In [ ]:

```
AIM: Create a classification model to predict whether price range of mobile
based on certain specifications
```

In [ ]:

```
DESCRIPTION: An entrepreneur 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, one cannot simply assume things. To solve this problem, he collects
sales data of mobile phones of various companies.
He wants to find out some relation between features of a mobile phone (e.g., 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
```

In [ ]:

```
SOURCE CODE:
```

In [ ]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.metrics import accuracy_score,confusion_matrix
import warnings
warnings.filterwarnings('ignore')
```

In [4]:

```
df=pd.read_csv('D:\mobile_price_range_data.csv')
df.head()
```

Out[4]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	px_height	px_width	ram	sc_h	sc_w	talk_time
0	842	0	2.2	0	1	0	7	0.6	188	2	...	20	756	2549	9	7	19
1	1021	1	0.5	1	0	1	53	0.7	136	3	...	905	1988	2631	17	3	7
2	563	1	0.5	1	2	1	41	0.9	145	5	...	1263	1716	2603	11	2	9
3	615	1	2.5	0	0	0	10	0.8	131	6	...	1216	1786	2769	16	8	11
4	1821	1	1.2	0	13	1	44	0.6	141	2	...	1208	1212	1411	8	2	15

rows × 21 columns

In [5]:

```
df.shape
```

Out[5]:

```
(2000, 21)
```

In [6]:

```
df.dtypes
```

Out[6]:

```
battery_power      int64
blue               int64
clock_speed      float64
dual_sim           int64
fc                 int64
four_g             int64
int_memory         int64
m_dep            float64
mobile_wt          int64
n_cores            int64
pc                 int64
px_height          int64
px_width           int64
ram                int64
sc_h               int64
sc_w               int64
talk_time          int64
three_g            int64
touch_screen       int64
wifi               int64
price_range        int64
dtype: object
```

In [7]:

```
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
```

In [8]:

```
df.describe()
```

Out[8]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	px_heigl
count	2000.000000	2000.0000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	...	2000.0000C
mean	1238.518500	0.4950	1.522250	0.509500	4.309500	0.521500	32.046500	0.501750	140.249000	4.520500	...	645.1080C
std	439.418206	0.5001	0.816004	0.500035	4.341444	0.499662	18.145715	0.288416	35.399655	2.287837	...	443.78081
min	501.000000	0.0000	0.500000	0.000000	0.000000	0.000000	2.000000	0.100000	80.000000	1.000000	...	0.0000C
25%	851.750000	0.0000	0.700000	0.000000	1.000000	0.000000	16.000000	0.200000	109.000000	3.000000	...	282.7500C
50%	1226.000000	0.0000	1.500000	1.000000	3.000000	1.000000	32.000000	0.500000	141.000000	4.000000	...	564.0000C
75%	1615.250000	1.0000	2.200000	1.000000	7.000000	1.000000	48.000000	0.800000	170.000000	7.000000	...	947.2500C
max	1998.000000	1.0000	3.000000	1.000000	19.000000	1.000000	64.000000	1.000000	200.000000	8.000000	...	1960.0000C

8 rows × 21 columns

In [9]:

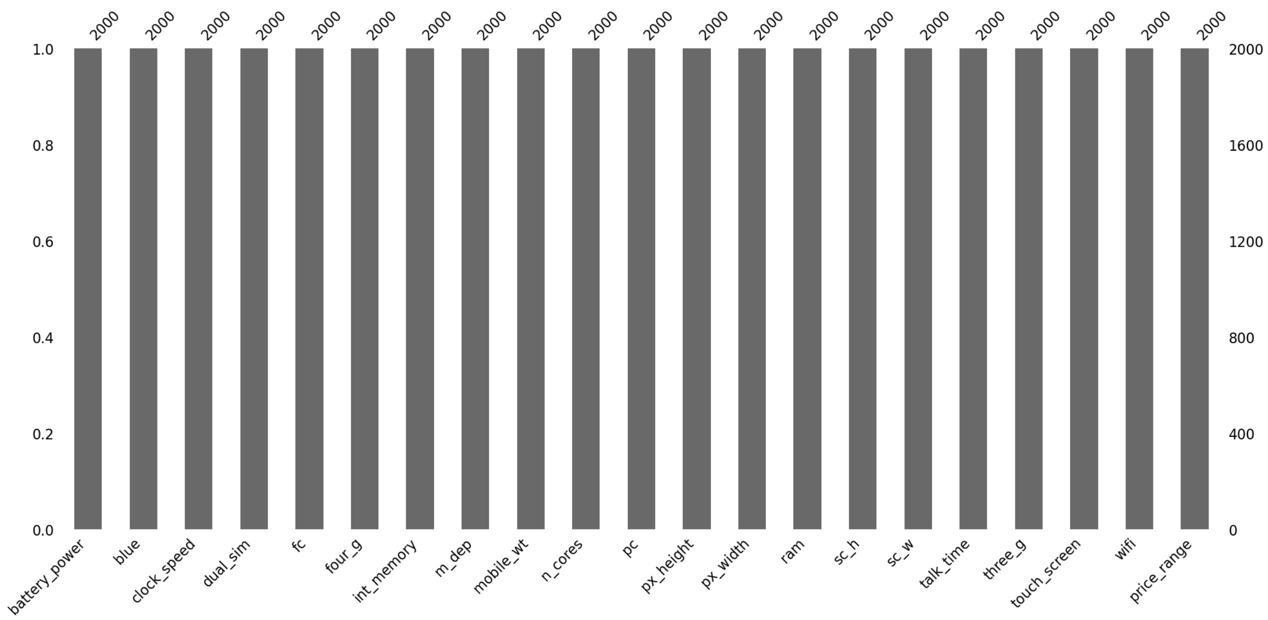
```
df.isnull().sum()
```

Out[9]:

```
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
```

In [11]:

```
import missingno as msno
msno.bar(df)
plt.show()
```



In [12]:

```
df.var()
```

Out[12]:

```
battery_power    1.930884e+05
blue             2.501001e-01
clock_speed      6.658629e-01
dual_sim         2.500348e-01
fc               1.884813e+01
four_g           2.496626e-01
int_memory       3.292670e+02
m_dep            8.318353e-02
mobile_wt        1.253136e+03
n_cores          5.234197e+00
pc               3.677592e+01
px_height        1.969414e+05
px_width         1.867964e+05
ram              1.176644e+06
sc_h             1.775143e+01
sc_w             1.897820e+01
talk_time        2.985481e+01
three_g          1.817086e-01
touch_screen     2.501161e-01
wifi             2.500760e-01
price_range      1.250625e+00
dtype: float64
```

In [13]:

```
df['price_range'].unique()
```

Out[13]:

```
array([1, 2, 3, 0], dtype=int64)
```

In [14]:

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

Out[14]:

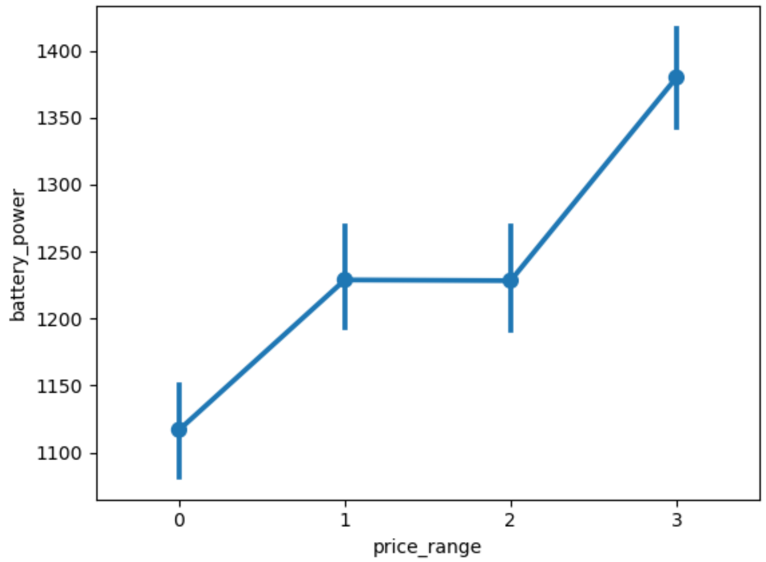
```
<AxesSubplot:xlabel='price_range', ylabel='ram'>
```

In [15]:

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

Out[15]:

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

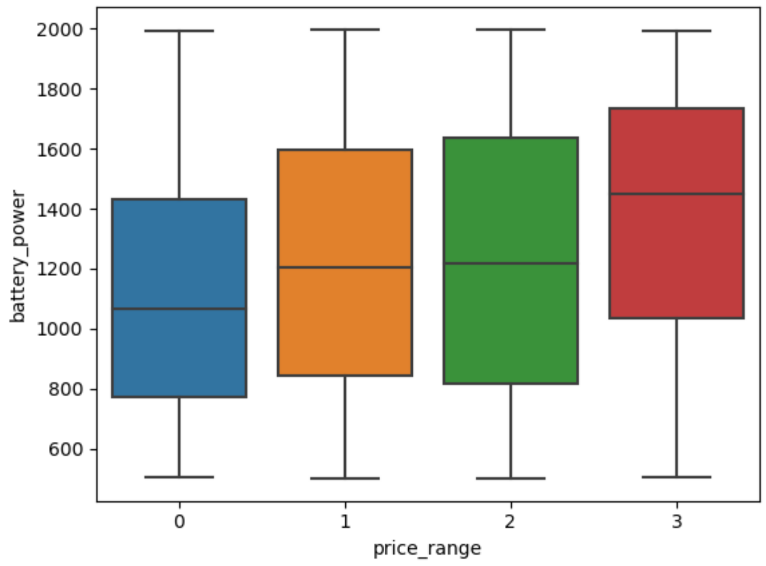


In [16]:

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

Out[16]:

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

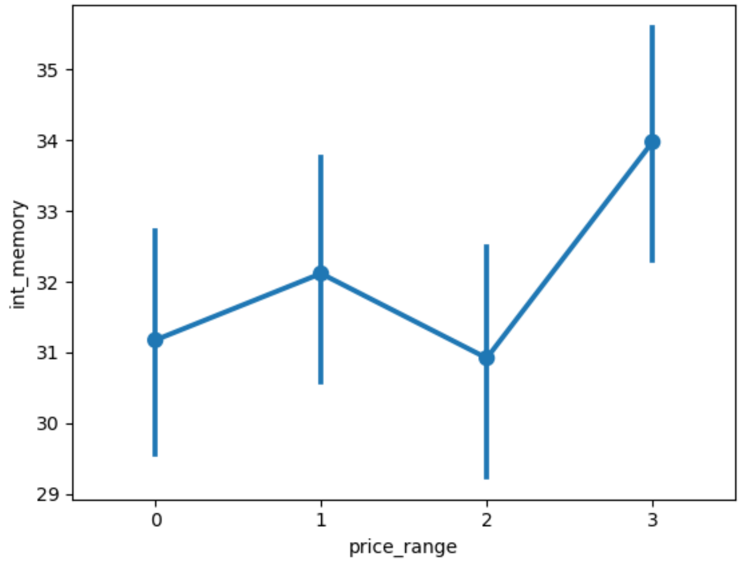


In [17]:

```
sns.pointplot(x='price_range',y='int_memory',data=df)
```

Out[17]:

<AxesSubplot:xlabel='price_range', ylabel='int_memory'>



In [18]:

```
col = df.columns
col
```

Out[18]:

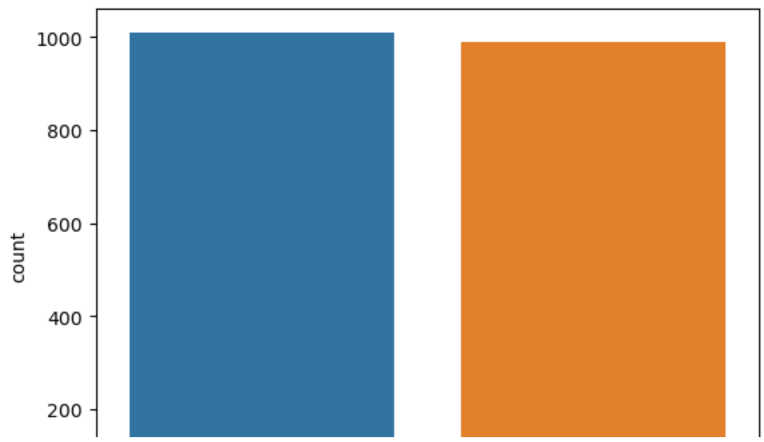
```
Index(['battery_power', 'blue', 'clock_speed', 'dual_sim', 'fc', 'four_g',
       'int_memory', 'm_dep', 'mobile_wt', 'n_cores', 'pc', 'px_height',
       'px_width', 'ram', 'sc_h', 'sc_w', 'talk_time', 'three_g',
       'touch_screen', 'wifi', 'price_range'],
      dtype='object')
```

In [22]:

```
categorical_col = ['blue', 'dual_sim', 'four_g', 'three_g','touch_screen', 'price_range']
```

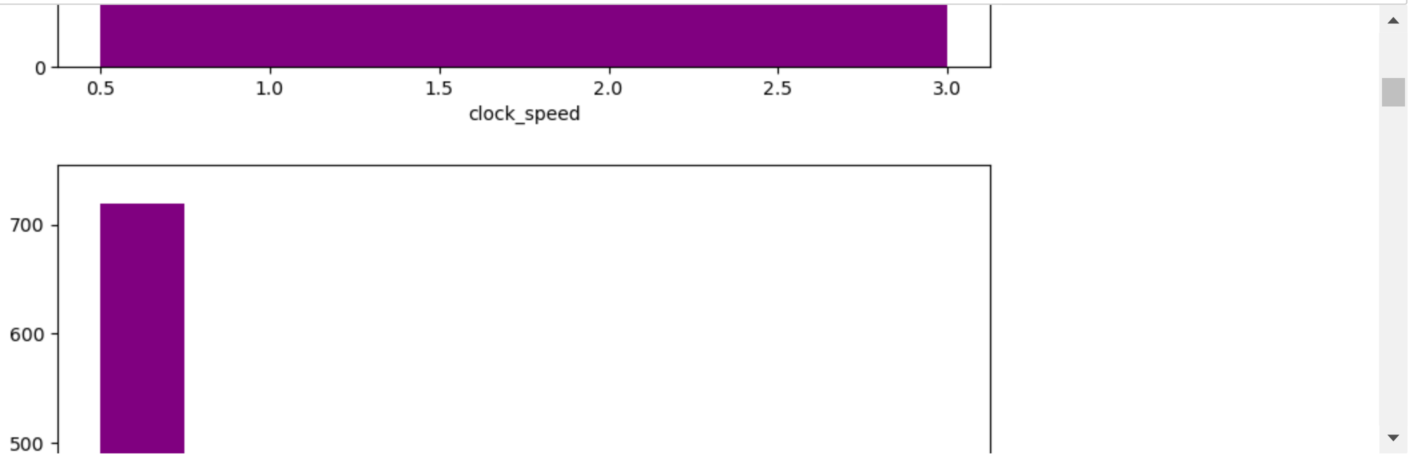
In [23]:

```
for i in categorical_col:
    sns.countplot(df[i])
    plt.xlabel(i)
    plt.show()
```



In [24]:

```
for i in df.drop(df[categorical_col],axis=1):
    fig = plt.figure(figsize=(9,8))
    plt.hist(df[i],color='purple',bins=10)
    plt.xlabel(i)
    plt.show()
```

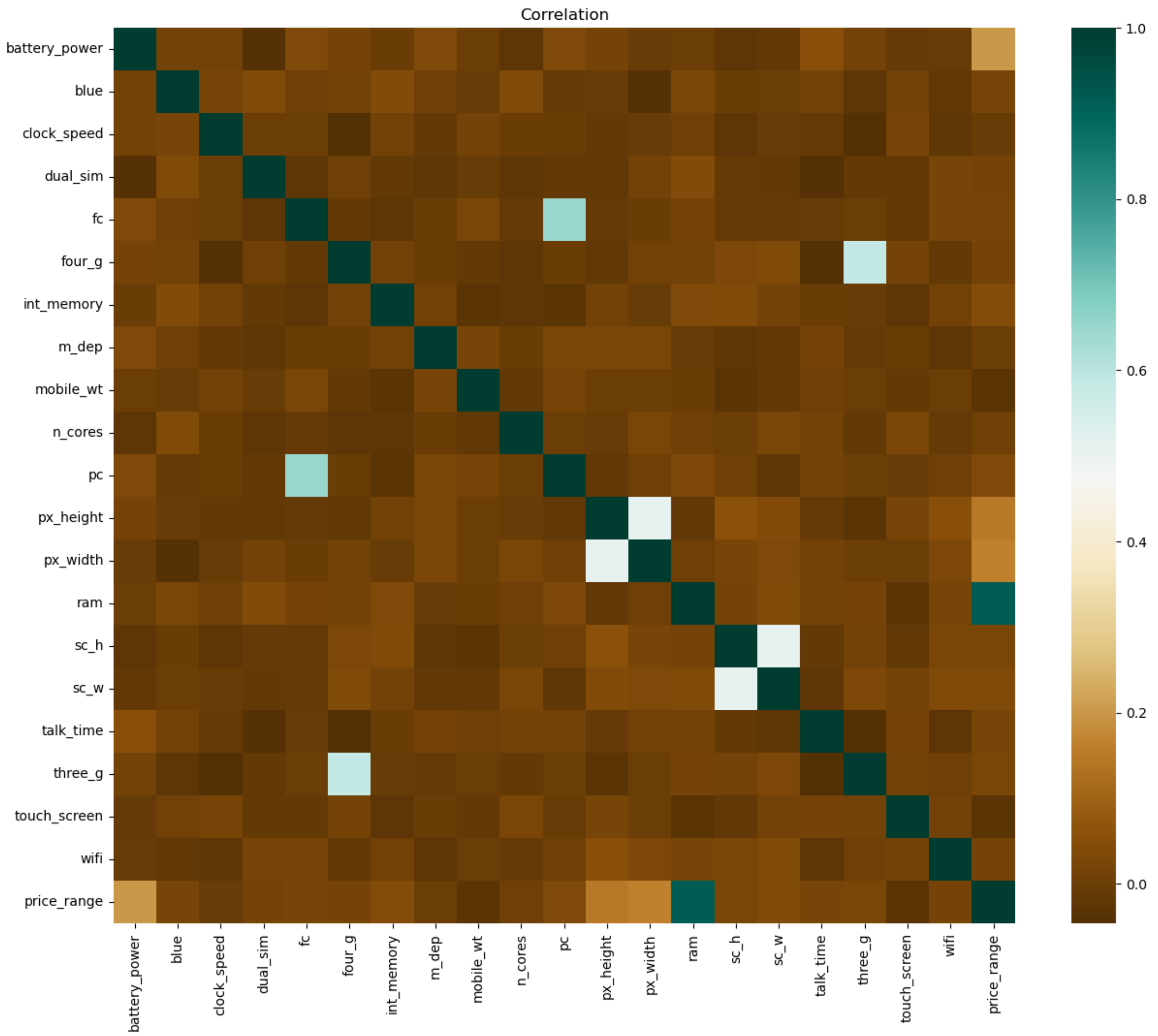


In [25]:

```
corr=df.corr()
fig = plt.figure(figsize=(15,12))
r = sns.heatmap(corr, cmap='BrBG')
r.set_title("Correlation")
```

Out[25]:

```
Text(0.5, 1.0, 'Correlation')
```

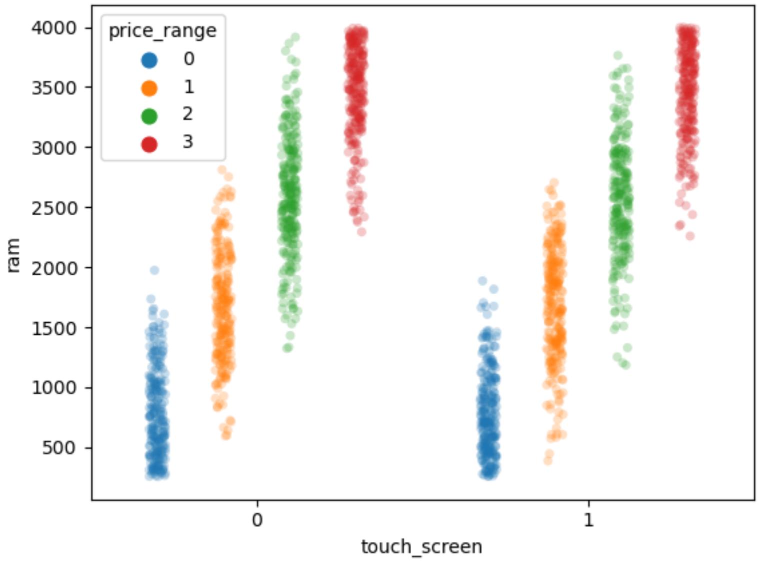


In [26]:

```
sns.stripplot(x="touch_screen",y="ram", hue="price_range",data=df, dodge=True,jitter=True, alpha=.25, zorder=1)
```

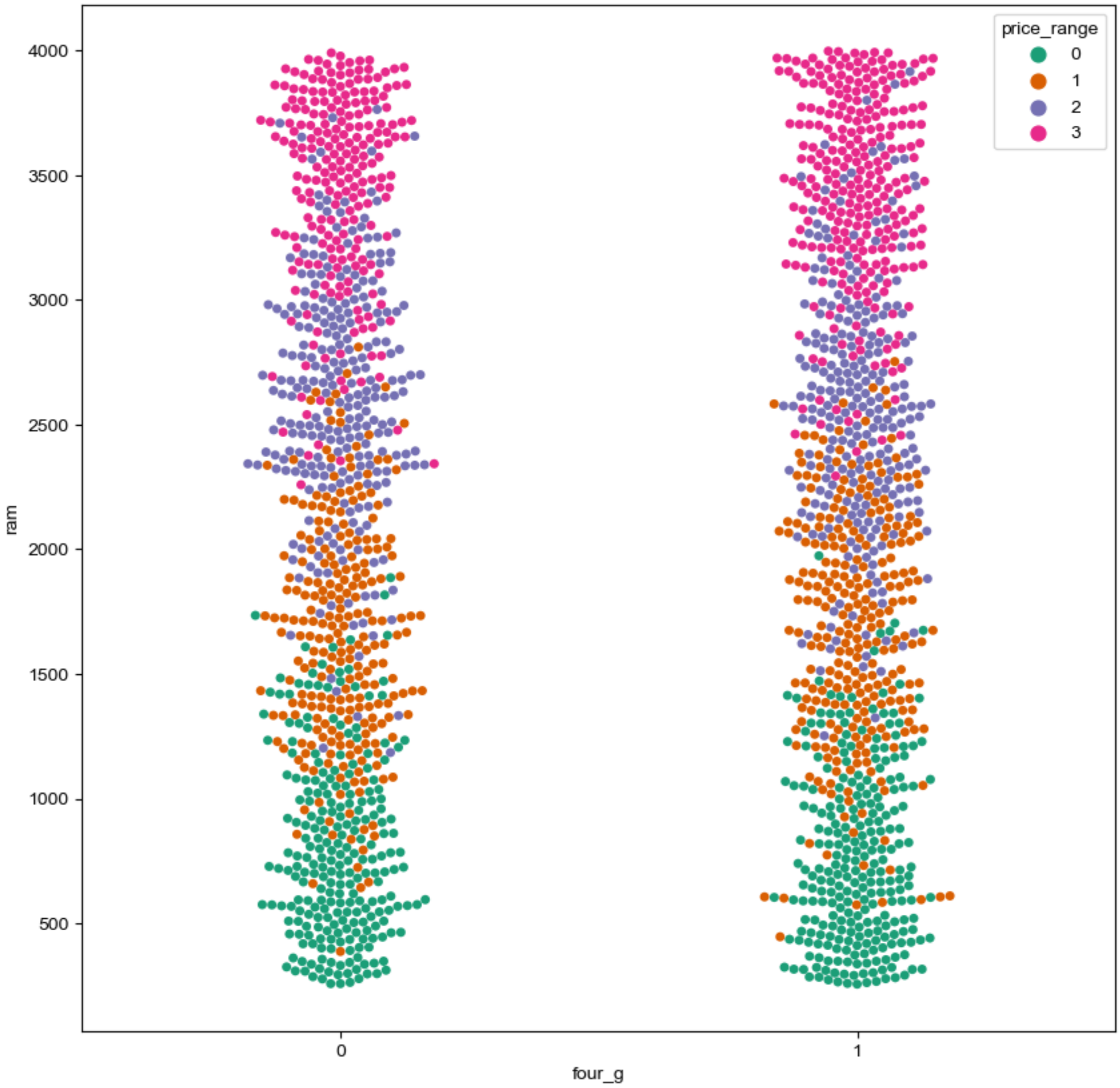
Out[26]:

```
<AxesSubplot:xlabel='touch_screen', ylabel='ram'>
```



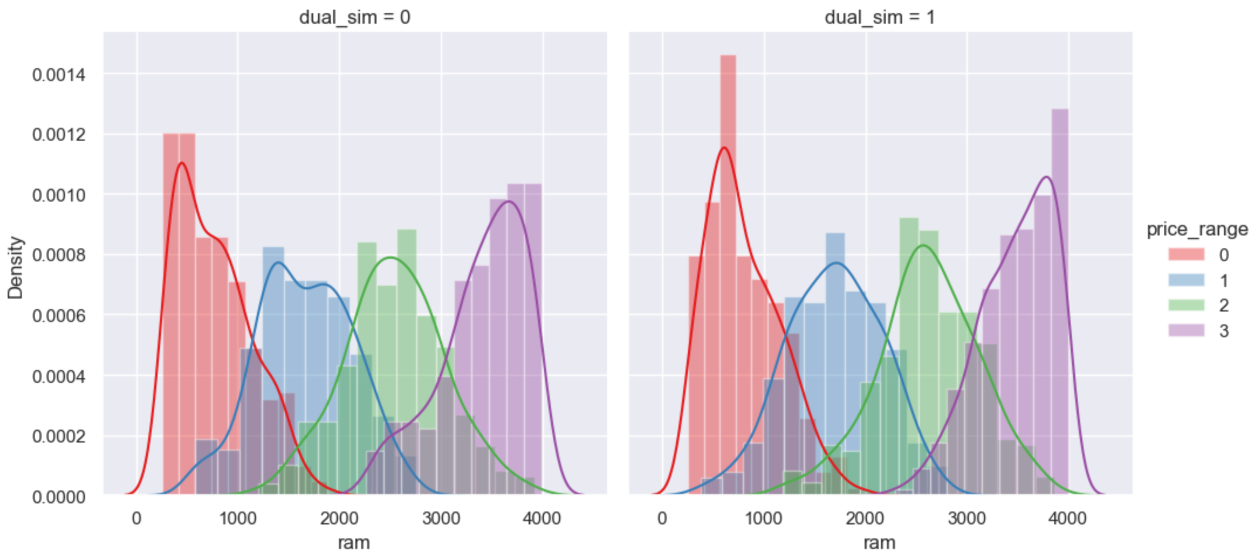
In [27]:

```
f, ax = plt.subplots(figsize=(10,10))
ax=sns.swarmplot(x="four_g", y="ram", hue="price_range", palette="Dark2", data=df)
ax=sns.set(style="darkgrid")
```



In [29]:

```
g = sns.FacetGrid(df, col="dual_sim", hue="price_range", palette="Set1", height=5)
g = (g.map(sns.distplot, "ram").add_legend())
```



In [30]:

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

In [31]:

```
scale=StandardScaler()
scaled=scale.fit_transform(x)
```

In [33]:

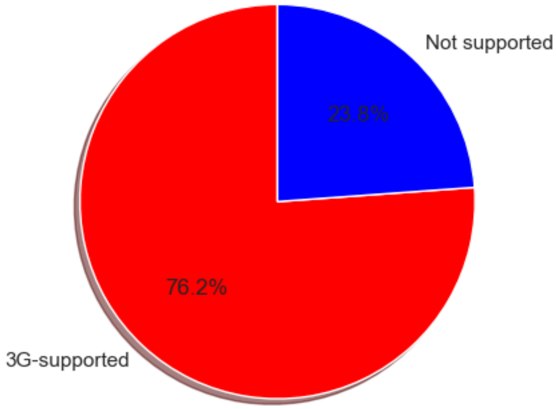
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
vif=pd.DataFrame()
vif['vif']=[variance_inflation_factor(scaled,i)for i in range(scaled.shape[1])]
vif['features']=x.columns
vif
```

Out[33]:

	vif	features
0	1.009945	battery_power
1	1.011342	blue
2	1.006025	clock_speed
3	1.011555	dual_sim
4	1.718987	fc
5	1.528509	four_g
6	1.009274	int_memory
7	1.006385	m_dep
8	1.004548	mobile_wt
9	1.008442	n_cores
10	1.720785	pc
11	1.369052	px_height
12	1.362399	px_width
13	1.008331	ram
14	1.356109	sc_h
15	1.353648	sc_w
16	1.010502	talk_time
17	1.527367	three_g
18	1.006278	touch_screen
19	1.009100	wifi

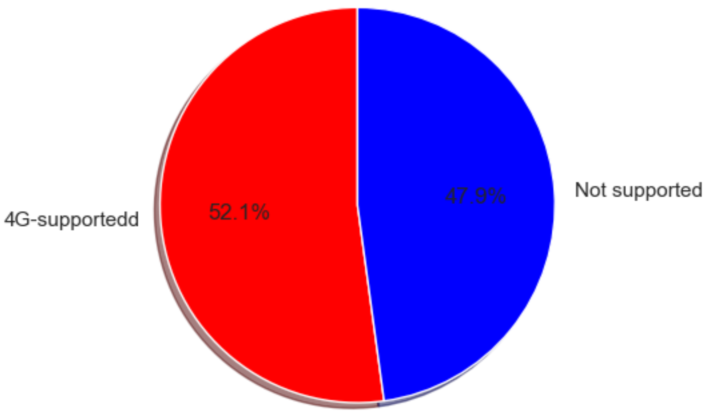
In [35]:

```
labels = ["3G-supported", 'Not supported']
values = df['three_g'].value_counts().values
fig1, ax1 = plt.subplots()
colors = ['red', 'blue']
ax1.pie(values, labels=labels, autopct='%1.1f%%',shadow=True,startangle=90,colors=colors)
plt.show()
```



In [39]:

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

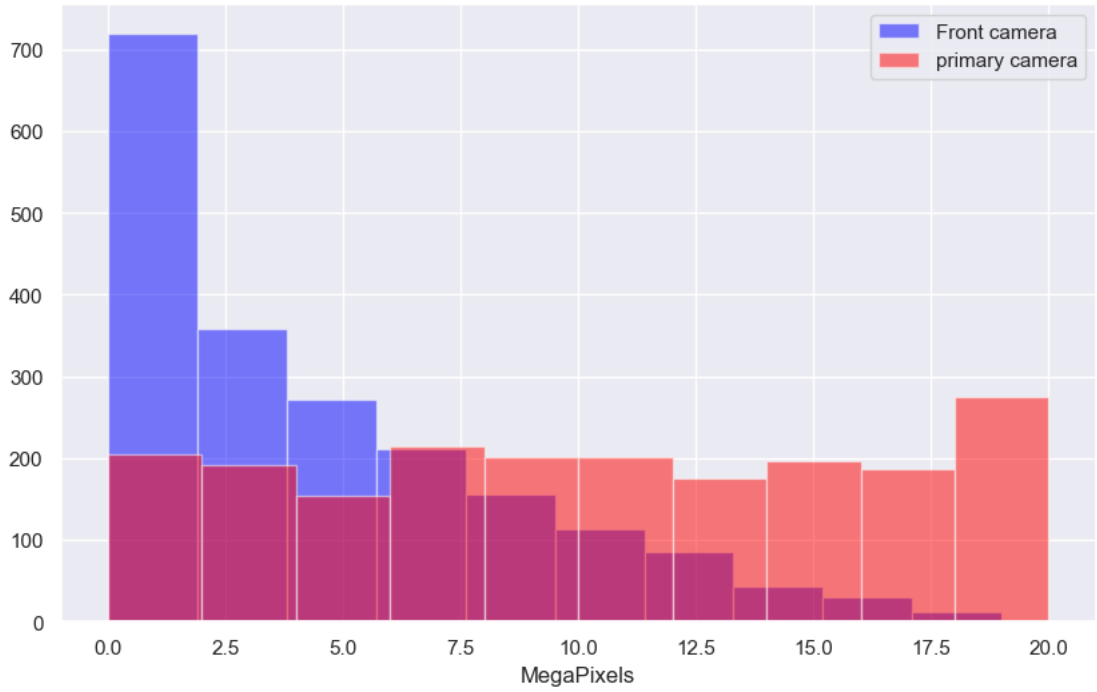


In [40]:

```
plt.figure(figsize=(10,6))
df['fc'].hist(alpha=0.5,color='blue',label='Front camera')
df['pc'].hist(alpha=0.5,color='red',label='primary camera')
plt.legend()
plt.xlabel('MegaPixels')
```

Out[40]:

```
Text(0.5, 0, 'MegaPixels')
```



In [41]:

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,random_state=123,stratify=y)
```

In [48]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
lr = LogisticRegression(penalty='l2',C=0.1)
lr.fit(x_train,y_train)

y_test_pred = lr.predict(x_test)
y_train_pred = lr.predict(x_train)

lr_acc=accuracy_score(y_test_pred,y_test)

print("Train Set Accuracy:"+str(accuracy_score(y_train_pred,y_train)*100))
print("Test Set Accuracy:"+str(accuracy_score(y_test_pred,y_test)*100))
print("\nConfusion Matrix:\n%s"%confusion_matrix(y_test_pred,y_test))
print("\nClassification Report:\n%s"%classification_report(y_test_pred,y_test))
```

```
Train Set Accuracy:64.26666666666667
Test Set Accuracy:63.0

Confusion Matrix:
[[97 24  0  0]
 [27 66 35  1]
 [ 1 30 57 29]
 [ 0  5 33 95]]

Classification Report:
              precision    recall  f1-score   support

           0       0.78      0.80      0.79       121
           1       0.53      0.51      0.52       129
           2       0.46      0.49      0.47       117
           3       0.76      0.71      0.74       133

    accuracy                           0.63       500
   macro avg       0.63      0.63      0.63       500
weighted avg       0.63      0.63      0.63       500
```

In [50]:

```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=8)
knn.fit(x_train,y_train)
y_test_pred1 = knn.predict(x_test)
y_train_pred1=knn.predict(x_train)
knn_acc=accuracy_score(y_test_pred1,y_test)
print("Train Set Accuracy:"+str(accuracy_score(y_train_pred1,y_train)*100))
print("Test Set Accuracy:"+str(accuracy_score(y_test_pred1,y_test)*100))
print("\nConfusion Matrix:\n%s"%confusion_matrix(y_test_pred1,y_test))
print("\nClassification Report:\n%s"%classification_report(y_test_pred1,y_test))
```

```
Train Set Accuracy:95.13333333333334
Test Set Accuracy:89.8

Confusion Matrix:
[[124  15   0   0]
 [  1 104  12   0]
 [  0   6 110  14]
 [  0   0   3 111]]

Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.89      0.94       139
           1       0.83      0.89      0.86       117
           2       0.88      0.85      0.86       130
           3       0.89      0.97      0.93       114

    accuracy                           0.90       500
   macro avg       0.90      0.90      0.90       500
weighted avg       0.90      0.90      0.90       500
```

In [51]:

```
from sklearn.svm import SVC
svc = SVC()
svc.fit(x_train, y_train)
y_test_pred2 = svc.predict(x_test)
y_train_pred2=svc.predict(x_train)
svc_acc=accuracy_score(y_test_pred2,y_test)
print("Train Set Accuracy:"+str(accuracy_score(y_train_pred2,y_train)*100))
print("Test Set Accuracy:"+str(accuracy_score(y_test_pred2,y_test)*100))
print("\nConfusion Matrix:\n%s"%confusion_matrix(y_test_pred2,y_test))
print("\nClassification Report:\n%s"%classification_report(y_test_pred2,y_test))
```

```
Train Set Accuracy:94.93333333333334
Test Set Accuracy:94.0

Confusion Matrix:
[[124   8   0   0]
 [  1 114   9   0]
 [  0   3 112   5]
 [  0   0   4 120]]

Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.94      0.96       132
           1       0.91      0.92      0.92       124
           2       0.90      0.93      0.91       120
           3       0.96      0.97      0.96       124

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

In [52]:

```
from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier()
dtc.fit(x_train, y_train)
y_test_pred3 = dtc.predict(x_test)
y_train_pred3=dtc.predict(x_train)
print("Train Set Accuracy:"+str(accuracy_score(y_train_pred3,y_train)*100))
print("Test Set Accuracy:"+str(accuracy_score(y_test_pred3,y_test)*100))
print("\nConfusion Matrix:\n%s"%confusion_matrix(y_test_pred3,y_test))
print("\nClassification Report:\n%s"%classification_report(y_test_pred3,y_test))
```

```
Train Set Accuracy:100.0
Test Set Accuracy:82.0

Confusion Matrix:
[[109  13   0   0]
 [ 15  98  19   1]
 [  1  14  92  13]
 [  0   0  14 111]]

Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.89      0.88       122
           1       0.78      0.74      0.76       133
           2       0.74      0.77      0.75       120
           3       0.89      0.89      0.89       125

    accuracy                           0.82       500
   macro avg       0.82      0.82      0.82       500
weighted avg       0.82      0.82      0.82       500
```

In [54]:

```
grid_params = {
    'criterion' : ['gini', 'entropy'],
    'max_depth' : [3, 5, 7 , 10],
    'min_samples_split' : range(2, 10, 1),
    'min_samples_leaf' : range(2, 10, 1)
}
grid_search = GridSearchCV(dtc, grid_params, cv = 5, n_jobs = -1,verbose = 1)
grid_search.fit(x_train, y_train)
```

```
Fitting 5 folds for each of 512 candidates, totalling 2560 fits
```

Out[54]:

```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(), n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [3, 5, 7, 10],
                         'min_samples_leaf': range(2, 10),
                         'min_samples_split': range(2, 10)},
             verbose=1)
```

In [56]:

```
grid_search.best_params_
```

Out[56]:

```
{'criterion': 'gini',
 'max_depth': 7,
 'min_samples_leaf': 8,
 'min_samples_split': 3}
```

In [57]:

```
dtc = grid_search.best_estimator_
```

In [58]:

```
y_predi=dtc.predict(x_test)
```

In [59]:

```
dtc_train_acc = accuracy_score(y_train, dtc.predict(x_train))
dtc_test_acc = accuracy_score(y_test, y_predi)
print(f"Trainig Accuracy of SVC Model is{dtc_train_acc}")
print(f"Test Accuracy of SVC Model is{dtc_test_acc}")
```

```
Trainig Accuracy of SVC Model is0.9226666666666666
Test Accuracy of SVC Model is0.836
```

In [63]:

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import GridSearchCV
ada = AdaBoostClassifier(base_estimator = dtc)
parameters = {
    'n_estimators' : [50, 70, 90, 120, 180, 200],
    'learning_rate' : [0.001, 0.01, 0.1, 1, 10],
    'algorithm' : ['SAMME', 'SAMME.R']
}
grid_search = GridSearchCV(ada, parameters, n_jobs = -1, cv = 5, verbose = 1)
grid_search.fit(x_train, y_train)
```

```
Fitting 5 folds for each of 60 candidates, totalling 300 fits
```

Out[63]:

```
GridSearchCV(cv=5,
             estimator=AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=7,
                                                                                  min_samples_leaf=8,
                                                                                  min_samples_split=3)),
             n_jobs=-1,
             param_grid={'algorithm': ['SAMME', 'SAMME.R'],
                         'learning_rate': [0.001, 0.01, 0.1, 1, 10],
                         'n_estimators': [50, 70, 90, 120, 180, 200]},
             verbose=1)
```

In [64]:

```
grid_search.best_params_
```

Out[64]:

```
{'algorithm': 'SAMME', 'learning_rate': 1, 'n_estimators': 200}
```

In [65]:

```
grid_search.best_score_
```

Out[65]:

```
0.922
```

In [66]:

```
ad = grid_search.best_estimator_
ad.fit(x_train,y_train)
```

Out[66]:

```
AdaBoostClassifier(algorithm='SAMME',
                   base_estimator=DecisionTreeClassifier(max_depth=7,
                                                         min_samples_leaf=8,
                                                         min_samples_split=3),
                   learning_rate=1, n_estimators=200)
```

In [67]:

```
y_pred = ad.predict(x_test)
```

In [68]:

```
ada_train_acc = accuracy_score(y_train, ad.predict(x_train))
ada_test_acc = accuracy_score(y_test, y_pred)
print(f"Trainig Accuracy of Random Forest Model is{ada_train_acc}")
print(f"Test Accuracy of random Forest Model is{ada_test_acc}")
```

```
Trainig Accuracy of Random Forest Model is1.0
Test Accuracy of random Forest Model is0.92
```

In [70]:

```
from sklearn.ensemble import GradientBoostingClassifier
gbc = GradientBoostingClassifier()
gbc.fit(x_train, y_train)
y_test_pred6 = gbc.predict(x_test)
y_train_pred6=gbc.predict(x_train)
gbc_acc=accuracy_score(y_test_pred6,y_test)
print("Train Set Accuracy:"+str(accuracy_score(y_train_pred6,y_train)*100))
print("Train Set Accuracy:"+str(accuracy_score(y_test_pred6,y_test)*100))
print("\nConfusion Matrix:\n%s"%confusion_matrix(y_test_pred6,y_test))
print("\nClassification Report:\n%s"%classification_report(y_test_pred6,y_test))
```

```
Train Set Accuracy:100.0
Train Set Accuracy:90.8

Confusion Matrix:
[[121   8   0   0]
 [  4 108  10   0]
 [  0   9 111  11]
 [  0   0   4 114]]

Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.94      0.95       129
           1       0.86      0.89      0.87       122
           2       0.89      0.85      0.87       131
           3       0.91      0.97      0.94       118

    accuracy                           0.91       500
   macro avg       0.91      0.91      0.91       500
weighted avg       0.91      0.91      0.91       500
```

In [76]:

```
from xgboost import XGBClassifier
xgb = XGBClassifier(booster = 'gbtree' , learning_rate = 0.1 , max_depth = 5, n_estimators = 10,gamma=5)
xgb.fit(x_train, y_train)
y_test_pred7 = xgb.predict(x_test)
y_train_pred7=xgb.predict(x_train)
xgb_acc= accuracy_score(y_test_pred7,y_test)
print("Train Set Accuracy:"+str(accuracy_score(y_train_pred7,y_train)*100))
print("Test Set Accuracy:"+str(accuracy_score(y_test_pred7,y_test)*100))
print("\nConfusion Matrix:\n%s"%confusion_matrix(y_test_pred7,y_test))
print("\nClassification report:\n%s"%classification_report(y_test_pred7,y_test))
```

```
[16:02:52] WARNING: C:\Windows\Temp\abs_557yfx631l\croots\recipe\xgboost-split_1659548953302\work\src\learner.cc:1115: Star
ting in XGBoost 1.3.0, the default evaluation metric used with the objective 'multi:softprob' was changed from 'merror' to
'mlogloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
Train Set Accuracy:93.06666666666666
Test Set Accuracy:84.6

Confusion Matrix:
[[118  15   0   0]
 [  7 101  21   0]
 [  0   9  95  16]
 [  0   0   9 109]]

Classification report:
              precision    recall  f1-score   support

           0       0.94      0.89      0.91       133
           1       0.81      0.78      0.80       129
           2       0.76      0.79      0.78       120
           3       0.87      0.92      0.90       118

    accuracy                           0.85       500
   macro avg       0.85      0.85      0.85       500
weighted avg       0.85      0.85      0.85       500
```

In [80]:

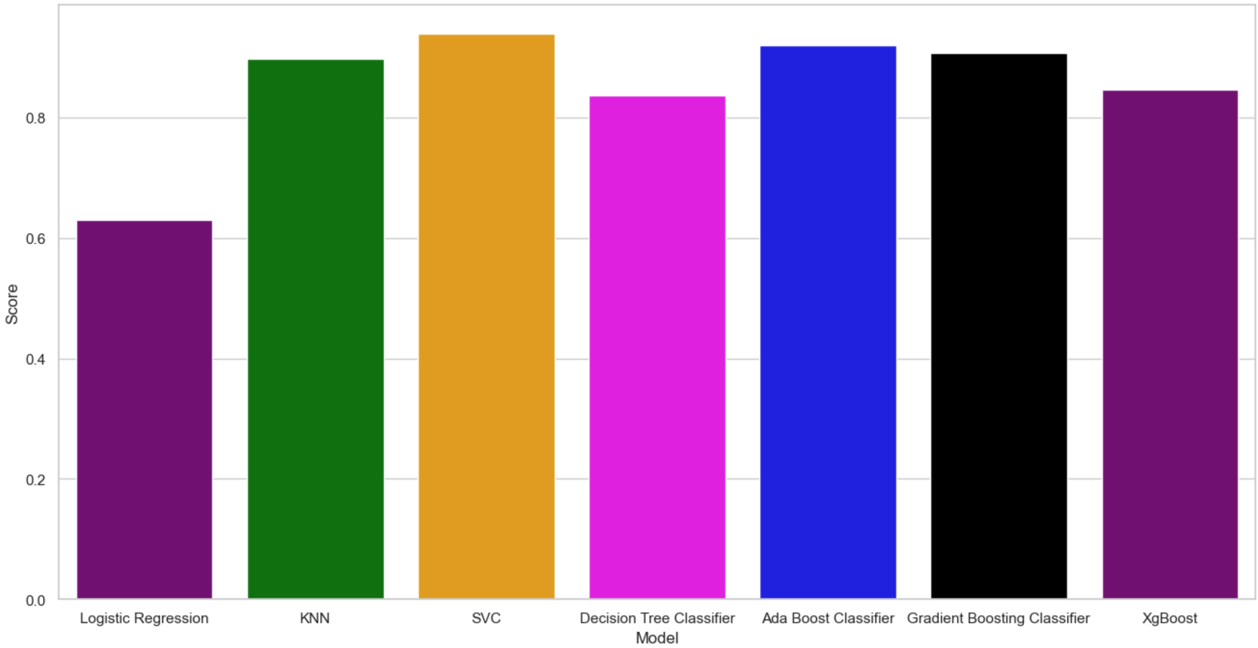
```
models = pd.DataFrame({
    'Model' : ['Logistic Regression', 'KNN', 'SVC', 'Decision Tree Classifier', 'Ada Boost Classifier', 'Gradient Boosting Classifier', 'X
    'Score' : [lr_acc, knn_acc, svc_acc, dtc_test_acc, ada_test_acc, gbc_acc, xgb_acc]
})
models.sort_values(by = 'Score', ascending = False)
```

Out[80]:

	Model	Score
2	SVC	0.940
4	Ada Boost Classifier	0.920
5	Gradient Boosting Classifier	0.908
1	KNN	0.898
6	XgBoost	0.846
3	Decision Tree Classifier	0.836
0	Logistic Regression	0.630

In [86]:

```
colors = ["purple", "green", "orange", "magenta", "blue", "black"]
sns.set_style("whitegrid")
plt.figure(figsize=(16,8))
plt.ylabel("Accuracy %")
plt.xlabel("Algorithms")
sns.barplot(x=models['Model'],y=models['Score'], palette=colors)
plt.show()
```



In [ ]:

```
CONCLUSION: we successfully Created a classification model to predict whether price range of mobile
based on certain specifications
```