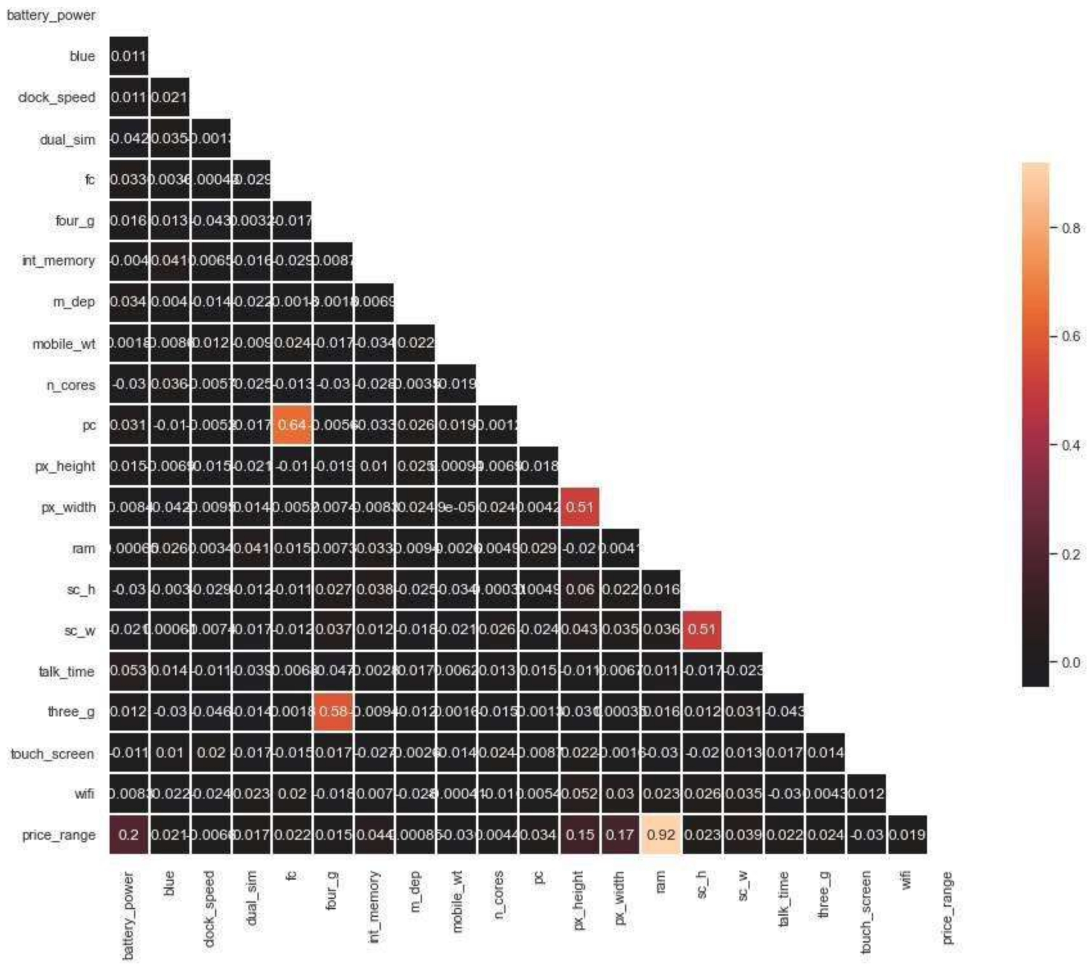
```
       battery_power       blue  clock_speed     dual_sim           fc  
\
count    2000.000000  2000.0000  2000.000000  2000.000000  2000.000000

mean     1238.518500     0.4950     1.522250     0.509500     4.309500

std       439.418206     0.5001     0.816004     0.500035     4.341444
```

```
min       501.000000     0.0000     0.500000     0.000000     0.000000
25%       851.750000     0.0000     0.700000     0.000000     1.000000
50%      1226.000000     0.0000     1.500000     1.000000     3.000000
75%      1615.250000     1.0000     2.200000     1.000000     7.000000
max      1998.000000     1.0000     3.000000     1.000000    19.000000

            four_g   int_memory        m_dep    mobile_wt      n_cores
...  \
count  2000.000000  2000.000000  2000.000000  2000.000000  2000.000000
...
mean      0.521500    32.046500     0.501750   140.249000     4.520500
...
std       0.499662    18.145715     0.288416    35.399655     2.287837
...
min       0.000000     2.000000     0.100000    80.000000     1.000000
...
25%       0.000000    16.000000     0.200000   109.000000     3.000000
...
50%       1.000000    32.000000     0.500000   141.000000     4.000000
...
75%       1.000000    48.000000     0.800000   170.000000     7.000000
...
max       1.000000    64.000000     1.000000   200.000000     8.000000
...

         px_height     px_width          ram         sc_h         sc_w
\
count  2000.000000  2000.000000  2000.000000  2000.000000  2000.000000
mean    645.108000  1251.515500  2124.213000    12.306500     5.767000
std     443.780811   432.199447  1084.732044     4.213245     4.356398
min       0.000000   500.000000   256.000000     5.000000     0.000000
25%     282.750000   874.750000  1207.500000     9.000000     2.000000
50%     564.000000  1247.000000  2146.500000    12.000000     5.000000
75%     947.250000  1633.000000  3064.500000    16.000000     9.000000
max    1960.000000  1998.000000  3998.000000    19.000000    18.000000
```

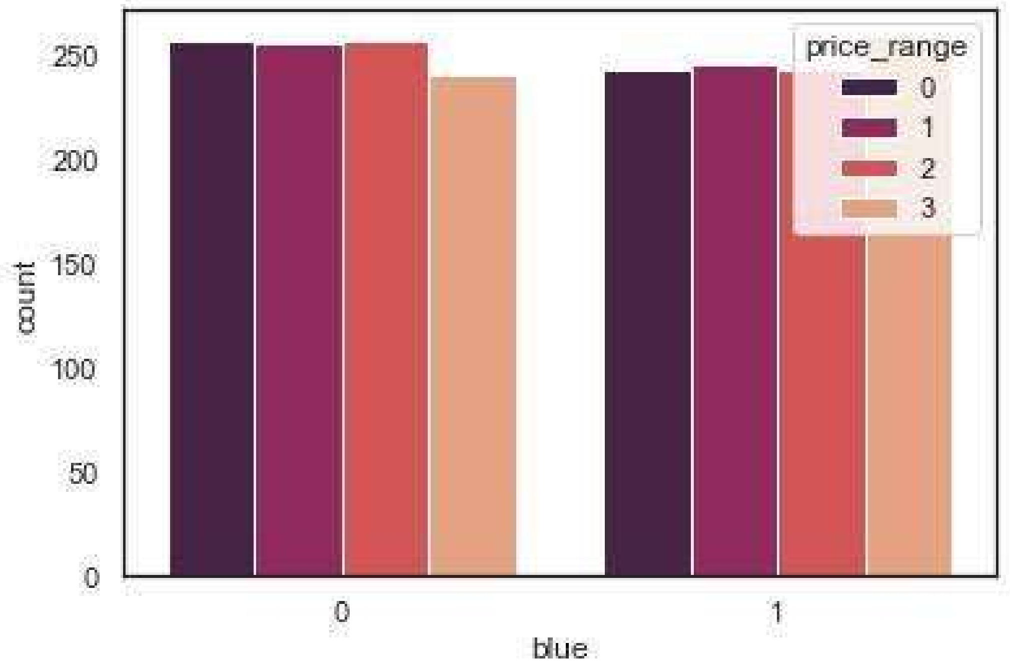
```
        talk_time     three_g  touch_screen         wifi
price_range
count  2000.000000  2000.000000   2000.000000  2000.000000
2000.000000
mean     11.011000     0.761500      0.503000     0.507000
1.500000
std       5.463955     0.426273      0.500116     0.500076
1.118314
min       2.000000     0.000000      0.000000     0.000000
0.000000
25%       6.000000     1.000000      0.000000     0.000000
0.750000
50%      11.000000     1.000000      1.000000     1.000000
1.500000
75%      16.000000     1.000000      1.000000     1.000000
2.250000
max      20.000000     1.000000      1.000000     1.000000
3.000000

[8 rows x 21 columns]
```

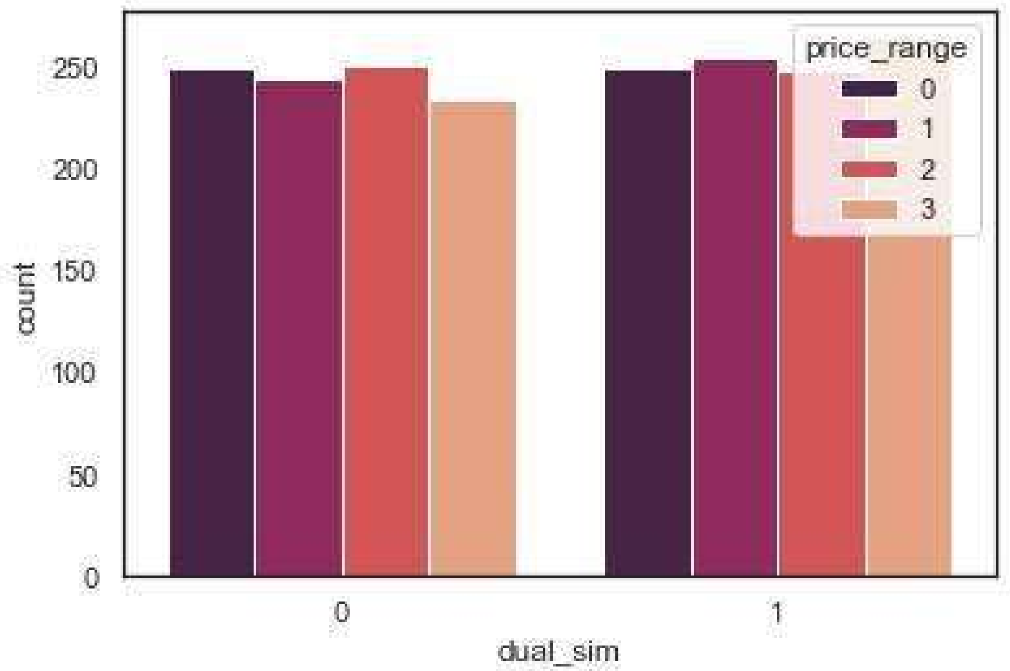
```
sns.set_theme(style="white")

corr = df.corr()
mask = np.triu(np.ones_like(corr, dtype=bool))

f, ax = plt.subplots(figsize=(15, 15))
cmap = sns.diverging_palette(230, 20)

sns.heatmap(corr, mask=mask, center=0, annot=True,
            square=True, linewidths=.3,cbar_kws={"shrink": 0.5})
```

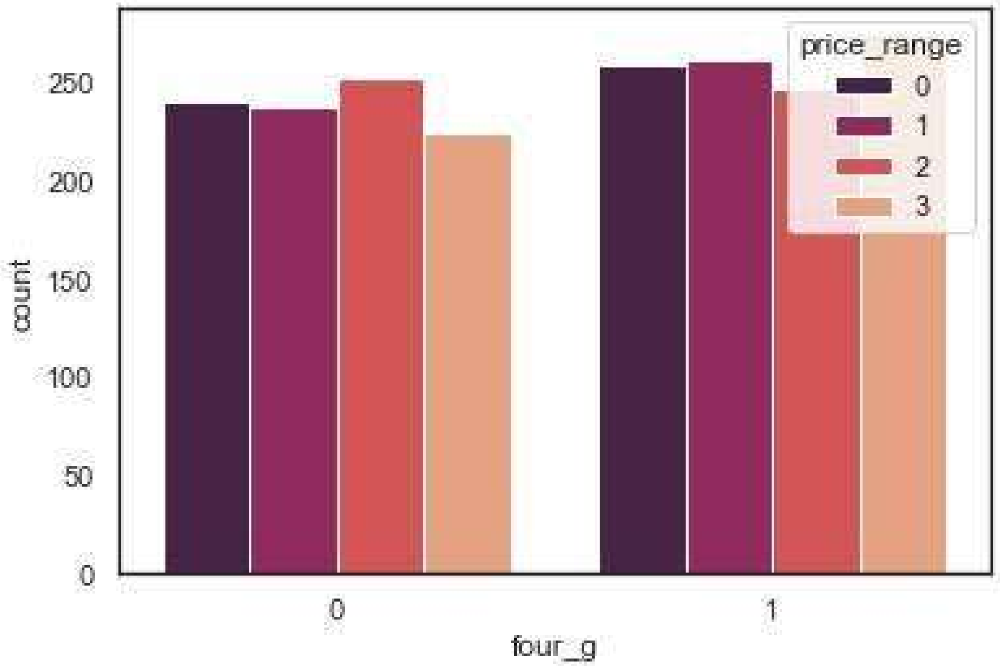
```
<AxesSubplot:>
```

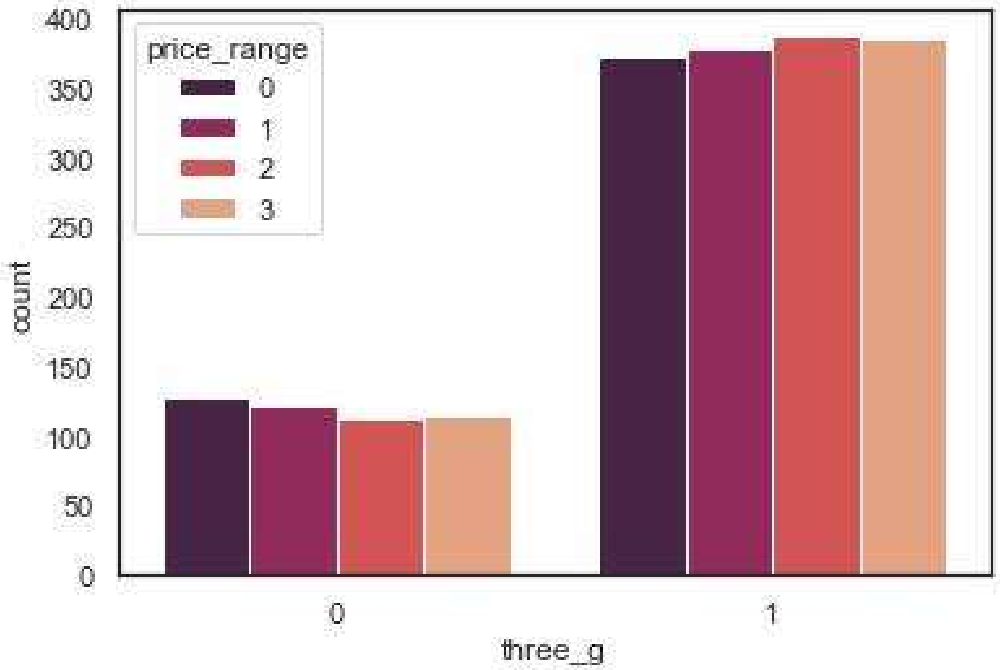


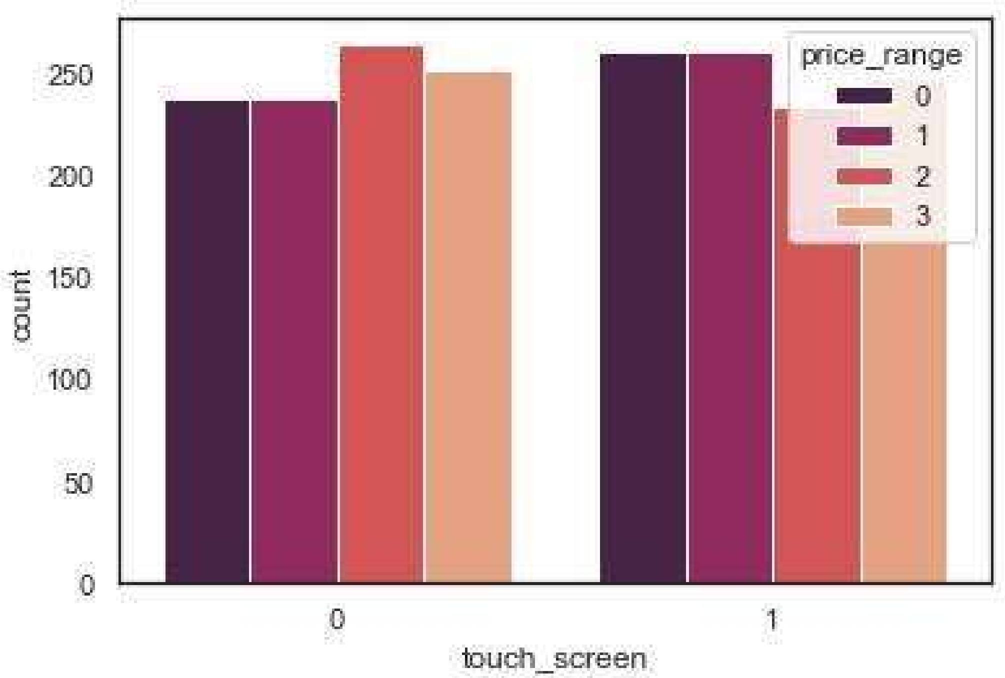
```
binary_col =
['blue','dual_sim','four_g','three_g','touch_screen','wifi
'] for i in binary_col:
    ax = sns.countplot(x=i,
hue='price_range',data=df,palette='rocket')
 plt.show()
```

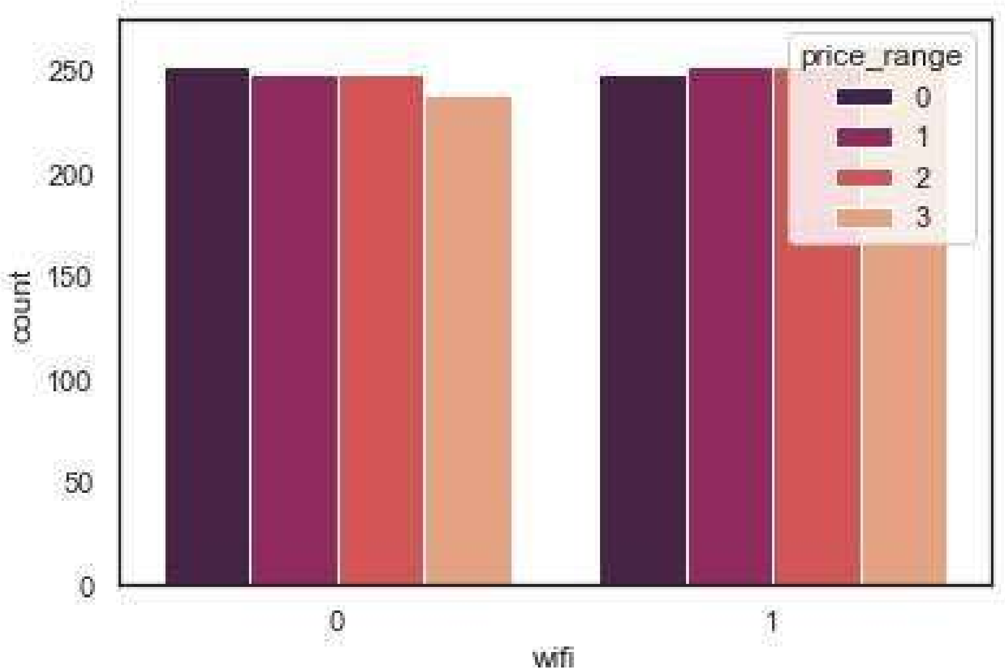

price_range
0
1
2
3
250
200
150
100
50
0
count
0
1
blue


price_range
0
1
2
3
250
200
150
100
50
0
count
0
1
dual_sim

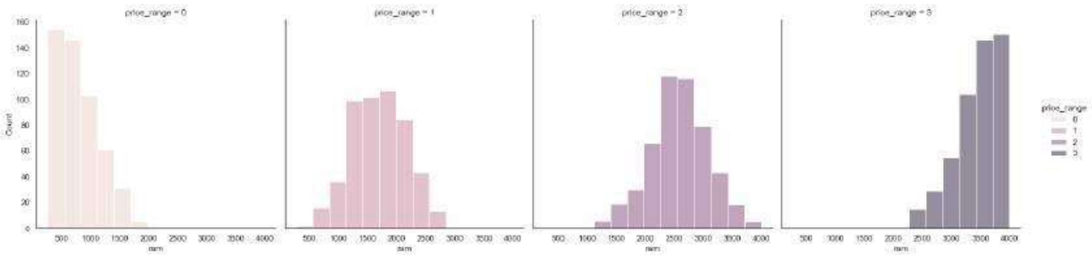

price_range
0
1
2
3
count
250
200
150
100
50
0
0
1
four_g


price_range
0
1
2
3
count
400
350
300
250
200
150
100
50
0
0
1
three_g

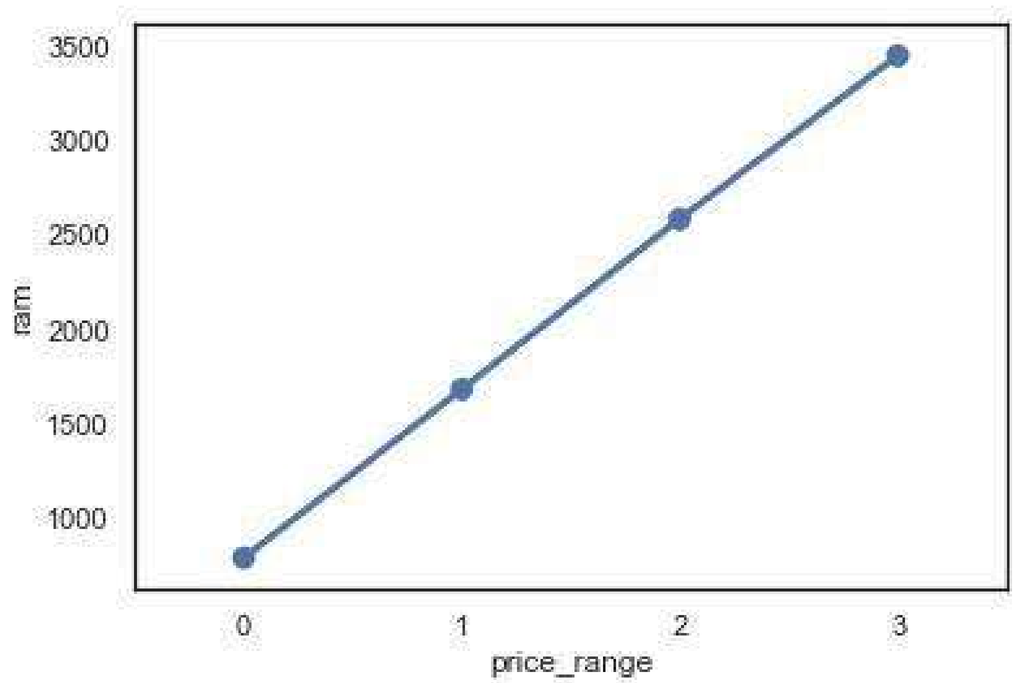




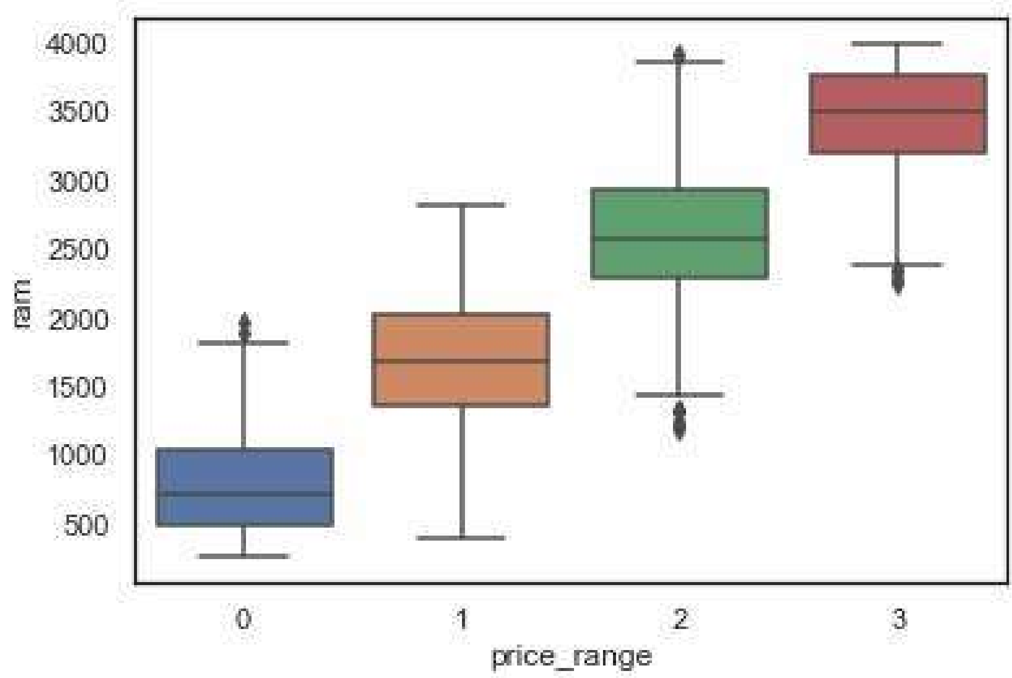

```
sns.displot(data=df, x="ram", hue="price_range", col="price_range")
plt.show()
```



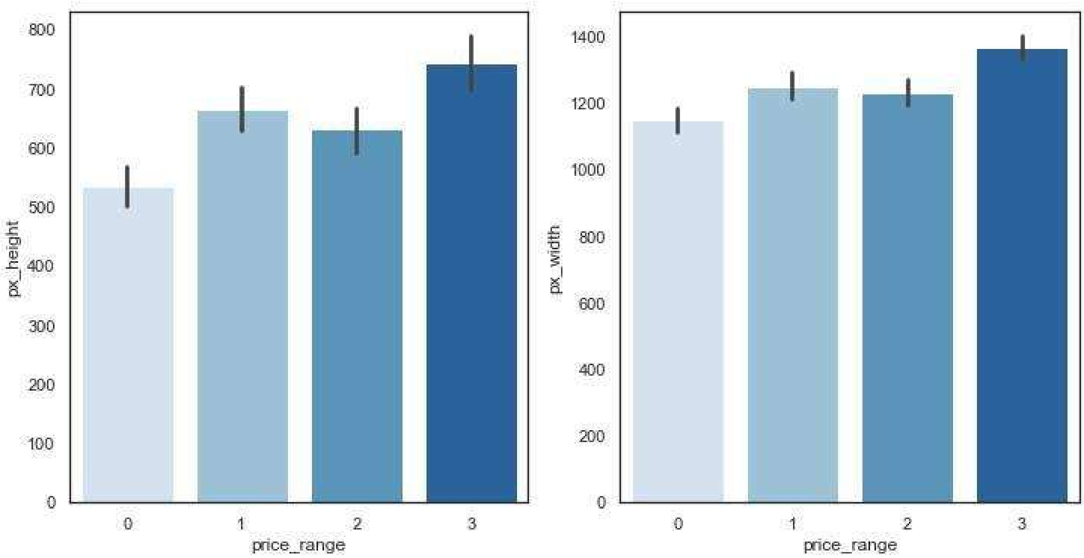
```
sns.pointplot(x='price_range',y='ram',data=df)
plt.show()
```




```
sns.boxplot(x='price_range',y='ram',data=df)
plt.show()
```

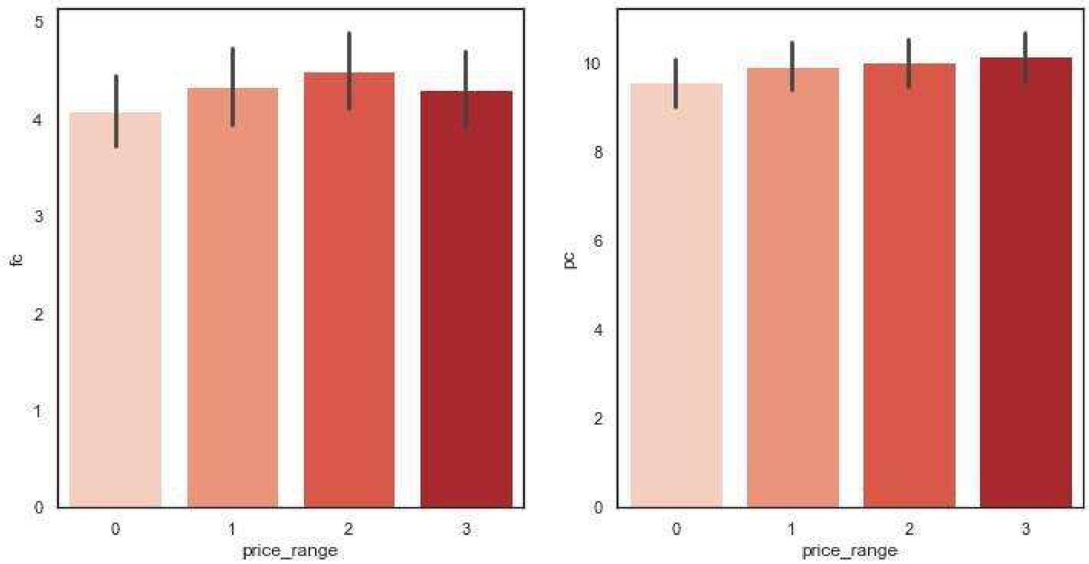



```
plt.figure(figsize=(12,6)) plt.subplot(1,2,1)
sns.barplot(x='price_range',y='px_height',data=df,palette='Blues')
plt.subplot(1,2,2)
sns.barplot(x='price_range',y='px_width',data=df,palette='Blues')
plt.show()
```

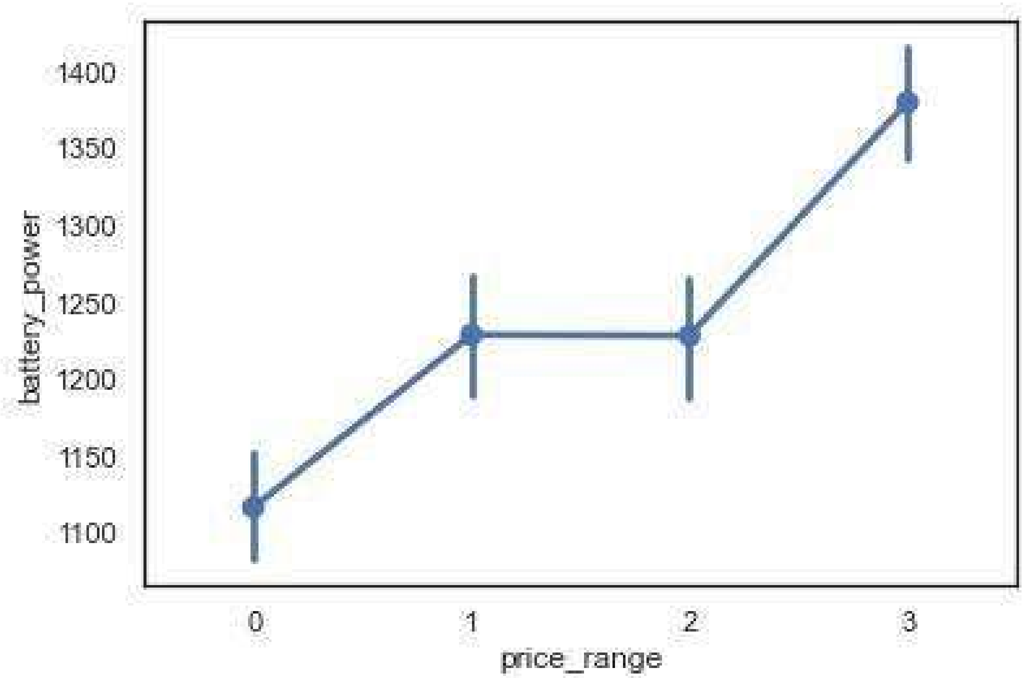



```
plt.figure(figsize=(12,6)) plt.subplot(1,2,1)
sns.barplot(x='price_range',y='fc',data=df,palette='Reds')
plt.subplot(1,2,2)
```

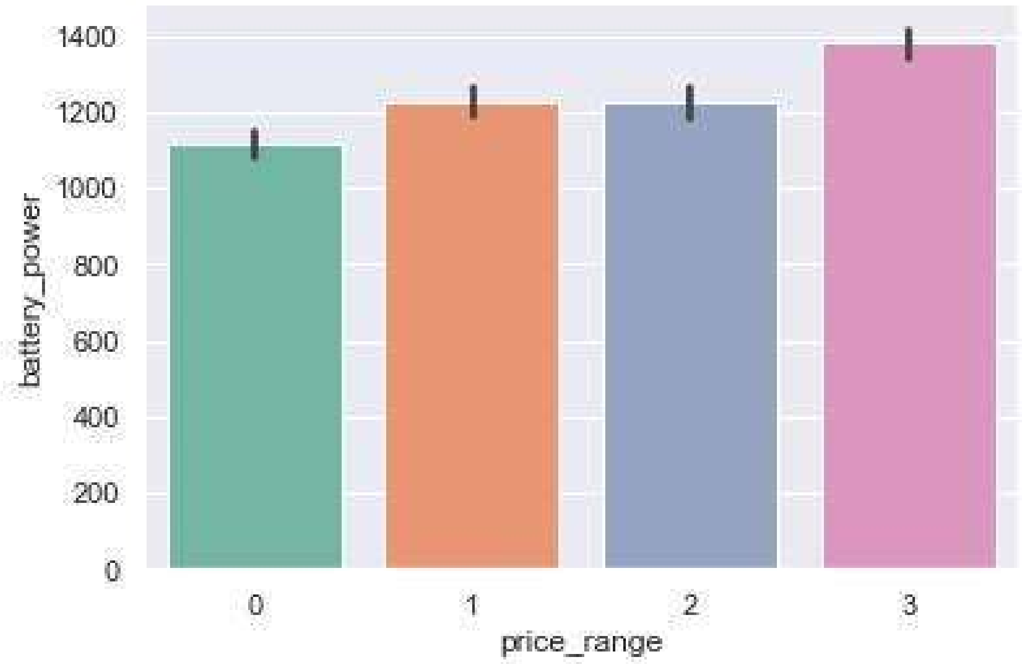
```
sns.barplot(x='price_range',y='pc',data=df,palette='Reds')
plt.show()
```




```
sns.pointplot(x='price_range',y='battery_power',data=df)
plt.show()
```

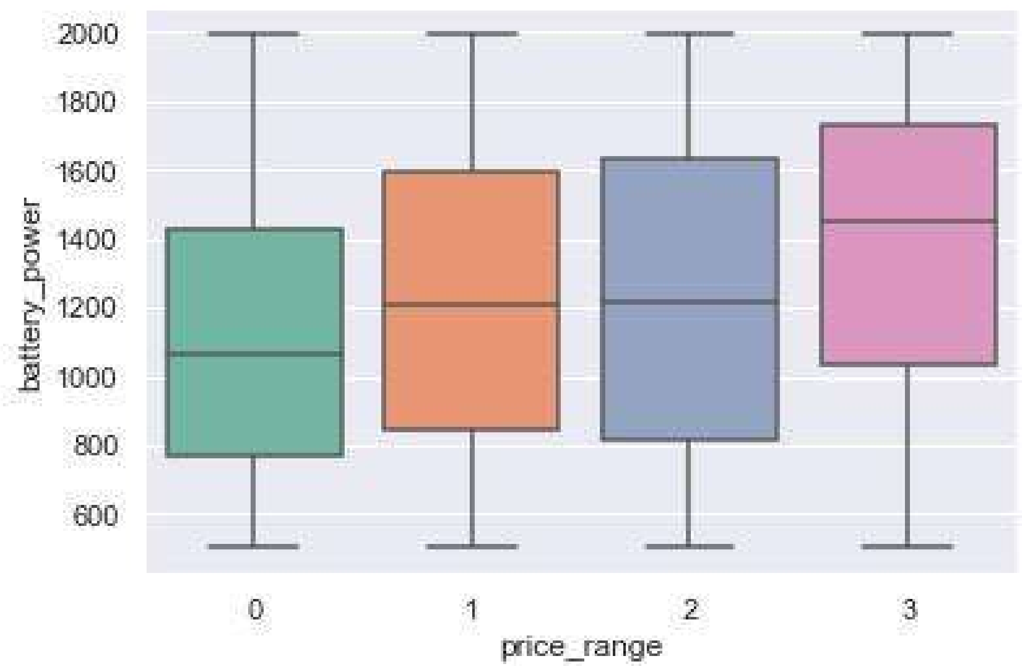



```
sns.set_style('darkgrid') sns.set_palette('Set2')
sns.barplot(x='price_range',y='battery_power',data=df)
plt.show()
```



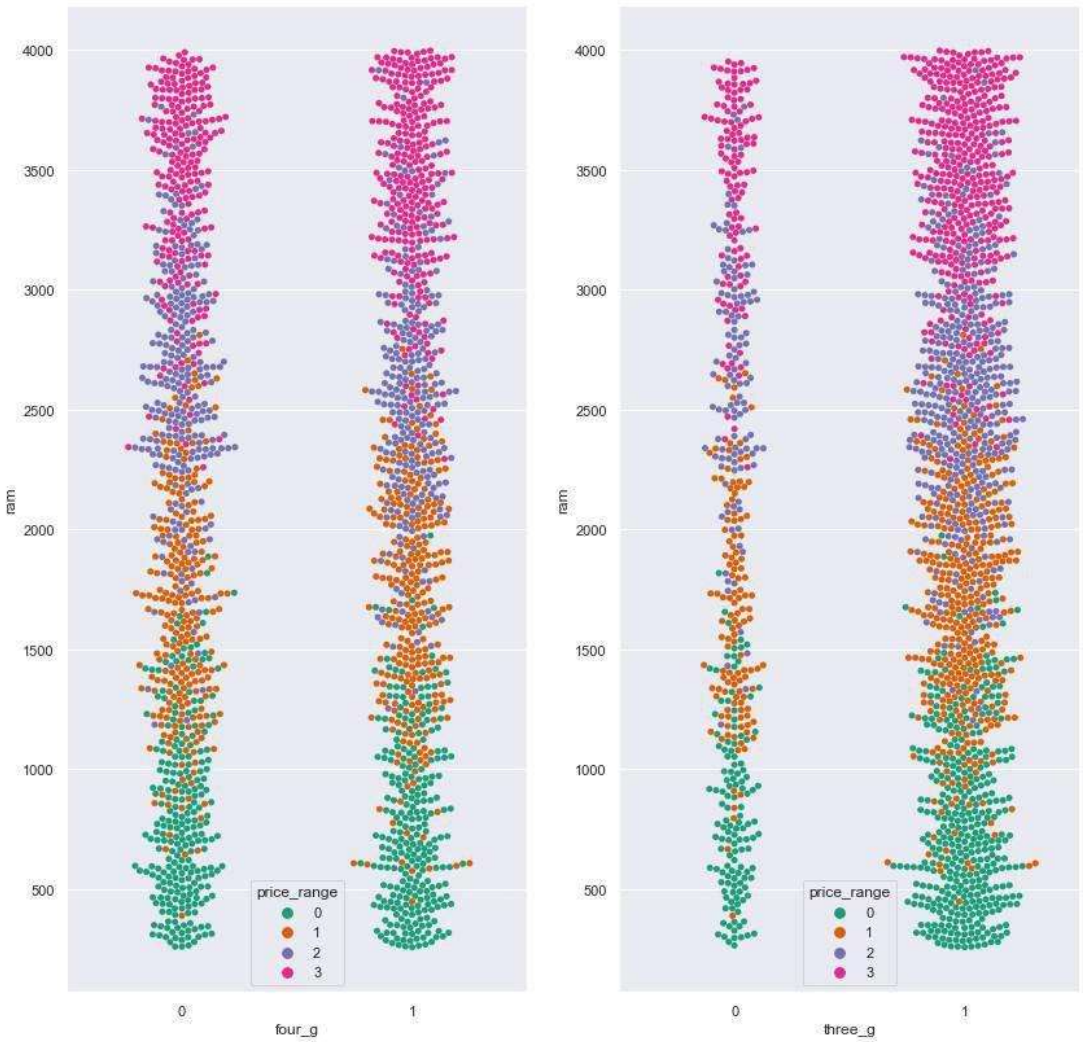

```
sns.boxplot(x='price_range',y='battery_power',data=df)
```

```
<AxesSubplot:xlabel='price_range', ylabel='battery_power'>
```

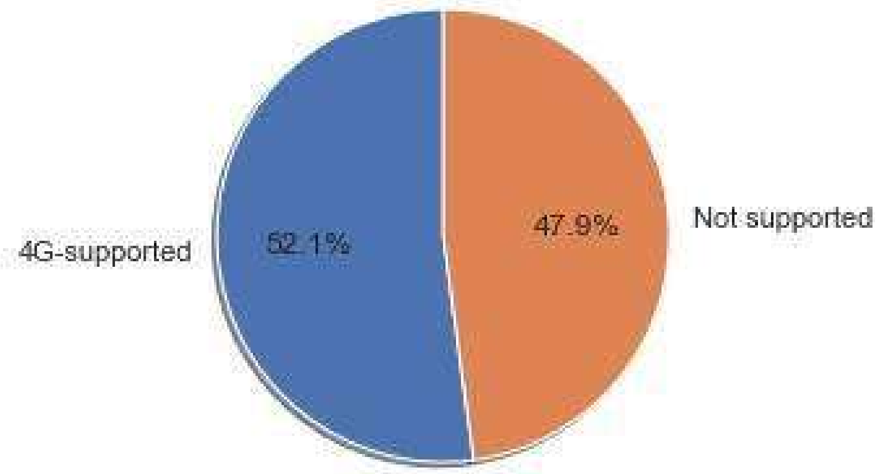



```
f, ax = plt.subplots(figsize=(14, 14))
plt.subplot(1,2,1)
ax=sns.swarmplot(x="four_g", y="ram", hue="price_range",
              palette="Dark2", data=df)
```

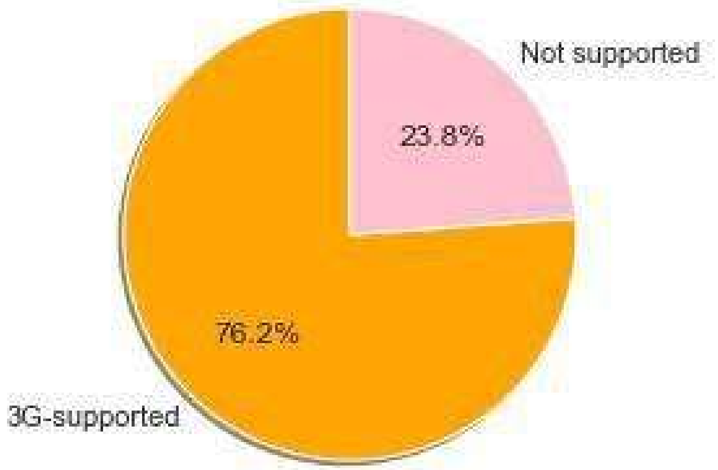
```
ax=sns.set(style="darkgrid")

plt.subplot(1,2,2)
ax=sns.swarmplot(x="three_g", y="ram", hue="price_range",
                 palette="Dark2", data=df)
ax=sns.set(style="darkgrid")
```




```
fig1, ax1 = plt.subplots()
columns =["4G-supported",'Not supported']
ax1.pie(df.four_g.value_counts().values, labels=columns,
autopct='%1.1f%%',shadow=True, startangle=90)
plt.show()
```




```
fig2, ax1 = plt.subplots()
columns =["3G-supported",'Not supported'] colors =
['orange', 'pink']
ax1.pie(df.three_g.value_counts().values, labels=columns,
autopct='%1.1f%%',shadow=True, startangle=90,colors=colors)
plt.show()
```




```
x=df.drop(['price_range'],axis=1)
y=df['price_range']

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,rando
m_state=23)
```

```
parameters = {'n_neighbors':np.arange(1,20,1)}
knn=KNeighborsClassifier()
clf = GridSearchCV(knn, parameters)
clf.fit(x_train,y_train)
```

```
GridSearchCV(estimator=KNeighborsClassifier(),
             param_grid={'n_neighbors': array([ 1,  2,  3,  4,  5,  6,
7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19])})
```

```
knn=KNeighborsClassifier(**clf.best_params_)
knn.fit(x_train,y_train)
```

```
KNeighborsClassifier(n_neighbors=15)
```

```
y_pred=knn.predict(x_test)
print("\nConfusion Matrix:\n%s"%confusion_matrix(y_pred,y_test))
print("\nTest Set Accuracy:"+str(accuracy_score(y_pred,y_test)*100))
```

```
Confusion Matrix:
[[148   6   0   0]
 [  3 141   6   0]
 [  0   7 134   5]
 [  0   0   8 142]]

Test Set Accuracy:94.16666666666667
```

```
knn_acc=accuracy_score(y_pred,y_test)
knn_acc
```

```
0.9416666666666667
```

```
svc = SVC()
svc.fit(x_train, y_train)
```

```
SVC()
```

```
y_pred=svc.predict(x_test)
print("\nConfusion Matrix:\n%s"%confusion_matrix(y_pred,y_test))
print("\nTest Set Accuracy:"+str(accuracy_score(y_pred,y_test)*100))
```

```
Confusion Matrix:
[[148   5   0   0]
 [  3 143   3   0]
 [  0   6 138   0]
 [  0   0   7 147]]

Test Set Accuracy:96.0
```

```
svc_acc=accuracy_score(y_pred,y_test)
svc_acc
```

```
0.96
```

```
parameters = {#'booster':'gbtree',
              'learning_rate':[0.1,0.2,0.3],
              'max_depth':[3,4,5],
              'n_estimators':[5,8,10],
              'gamma':[3,4,5]}
xgb=XGBClassifier()
clf = GridSearchCV(xgb, parameters)
clf.fit(x_train,y_train,eval_metric='rmse')
```

```
GridSearchCV(estimator=XGBClassifier(base_score=None, booster=None,
                                     colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=None,
                                     enable_categorical=False,
gamma=None,
                                     gpu_id=None,
importance_type=None,
                                     interaction_constraints=None,
                                     learning_rate=None,
max_delta_step=None,
                                     max_depth=None,
min_child_weight=None,
                                     missing=nan,
monotone_constraints=None,
                                     n_estimators=100, n_jobs=None,
                                     num_parallel_tree=None,
predictor=None,
                                     random_state=None,
reg_alpha=None,
                                     reg_lambda=None,
scale_pos_weight=None,
                                     subsample=None, tree_method=None,
                                     validate_parameters=None,
verbosity=None),


             param_grid={'gamma': [3, 4, 5], 'learning_rate': [0.1,
0.2, 0.3],
                         'max_depth': [3, 4, 5], 'n_estimators': [5,
8, 10]})
```

```
xgb = XGBClassifier(**clf.best_params_)
xgb.fit(x_train,y_train,eval_metric='rmse')
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1,
enable_categorical=False,
              gamma=3, gpu_id=-1, importance_type=None,
```

```
              interaction_constraints='', learning_rate=0.3,
max_delta_step=0,
              max_depth=5, min_child_weight=1, missing=nan,
              monotone_constraints='()', n_estimators=10, n_jobs=4,
              num_parallel_tree=1, objective='multi:softprob',
predictor='auto',
              random_state=0, reg_alpha=0, reg_lambda=1,
scale_pos_weight=None,
              subsample=1, tree_method='exact', validate_parameters=1,
              verbosity=None)
```

```
y_pred=xgb.predict(x_test)
print("\nConfusion Matrix:\n%s"%confusion_matrix(y_pred,y_test))
print("\nTest Set Accuracy:"+str(accuracy_score(y_pred,y_test)*100))
```

```
Confusion Matrix:
[[141  11   0   0]
 [ 10 126  11   0]
 [  0  17 123   9]
 [  0   0  14 138]]

Test Set Accuracy:88.0
```

```
xgb_acc=accuracy_score(y_pred,y_test)
xgb_acc
```

```
0.88
```

```
gbc = GradientBoostingClassifier()
gbc.fit(x_train, y_train)
```

```
GradientBoostingClassifier()
```

```
y_pred=gbc.predict(x_test)
print("\nConfusion Matrix:\n%s"%confusion_matrix(y_pred,y_test))
print("\nTest Set Accuracy:"+str(accuracy_score(y_pred,y_test)*100))
```

```
Confusion Matrix:
[[146  10   0   0]
 [  5 128   8   0]
 [  0  16 135   8]
 [  0   0   5 139]]

Test Set Accuracy:91.33333333333333
```

```
gbc_acc=accuracy_score(y_pred,y_test)
gbc_acc
```

```
0.9133333333333333
```

```
parameters = {'learning_rate':[0.01,0.1,1,10],
              'n_estimators':[50,80,100,120,150,180],
              'algorithm':['SAMME', 'SAMME.R']}
abc=AdaBoostClassifier()
clf = GridSearchCV(abc, parameters)
clf.fit(x_train,y_train)
```

```
GridSearchCV(estimator=AdaBoostClassifier(),
             param_grid={'algorithm': ['SAMME', 'SAMME.R'],
                         'learning_rate': [0.01, 0.1, 1, 10],
                         'n_estimators': [50, 80, 100, 120, 150,
180]})
```

```
abc = AdaBoostClassifier(**clf.best_params_)
abc.fit(x_train, y_train)
```

```
AdaBoostClassifier(algorithm='SAMME', learning_rate=0.1,
n_estimators=150)
```

```
y_pred=abc.predict(x_test)
print("\nConfusion Matrix:\n%s"%confusion_matrix(y_pred,y_test))
print("\nTest Set Accuracy:"+str(accuracy_score(y_pred,y_test)*100))
```

```
Confusion Matrix:
[[126  22   0   0]
 [ 25 110  28   0]
 [  0  22  75  11]
 [  0   0  45 136]]

Test Set Accuracy:74.5
```

```
abc_acc=accuracy_score(y_pred,y_test)
abc_acc
```

```
0.745
```

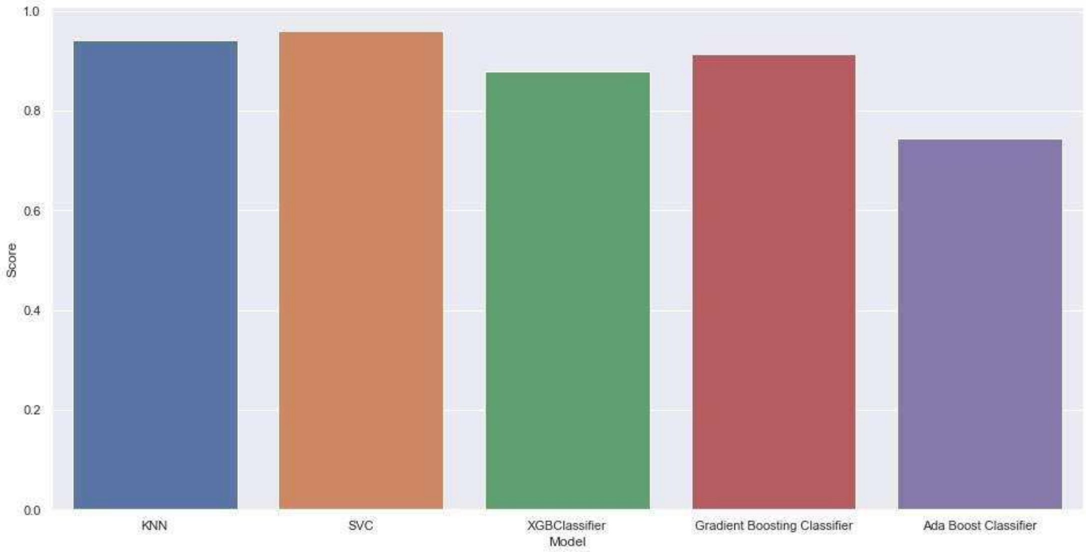
```
models = pd.DataFrame({
    'Model': ['KNN','SVC','XGBClassifier','Gradient Boosting
Classifier',
              'Ada Boost Classifier'],
    'Score': [knn_acc, svc_acc, xgb_acc, gbc_acc, abc_acc]
})
```

```
models.sort_values(by = 'Score', ascending = False)
```

	Model	Score
1	SVC	0.960000
0	KNN	0.941667
3	Gradient Boosting Classifier	0.913333
2	XGBClassifier	0.880000
4	Ada Boost Classifier	0.745000

```
plt.figure(figsize=(16,8))
sns.barplot(x='Model',y='Score',data=models)
```

```
<AxesSubplot:xlabel='Model', ylabel='Score'>
```




```
from mlxtend.evaluate import bias_variance_decomp
```

```
mse, bias, var = bias_variance_decomp(svc,x_train.values,
y_train.values, x_test.values, y_test.values, loss='mse',
num_rounds=100, random_seed=53)
```

```
print('Bias: %.3f' %bias)
```

```
Bias: 0.034
```

```
print('Variance: %.3f' %var)
```

```
Variance: 0.017
```

```
print('MSE: %.3f' %mse)
```

```
MSE: 0.051
```