Import Libraries

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score

from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
```

Load Training Dataset

In [4]:
```
df=pd.read_csv("train.csv")
df.head()
```

Out[4]:

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

5 rows × 21 columns

Data Exploration

In [6]:
```
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 [7]:
```
df.describe()
```

Out[7]:

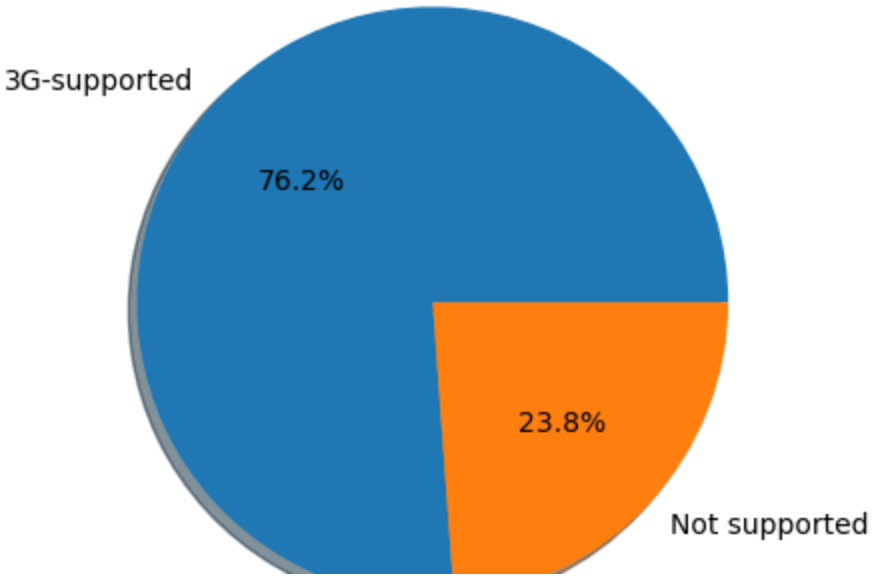
	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_ı
count	2000.000000	2000.0000	2000.000000	2000.000000	2000.000000	2000.000000	2000
mean	1238.518500	0.4950	1.522250	0.509500	4.309500	0.521500	32
std	439.418206	0.5001	0.816004	0.500035	4.341444	0.499662	18
min	501.000000	0.0000	0.500000	0.000000	0.000000	0.000000	2
25%	851.750000	0.0000	0.700000	0.000000	1.000000	0.000000	16
50%	1226.000000	0.0000	1.500000	1.000000	3.000000	1.000000	32
75%	1615.250000	1.0000	2.200000	1.000000	7.000000	1.000000	48
max	1998.000000	1.0000	3.000000	1.000000	19.000000	1.000000	64

8 rows × 21 columns

Data Visualizations

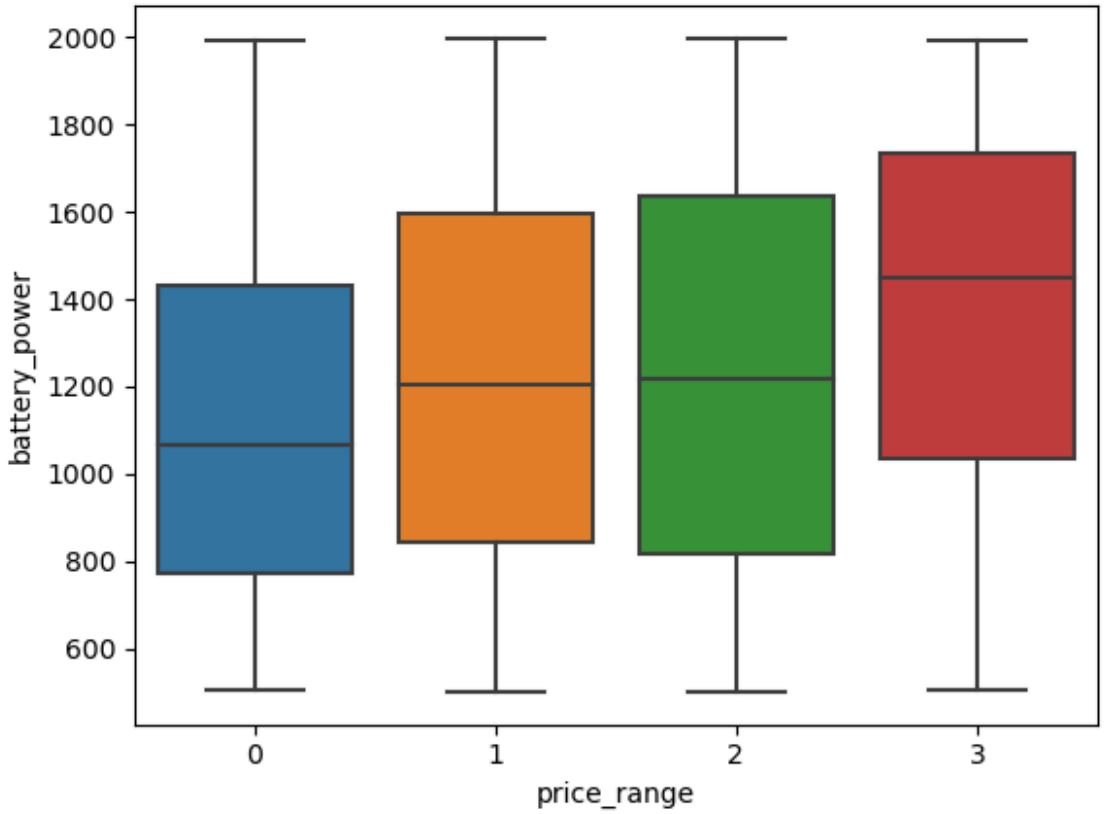
In [9]:

```
#percentage of phone with 3g techonology support
labels = ["3G-supported",'Not supported']
values=df['three_g'].value_counts().values
fig1, ax1 = plt.subplots()
ax1.pie(values, labels=labels, autopct='%1.1f%%',shadow=True)
plt.show()


#percetage of phone with 4g support
labels4g = ["4G-supported",'Not supported']
values4g = df['four_g'].value_counts().values
fig1, ax1 = plt.subplots()
ax1.pie(values4g, labels=labels4g, autopct='%1.1f%%',shadow=True)
plt.show()
```



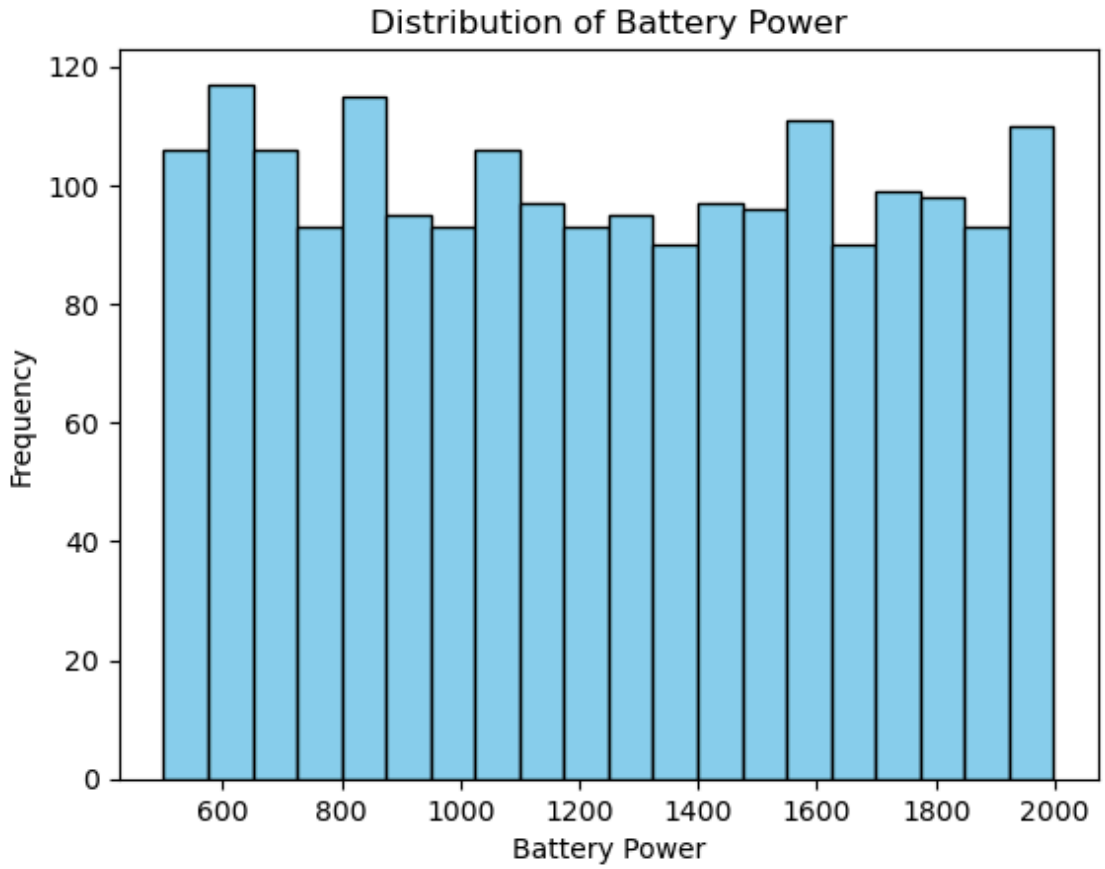
In [10]:
```
#battery power vs pecrecentage

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

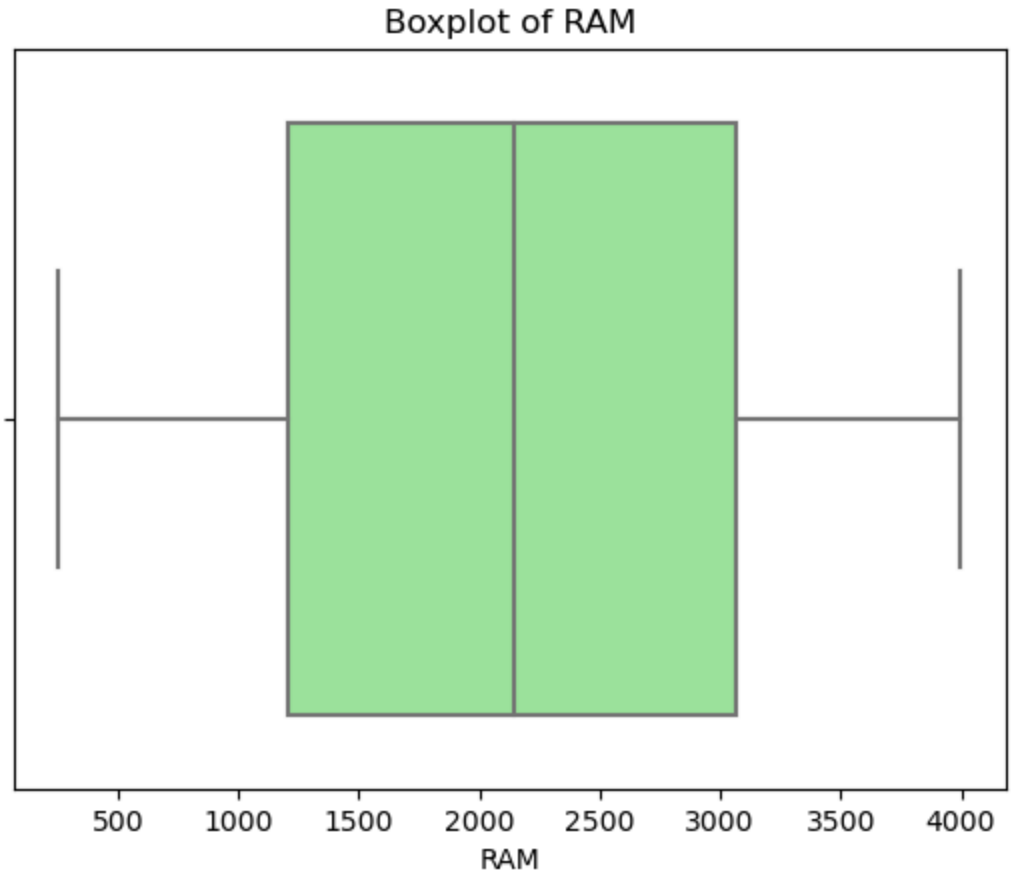
Out[10]: <AxesSubplot:xlabel='price_range', ylabel='battery_power'>



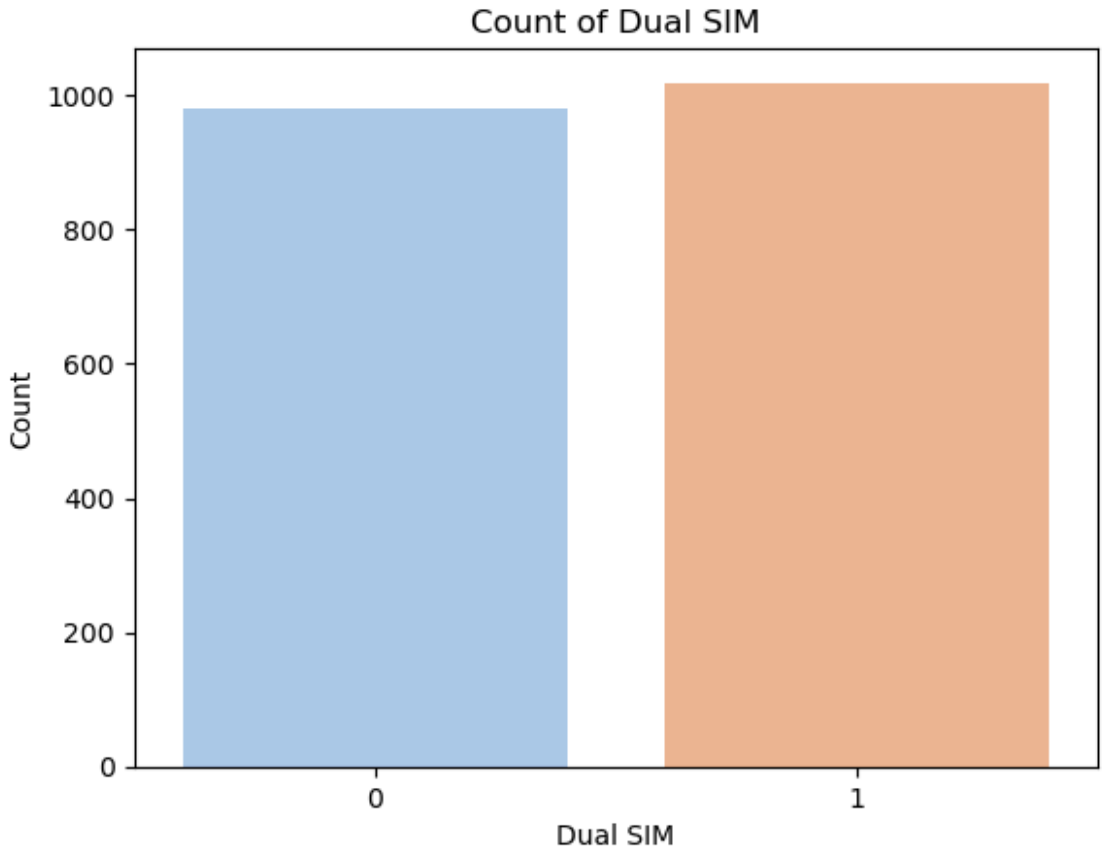
In [11]:

```
# battery_power
plt.hist(df['battery_power'], bins=20, color='skyblue', edgecolor='black')
plt.xlabel('Battery Power')
plt.ylabel('Frequency')
plt.title('Distribution of Battery Power')
plt.show()



# ram
sns.boxplot(x='ram', data=df, color='lightgreen')
plt.xlabel('RAM')
plt.title('Boxplot of RAM')
plt.show()
```



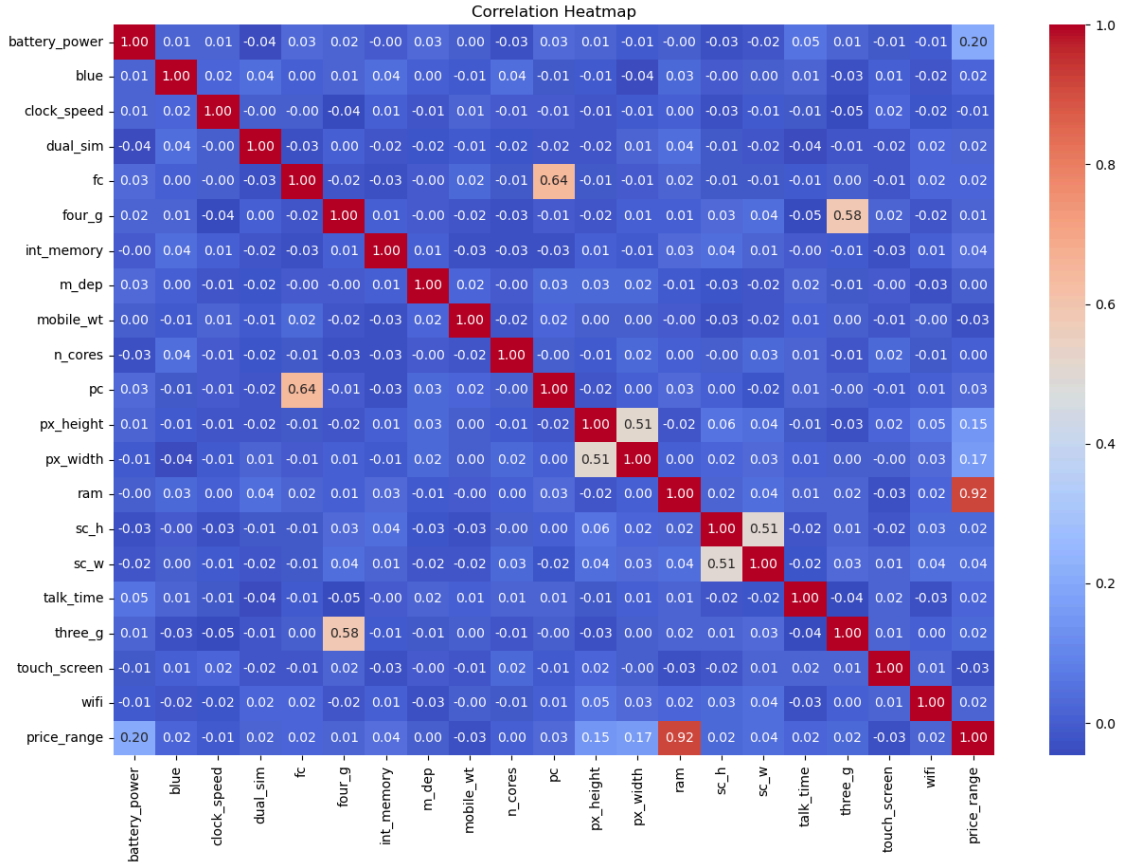

Boxplot of RAM
500
1000
1500
2000
2500
3000
3500
4000
RAM

In [12]:
```
# dual_sim
sns.countplot(x='dual_sim', data=df, palette='pastel')
plt.xlabel('Dual SIM')
plt.ylabel('Count')
plt.title('Count of Dual SIM')
plt.show()
```

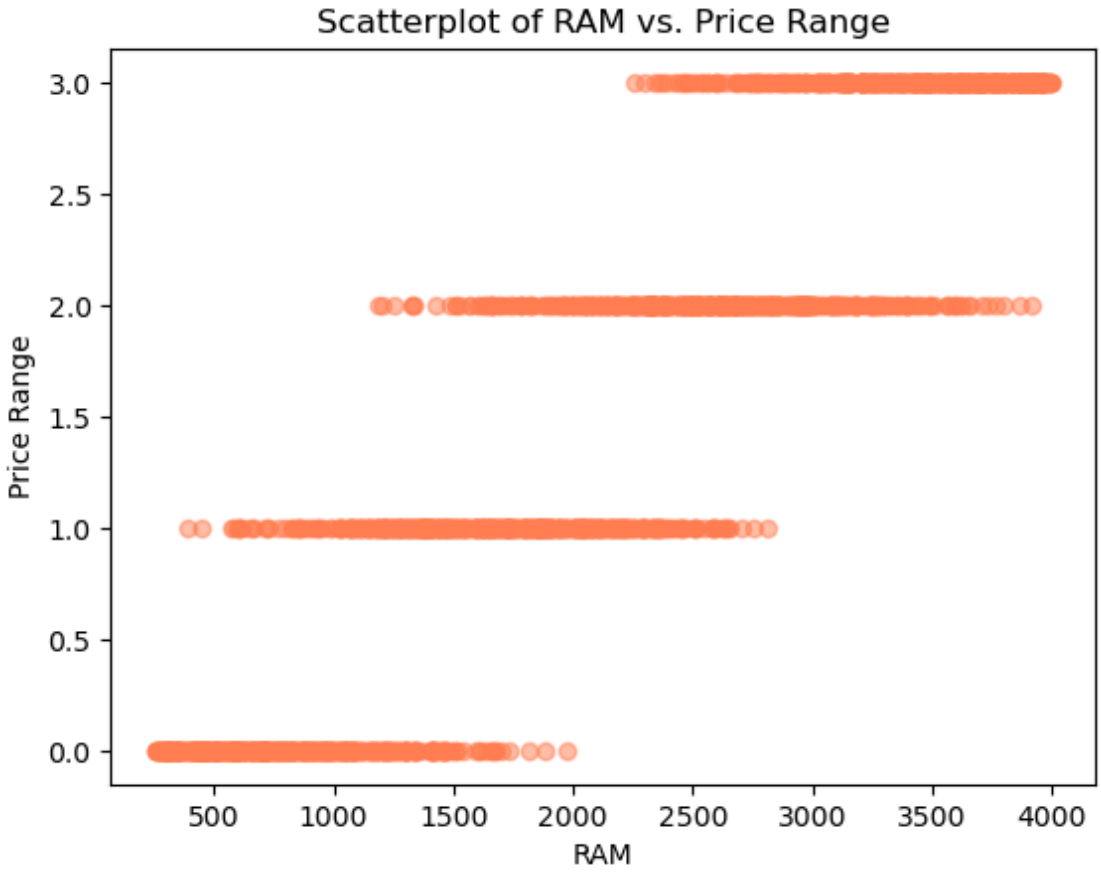


In [13]:

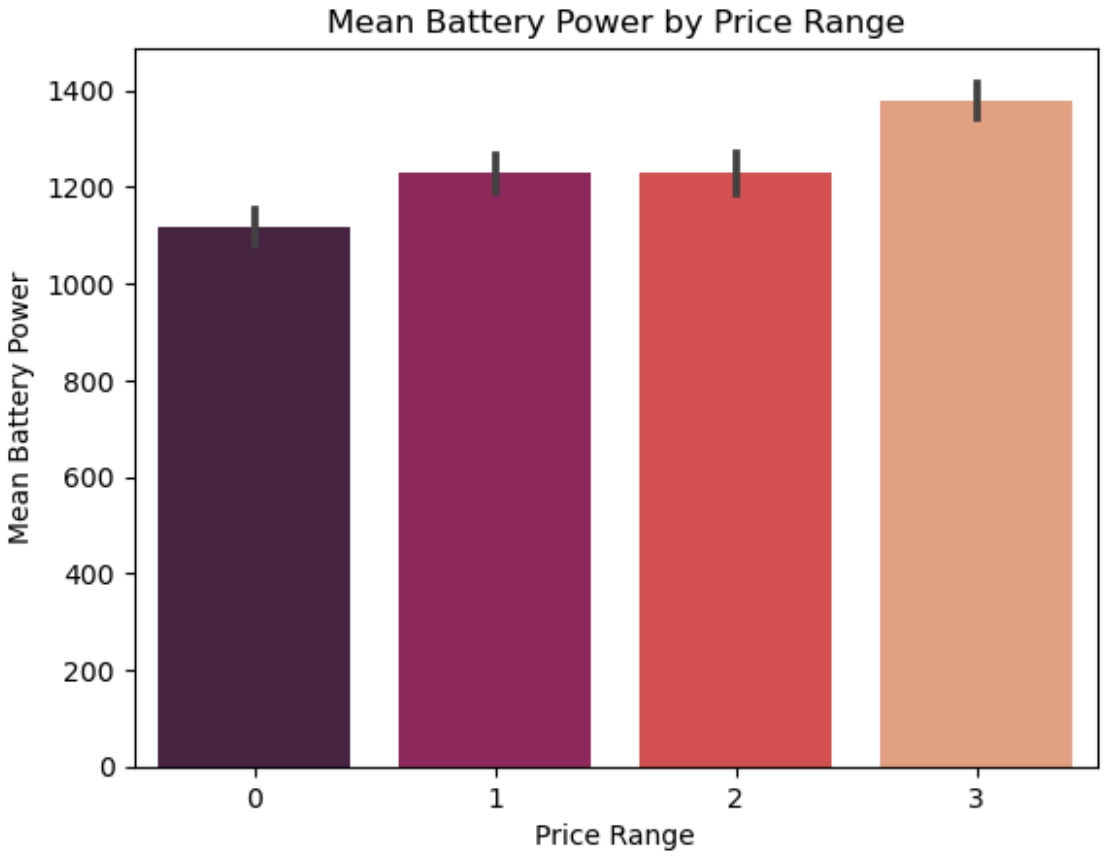
```
# correlation heatmap
corr = df.corr()
plt.figure(figsize=(15, 10))
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap')
plt.show()
```



In [14]:
```
# ram vs. price_range
plt.scatter(df['ram'], df['price_range'], color='coral', alpha=0.5)
plt.xlabel('RAM')
plt.ylabel('Price Range')
plt.title('Scatterplot of RAM vs. Price Range')
plt.show()
```

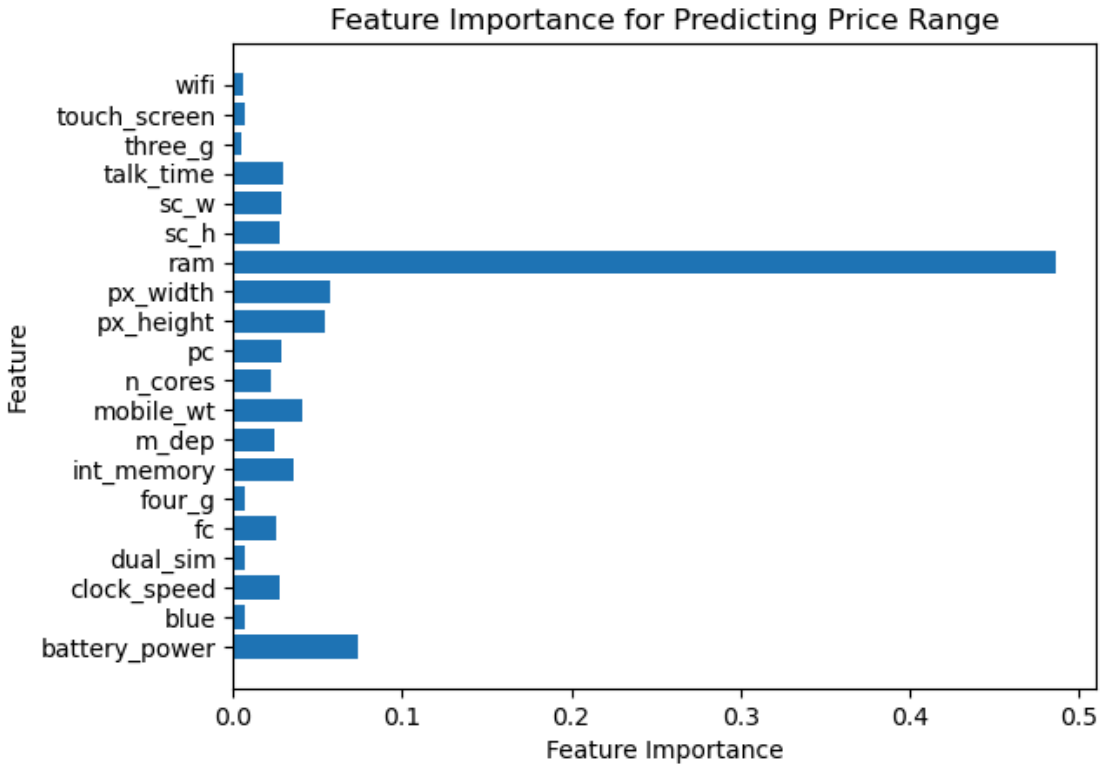


In [15]:
```
# price_range
sns.barplot(x='price_range', y='battery_power', data=df, palette='rocket')
plt.xlabel('Price Range')
plt.ylabel('Mean Battery Power')
plt.title('Mean Battery Power by Price Range')
plt.show()
```

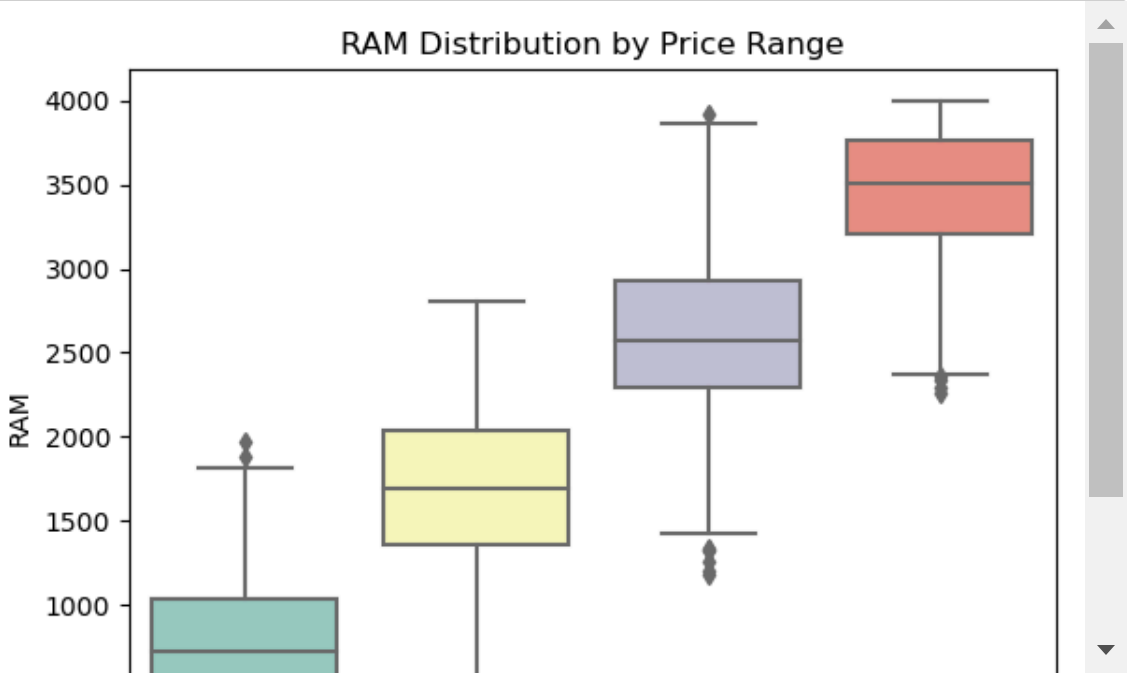


In [16]:

```
# Fit a random forest classifier
rf = RandomForestClassifier(n_estimators=100)
rf.fit(df.drop('price_range', axis=1), df['price_range'])

# Get feature importances
feature_importances = rf.feature_importances_

# Plot feature importances
plt.barh(df.drop('price_range', axis=1).columns, feature_importances)
plt.xlabel('Feature Importance')
plt.ylabel('Feature')
plt.title('Feature Importance for Predicting Price Range')
plt.show()
```



In [17]:
```
# RAM by price range
sns.boxplot(x='price_range', y='ram', data=df, palette='Set3')
plt.xlabel('Price Range')
plt.ylabel('RAM')
plt.title('RAM Distribution by Price Range')
plt.show()
```



In [18]:
```
#splitting data into training and testing
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=0.33,
```

First model

In [19]:
```
lm = LinearRegression()
lm.fit(X_train,y_train)
a=lm.score(X_test,y_test)
print(a)
```

```
0.9132801488185277
```

Second model

In [20]:
```
knn = KNeighborsRegressor(n_neighbors=10)
knn.fit(X_train,y_train)
y_pred=knn.predict(X_test)
b=knn.score(X_test, y_test)
print(b)
```

```
0.9504435639194675
```

Third model

In [21]:
```
rfc = RandomForestClassifier(n_estimators=200)
rfc.fit(X_train, y_train)
c=rfc.score(X_test,y_test)
print(c)
```

```
0.8742424242424243
```

Fourth model

In [22]:
```
logmodel = LogisticRegression()
logmodel.fit(X_train,y_train)
d=logmodel.score(X_test,y_test)
print(d)
```

```
0.6181818181818182
```

Test Data

In [23]:
```
df1=pd.read_csv("test.csv")
df1.head()
df1 = df1.drop('id', axis=1)
```

In [24]:
```
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=0.30,
# here we split the test data set so we can validate our results.
```

In [25]:
```
# Check if the 'predicted_price_range' column exists before trying to drop
if 'predicted_price_range' in df1.columns:
    df1 = df1.drop('predicted_price_range', axis=1)
```

In [26]:
```
# Since KNN provides the highest R2 score, we select that model to run on
knn = KNeighborsRegressor(n_neighbors=10)
knn.fit(X_train,y_train)
y_pred=knn.predict(X_test)
knn.score(X_test, y_test)
```

Out[26]:
```
0.9525209769253821
```

In [28]:
```
# Display only the predictions
print(df1['predicted_price_range'])
```

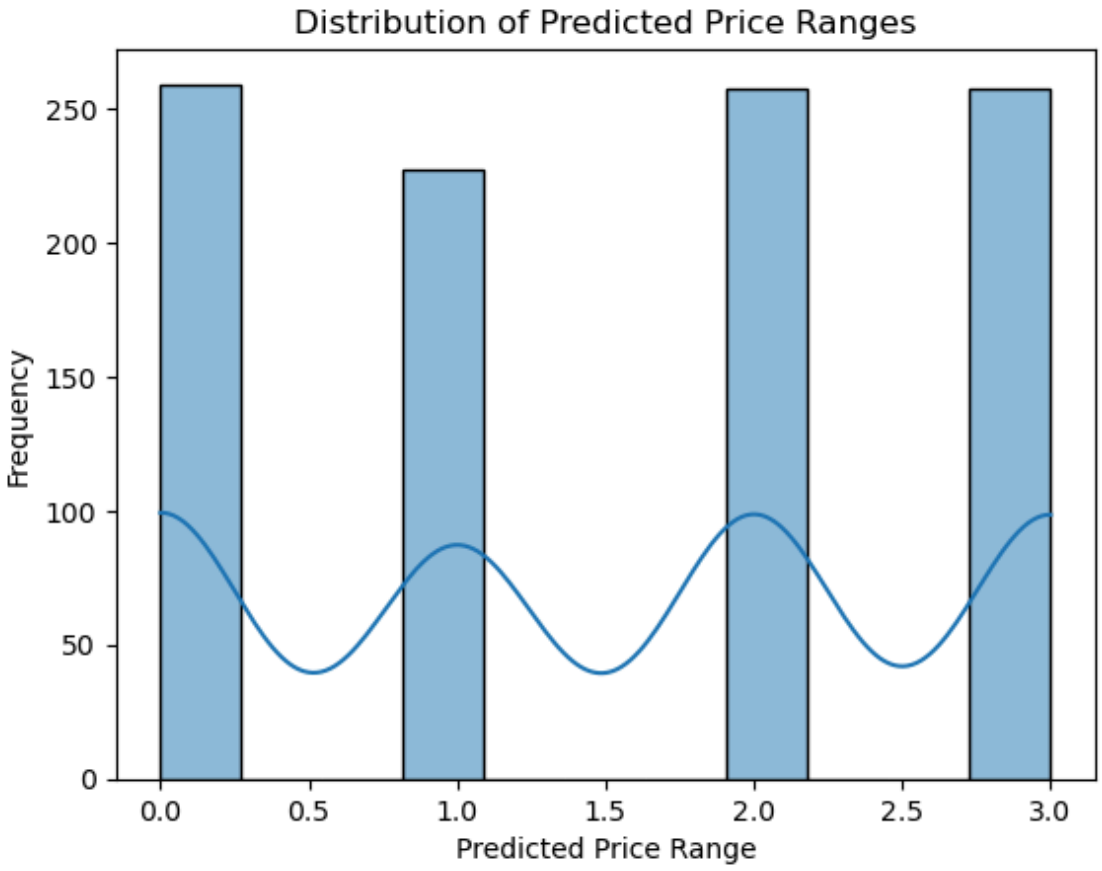
```
0      3
1      3
2      2
3      3
4      1
      ..
995    2
996    1
997    0
998    2
999    2
Name: predicted_price_range, Length: 1000, dtype: int32
```

In [29]:
```
print(df1['predicted_price_range'].describe())
```

```
count    1000.000000
mean        1.512000
std         1.132757
min         0.000000
25%         0.000000
50%         2.000000
75%         3.000000
max         3.000000
Name: predicted_price_range, dtype: float64
```

In [27]:
```
# Assuming df1 is your testing dataset and it is ready for prediction
y_test_pred = knn.predict(df1)

# Round the predicted values
y_test_pred_int = np.round(y_test_pred).astype(int)

# If you want to add these predictions back to df1 to see them next to the
df1['predicted_price_range'] = y_test_pred_int
```

In [30]:
```
sns.histplot(y_test_pred_int, kde=True)
plt.title('Distribution of Predicted Price Ranges')
plt.xlabel('Predicted Price Range')
plt.ylabel('Frequency')
plt.show()
```



In [31]:
```
print(df.columns)
print(df1.columns)
```

```
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')
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', 'predicted_price_range'],
      dtype='object')
```

In [32]:

```
print("df size:", df.shape)
print("df1 size:", df1.shape)
print(df.head())
print(df1.head())
# Randomly sample 1000 records from df1
df_reduced = df.sample(n=1000, random_state=42)
# Now you can use df1_reduced for your analysis or predictions
```

```
df size: (2000, 21)
df1 size: (1000, 21)
   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

   mobile_wt  n_cores  ...  px_height  px_width   ram  sc_h  sc_w  talk_t
ime  \
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

   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 rows x 21 columns]
   battery_power  blue  clock_speed  dual_sim  fc  four_g  int_memory  m_
dep  \
0           1043     1          1.8         1  14       0           5
0.1
1            841     1          0.5         1   4       1          61
0.8
2           1807     1          2.8         0   1       0          27
0.9
3           1546     0          0.5         1  18       1          25
0.5
4           1434     0          1.4         0  11       1          49
0.5

   mobile_wt  n_cores  ...  px_height  px_width   ram  sc_h  sc_w  talk_t
ime  \
0        193        3  ...        226      1412  3476    12     7
2
1        191        5  ...        746       857  3895     6     0
7
2        186        3  ...       1270      1366  2396    17    10
10
```

```
3          96        8  ...         295       1752  3893    10      0
7
4         108        6  ...         749        810  1773    15      8
7

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

[5 rows x 21 columns]
```

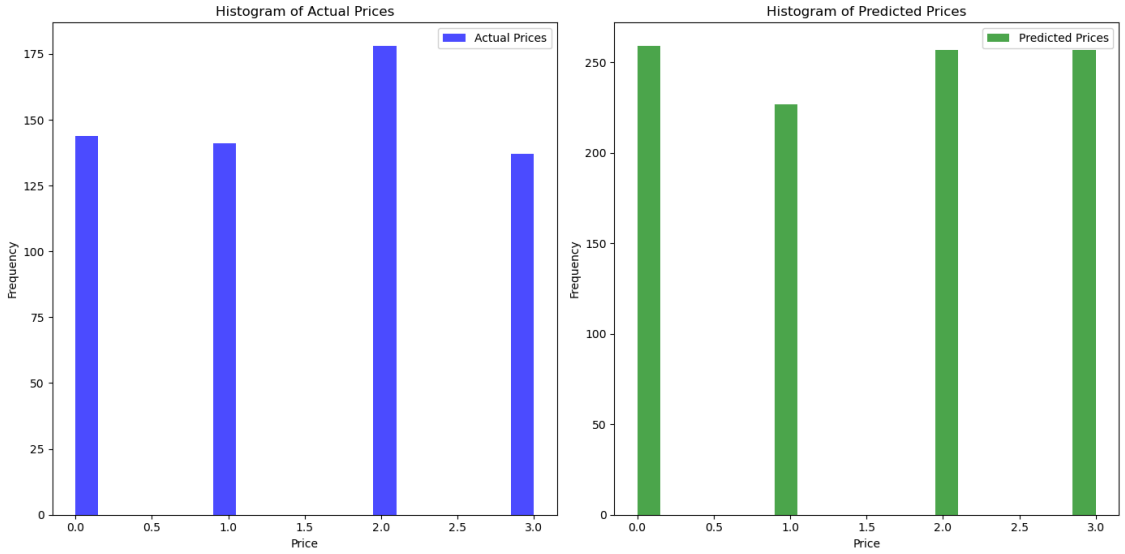
In [33]:

```
y_true =df_reduced['price_range']
y_pred =df1['predicted_price_range']

# Compute Mean Absolute Error
mae = mean_absolute_error(y_true, y_pred)
print("Mean Absolute Error:", mae)

# Compute R-squared
r2 = r2_score(y_true, y_pred)
print("R-squared:", r2)
```

```
Mean Absolute Error: 1.243
R-squared: -0.9789715269769907
```

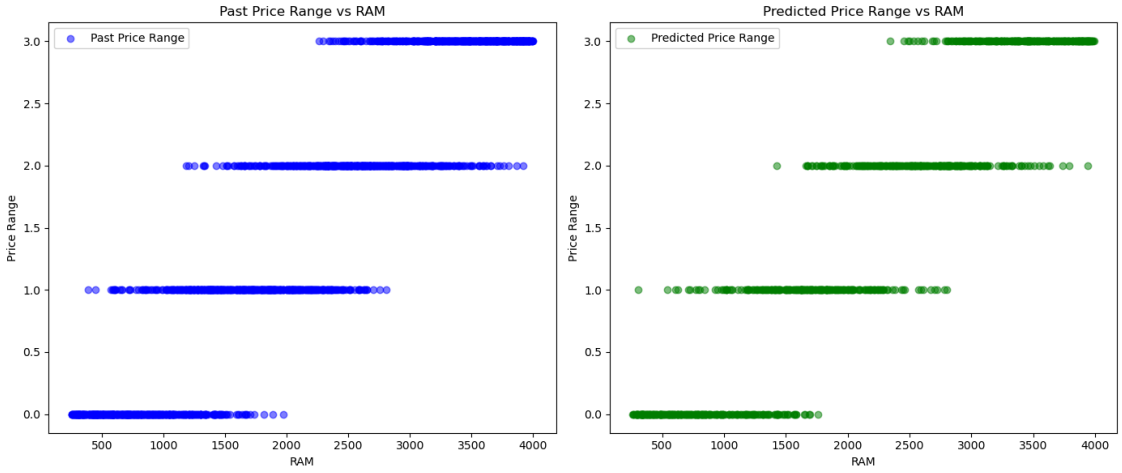
In [34]:

```
# Assuming y_test and y_pred are ready
plt.figure(figsize=(14, 7))

# Histogram for Actual Prices
plt.subplot(1, 2, 1)  # 1 row, 2 columns, 1st subplot
plt.hist(y_test, bins=20, color='blue', alpha=0.7, label='Actual Prices')
plt.title('Histogram of Actual Prices')
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.legend()

# Histogram for Predicted Prices
plt.subplot(1, 2, 2)  # 1 row, 2 columns, 2nd subplot
plt.hist(y_test_pred_int, bins=20, color='green', alpha=0.7, label='Predic
plt.title('Histogram of Predicted Prices')
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.legend()

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



In [35]:
```
# df1 includes 'ram', 'past_price_range', and 'predicted_price_range'
# Check the columns just to be sure
print(df1.columns)

plt.figure(figsize=(14, 6))

# Scatter plot for Past Price Range vs RAM
plt.subplot(1, 2, 1)  # 1 row, 2 columns, 1st subplot
plt.scatter(df['ram'], df['price_range'], color='blue', alpha=0.5, label='
plt.title('Past Price Range vs RAM')
plt.xlabel('RAM')
plt.ylabel('Price Range')
plt.legend()

# Scatter plot for Predicted Price Range vs RAM
plt.subplot(1, 2, 2)  # 1 row, 2 columns, 2nd subplot
plt.scatter(df1['ram'], df1['predicted_price_range'], color='green', alpha
plt.title('Predicted Price Range vs RAM')
plt.xlabel('RAM')
plt.ylabel('Price Range')
plt.legend()

plt.tight_layout()
plt.show()
```

```
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', 'predicted_price_range'],
      dtype='object')
```



In [ ]:

In [ ]:

In [ ]: