Mobile Price Classification With Machine Learning

Project : Mobile Price Classification

Group No : 50

Name : Neshadi Hirunika R.K

Registration No : EG/2020/4093

Name : Fernando H.T

Registration No : EG/2021/4507

INTRODUCTION

In today's highly competitive smartphone market, determining the right price for a mobile phone is crucial for both manufacturers and consumers. With numerous features and specifications available, finding a balance between functionality and cost has become a key factor in decision-making. The goal of this project is to build a machine learning model that can classify mobile phones into different price ranges based on their technical features. This can be particularly useful for manufacturers looking to position their products competitively or for customers trying to make informed purchase decisions.

By leveraging machine learning techniques, we will analyze various features such as battery power,clock speed,internal memory,mobile weight to predict the price category of a mobile phone. The insights gained from this project can help in making strategic decisions regarding pricing, feature inclusion, and product positioning in the market.

Through this project, we aim to develop a robust classification model that accurately predicts the price range of mobile phones, demonstrating the power of machine learning in real-world applications.

Dataset Link : https://www.kaggle.com/datasets/iabhishekofficial/mobile-price-classification

LITERATURE SURVEY

Mobile price classification has garnered significant attention as an application of machine learning in consumer electronics. Researchers have leveraged diverse datasets and algorithms to predict mobile phone price categories based on features like hardware specifications and performance metrics. Below, we discuss notable studies that have informed this field

Feature-Based Classification : A study by researchers using the Kaggle mobile price dataset explored the application of supervised machine learning algorithms such as Decision Trees, Random Forest, and K-Nearest Neighbors (KNN). It highlighted the importance of feature selection in achieving accurate classification. The Decision Tree algorithm achieved a 75.75% accuracy, while the Random Forest and KNN models reached higher accuracy levels of 87% and 92.75%, respectively. This study emphasized the role of feature engineering in improving model predictions

https://ceur-ws.org/Vol-3682/Paper5.pdf

https://ijisrt.com/assets/upload/files/IJISRT22JAN380.pdf

Comparative Analysis of Algorithms : Another study compared the performance of advanced classifiers, including Support Vector Machines (SVM) and Logistic Regression, alongside traditional algorithms like Naïve Bayes. The research showed that SVM performed particularly well due to its ability to handle complex decision boundaries, making it ideal for non-linear feature spaces. The study underlined the significance of hyperparameter tuning to optimize classifier performance

https://ieeexplore.ieee.org/document/9604550

Feature Engineering Techniques : Recent work on mobile price classification delved into feature engineering techniques to enhance data quality and relevance. Attributes such as pixel resolution, battery capacity, and RAM were identified as highly correlated with mobile prices. This study used correlation analysis and feature importance scoring to refine input data for better prediction accuracy. These methods contributed to an improvement in classification outcomes, as demonstrated in experiments using Random Forest and Gradient Boosting classifiers

https://ieeexplore.ieee.org/document/9604550

Real-World Application and Scalability : Practical implementations of price classification models have also been explored, showing their applicability in e-commerce and mobile retail industries. By predicting price categories based on user-selected features, such models aid consumers in decision-making while enabling businesses to optimize pricing strategies

https://ijisrt.com/assets/upload/files/IJISRT22JAN380.pdf

Feature Engineering for Mobile Price Prediction : Research has demonstrated the importance of feature selection in mobile price prediction. Studies by Karur and Balaje (2021) emphasize utilizing features like battery power, RAM, processor speed, camera resolution, and display size for accurate price classification. These features were found to have a significant correlation with mobile price categoriesSTM JOURNALS www.irjmets.com/uploadedfiles/paper/volume3/issue_6_june_2021/12265/1628083492.pdf

Application of Decision Trees and KNN : Decision trees and K-Nearest Neighbors (KNN) have been applied successfully in mobile price classification. A study by Singh et al. (2022) concluded that decision trees performed better for classification tasks on structured datasets like mobile prices. Meanwhile, KNN's simplicity is noted, although it often lags in performance for larger datasets www.irjmets.com/uploadedfiles/paper/volume3/issue_6_june_2021/12265/1628083492.pdf

Comparative Analysis of Machine Learning Algorithms : Suganyadevi et al. reviewed several classification algorithms, noting that logistic regression often outperformed methods like SVM and Random Forest in accuracy for mobile price datasets, particularly when the features were well-defined and normalized

https://journals.stmjournals.com/joaira/article=2024/view=155857/

www.irjmets.com/uploadedfiles/paper/volume3/issue_6_june_2021/12265/1628083492.pdf

Integration of Ensemble Learning Techniques : Recent studies have incorporated ensemble methods like Random Forests and Gradient Boosting Machines. These methods improve accuracy and generalization, especially on unbalanced datasets, as noted in studies like those by Dev et al. (2023)

https://journals.stmjournals.com/joaira/article=2024/view=155857/

Use of Deep Learning Approaches : Deep learning is emerging as a tool for mobile price prediction. Jose et al. (2023) highlighted the use of neural networks for capturing non-linear relationships in mobile price data. However, the computational cost and data requirements remain a challenge

https://journals.stmjournals.com/joaira/article=2024/view=155857/

www.irjmets.com/uploadedfiles/paper/volume3/issue_6_june_2021/12265/1628083492.pdf

Practical Implications for the Industry : Research by Asim and Khan (2018) showcased how mobile price predictions help manufacturers design budget-friendly products and enable customers to make informed buying decisions. This study reinforced the importance of accurate models for strategic pricing

https://journals.stmjournals.com/joaira/article=2024/view=155857/

www.irjmets.com/uploadedfiles/paper/volume3/issue_6_june_2021/12265/1628083492.pdf

Insights from Exploratory Data Analysis (EDA) : Analysis techniques such as correlation matrices and feature impact graphs provide essential insights into data distribution and relationships. For instance, higher RAM and internal memory were consistently linked with premium price ranges

www.irjmets.com/uploadedfiles/paper/volume3/issue_6_june_2021/12265/1628083492.pdf

DATASET DESCRIPTION

In the competitive mobile phone market, companies aim to analyze sales data to understand the factors influencing mobile phone prices. The goal is to identify the relationship between various features of a mobile phone (e.g., Battery Power, Internal Memory, Clock Speed etc.) and its price range. Instead of predicting the exact price, the focus is on classifying the phones into different price ranges based on their features (0-low price, 1- medium price, 2- high price and 3- very high price)

In our dataset altogether 20 features are available with 7 categorical features and 13 numerical features.Instead of 20 features there is a categorical feature called Price_Range which is the target variable (feature) of our dataset.Altogether 2000 enties are included for the dataset

Features Description

id : ID

battery_power : Total energy a battery can store in one time measured in mAh

blue : Has bluetooth or not

clock_speed : Speed at which microprocessor executes instructions dual_sim : Has dual sim support or not

fc : Front Camera mega pixels

four_g : Has 4G or not

int_memory : Internal Memory in Gigabytes

m_dep : Mobile Depth in cm

mobile_wt : Weight of mobile phone

n_cores : Number of cores of processor

pc : Primary Camera mega pixels

px_height : Pixel Resolution Height

px_width : Pixel Resolution Width

ram : Random Access Memory in Megabytes

sc_h : Screen Height of mobile in cm

sc_w : Screen Width of mobile in cm

talk_time : Longest time that a single battery charge will last when you are

three_g : Has 3G or not

touch_screen : Has touch screen or not wifi : Has wifi or not

price_range (Target)

0 : low cost

1 : medium cost

2 : high cost

3 : very high cost

Mount Gogle Drive

In [2]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Importing Libraries

In [3]:
```
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
```

Importing Dataset

In [4]:
```
#Load data as a pandas dataframe
df = pd.read_csv("/content/drive/MyDrive/MobilePriceClassification/train.csv"
df.head() # To display the first five rows in data set
```

Out[4]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mob
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 rows × 21 columns

Exploratory Data Analysis (EDA)

In [5]:
```
# Print the shape of the dataframe
df.shape
```

Out[5]:
```
(2000, 21)
```

In [6]:
```
#Print a concise summary of the pandas dataframe
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]:

```
# Generate descriptive analytics for the numerical features in the dataset

df.describe()
```

Out[7]:

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

8 rows × 21 columns

Data Visualization

Train-Test Split

In [8]:

```
# Train test split
from sklearn.model_selection import train_test_split
#Seperate the independent and dependent variables
X=df.drop('price_range',axis=1) # axis=1 means drop the coloumn 'price_range'
y=df.price_range
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.20,random_st
```

In [9]:

```
# Print number of training data points
print(X_train.shape)
```

```
(1600, 20)
```

In [10]:

```
# Print number of testing data points
print(X_test.shape)
```

```
(400, 20)
```

In [11]:

```
# Define categorical and numerical list
categorical = []
numerical = []

# loop for
for i in X_train.columns  :
    if X_train[i].nunique () > 8 :
        numerical.append (i)
    else :
        categorical.append (i)
print ("Categorical variables : " , categorical)
print ("numerical variables : " , numerical)
```

```
Categorical variables :  ['blue', 'dual_sim', 'four_g', 'n_cores', 'three_g',
'touch_screen', 'wifi']
numerical variables :  ['battery_power', 'clock_speed', 'fc', 'int_memory', 'm
_dep', 'mobile_wt', 'pc', 'px_height', 'px_width', 'ram', 'sc_h', 'sc_w', 'tal
k_time']
```

In [12]:

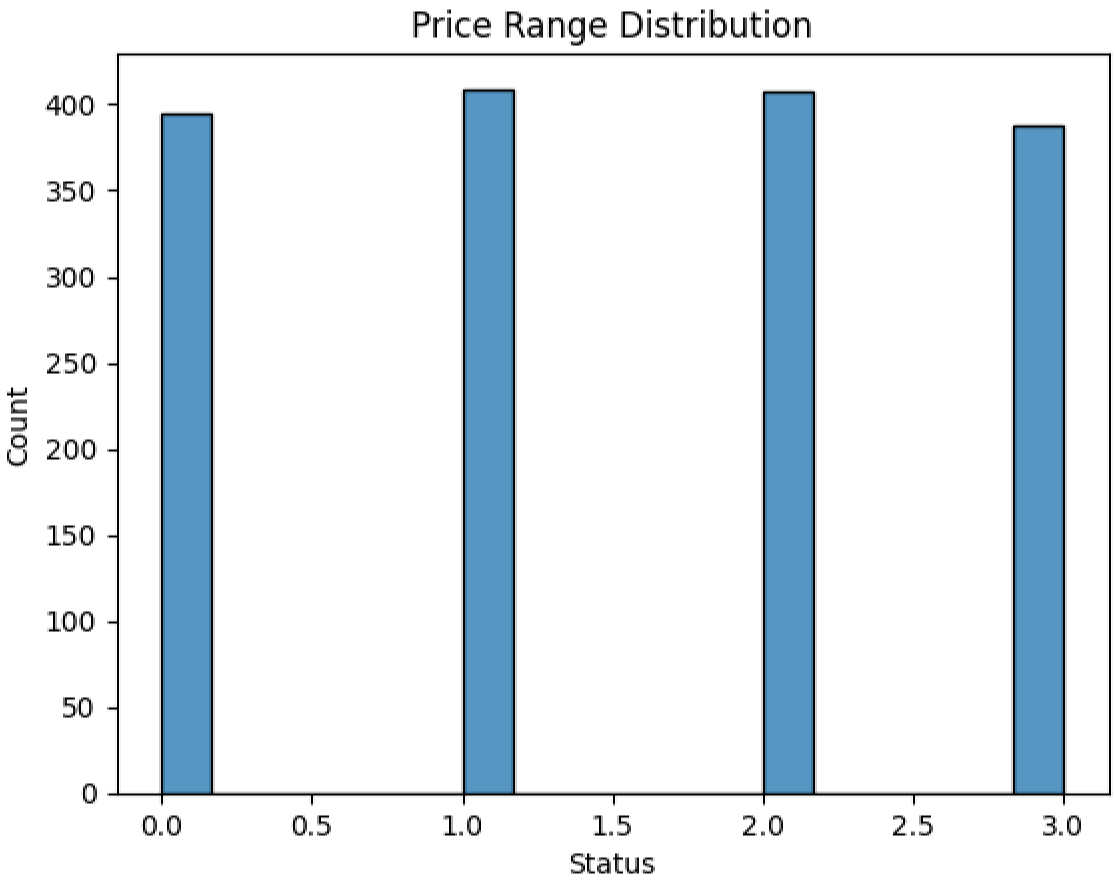
```
#Describe the characteristics of the features in the train test
X_train.describe()
```

Out[12]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g
count	1600.000000	1600.000000	1600.000000	1600.000000	1600.000000	1600.00000
mean	1240.808750	0.490625	1.513625	0.515000	4.310000	0.5225
std	440.727396	0.500068	0.820189	0.499931	4.339288	0.4996
min	501.000000	0.000000	0.500000	0.000000	0.000000	0.0000
25%	852.000000	0.000000	0.675000	0.000000	1.000000	0.0000
50%	1231.000000	0.000000	1.500000	1.000000	3.000000	1.0000
75%	1619.000000	1.000000	2.225000	1.000000	7.000000	1.0000
max	1998.000000	1.000000	3.000000	1.000000	19.000000	1.0000

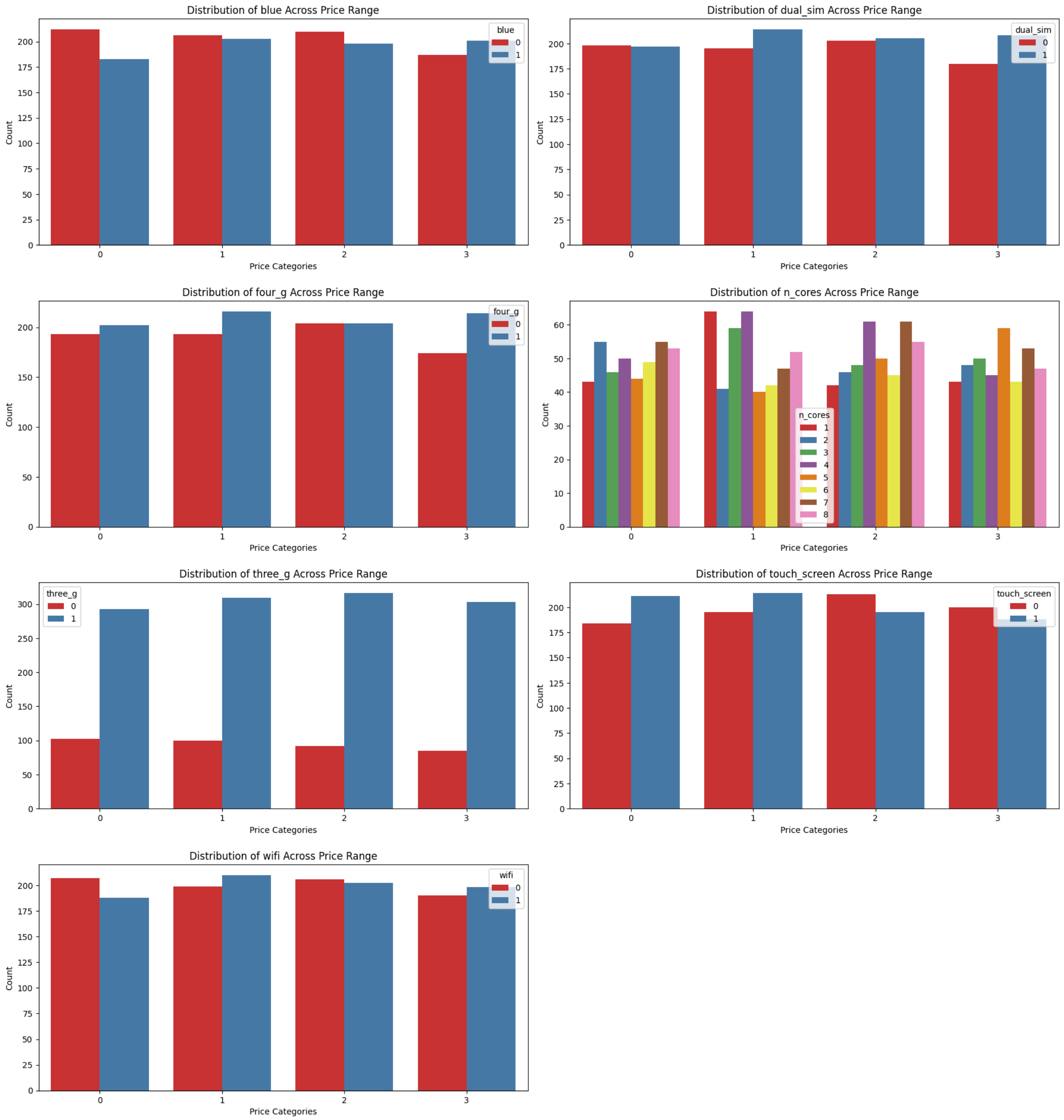
In [13]:

```
# Print the counts of status (the target variable)

sns.histplot(x=y_train)
plt.title('Price Range Distribution')
plt.xlabel('Status')
plt.ylabel('Count')
plt.show()
```

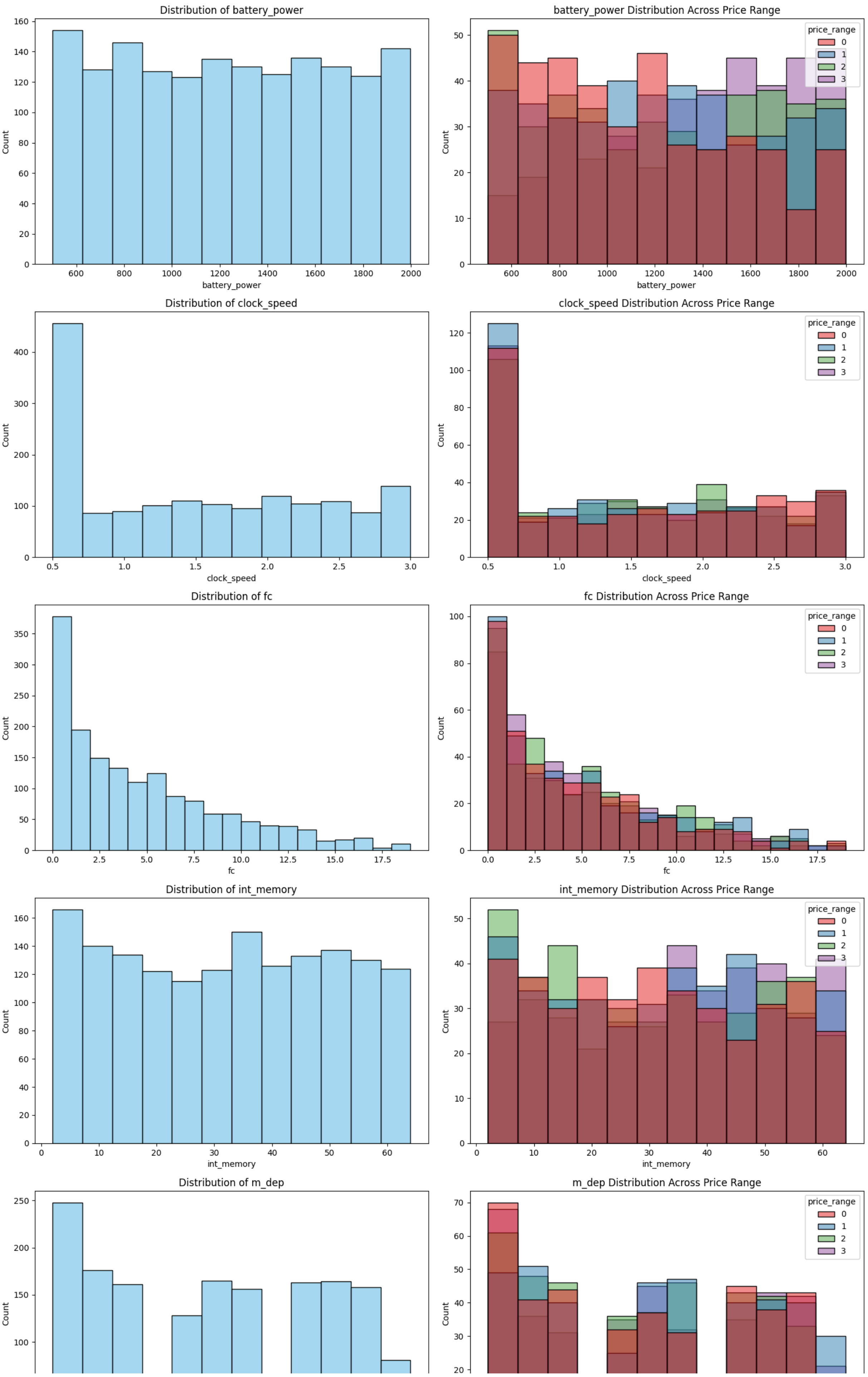


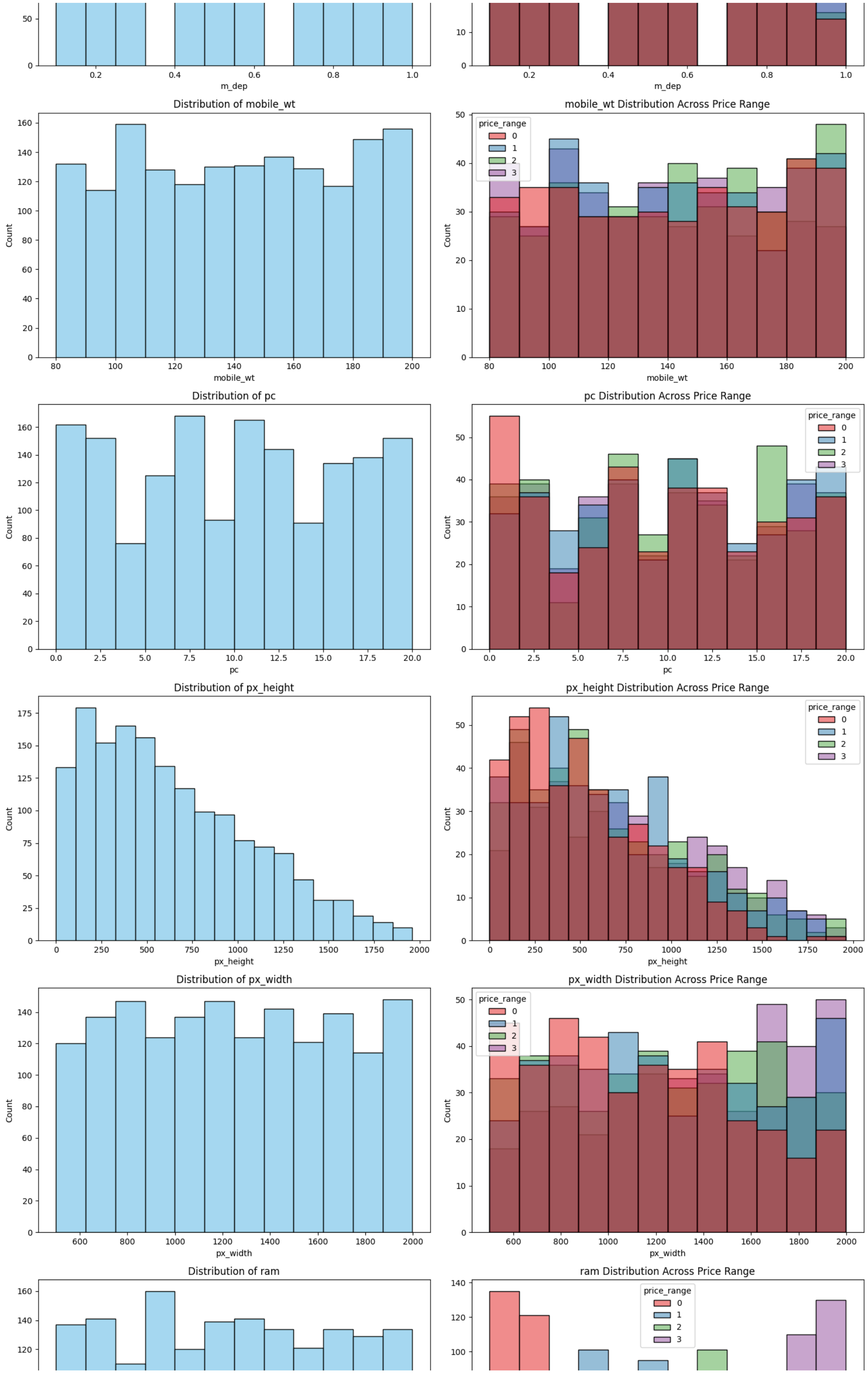
In [14]:

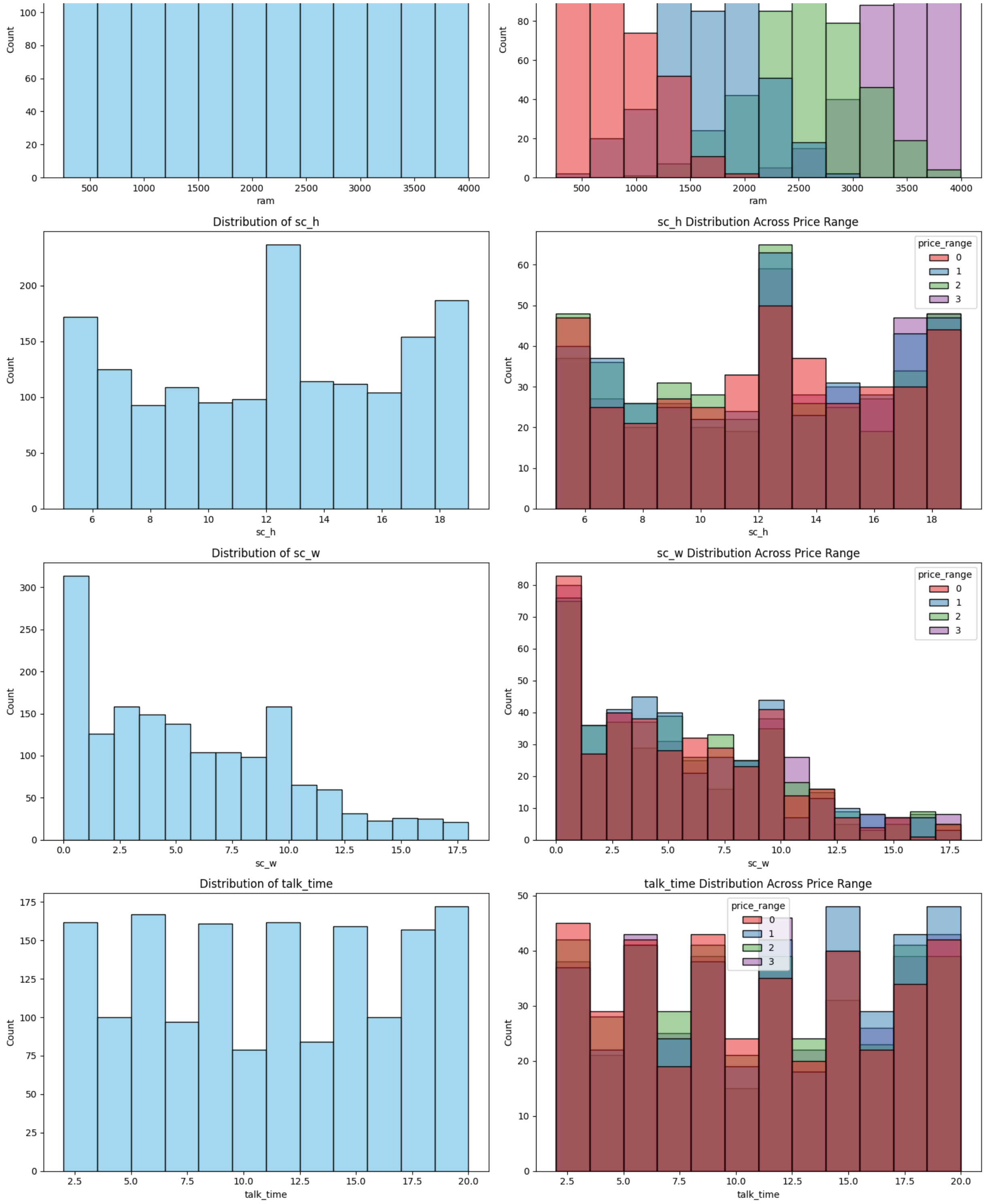
```
import matplotlib.pyplot as plt
import seaborn as sns

categorical_columns = X_train[categorical]

# Combine the target variable with the features DataFrame for visualization
X_train_combined = X_train.copy()
X_train_combined['price_range'] = y_train  # Assuming y_train contains the ta

# Set the size of the plots
plt.figure(figsize=(18, 28))

# Loop through each categorical column and create a count plot based on price
for i, col in enumerate(categorical_columns, 1):
    plt.subplot(6, 2, i)  # Adjust the grid size (6 rows, 2 columns)
    sns.countplot(data=X_train_combined, x='price_range', hue=col, palette='S
    plt.title(f'Distribution of {col} Across Price Range', fontsize=12)  # Ad
    plt.xlabel('Price Categories', fontsize=10)
    plt.ylabel('Count', fontsize=10)

plt.tight_layout(h_pad=2)  # Adjusted horizontal padding
plt.show()
```



In [15]:
```
import matplotlib.pyplot as plt
import seaborn as sns

numerical_columns = ['battery_power', 'clock_speed', 'fc', 'int_memory', 'm_d
                     'pc', 'px_height', 'px_width', 'ram', 'sc_h', 'sc_w', 't

# Combine the target variable with the features DataFrame for visualization
X_train_combined = X_train.copy()
X_train_combined['price_range'] = y_train

# Create subplots dynamically
n_features = len(numerical_columns)
fig, ax = plt.subplots(n_features, 2, figsize=(15, n_features * 5))  # Adjust

# Define a custom palette
custom_palette = sns.color_palette("husl", len(X_train_combined['price_range'
# Loop through each numerical column
for i, col in enumerate(numerical_columns):
    sns.histplot(x=col, data=X_train_combined, ax=ax[i, 0], color="skyblue")
    sns.histplot(x=col, hue='price_range', data=X_train_combined, ax=ax[i, 1]

    # Set titles and labels for clarity
    ax[i, 0].set(title=f'Distribution of {col}', xlabel=col, ylabel='Count')
    ax[i, 1].set(title=f'{col} Distribution Across Price Range', xlabel=col,

# Adjust layout to ensure no overlap
plt.tight_layout()
plt.show()
```


Distribution of battery_power
Count
0
20
40
60
80
100
120
140
160
600
800
1000
1200
1400
1600
1800
2000
battery_power
battery_power Distribution Across Price Range
price_range
0
1
2
3
Count
0
10
20
30
40
50
battery_power
Distribution of clock_speed
Count
0
100
200
300
400
0.5
1.0
1.5
2.0
2.5
3.0
clock_speed
clock_speed Distribution Across Price Range
Count
0
20
40
60
80
100
120
clock_speed
Distribution of fc
Count
0
50
100
150
200
250
300
350
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
fc
fc Distribution Across Price Range
Count
0
20
40
60
80
100
fc
Distribution of int_memory
Count
0
20
40
60
80
100
120
140
160
0
10
20
30
40
50
60
int_memory
int_memory Distribution Across Price Range
Count
0
10
20
30
40
50
int_memory
Distribution of m_dep
Count
100
150
200
250
m_dep Distribution Across Price Range
Count
20
30
40
50
60
70


m_dep
Distribution of mobile_wt
mobile_wt Distribution Across Price Range
Distribution of pc
pc Distribution Across Price Range
Distribution of px_height
px_height Distribution Across Price Range
Distribution of px_width
px_width Distribution Across Price Range
Distribution of ram
ram Distribution Across Price Range
Count
mobile_wt
pc
px_height
px_width
price_range
0
1
2
3



Missing/Null Values

In [16]:
```
X_train.isnull().sum() # Check whether is there any null value indicates in t
```

Out[16]:

	0
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

dtype: int64

In [17]:

```
X_test.isnull().sum() # Check whether is there any null value indicates in te
```

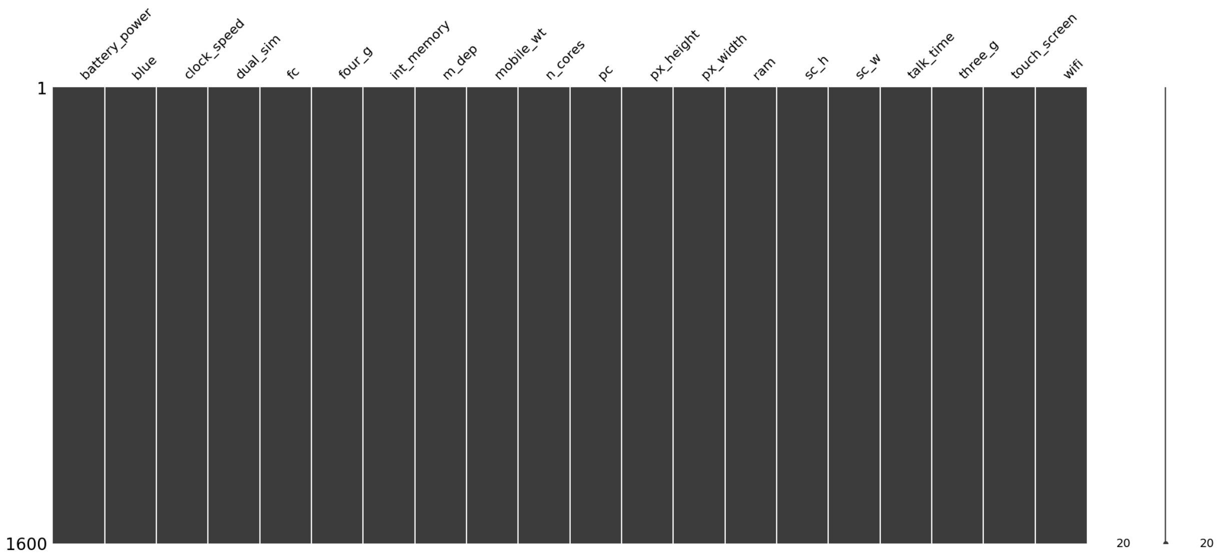
Out[17]:

	0
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

dtype: int64

In [18]:

```
# Display the missing values in the train set using matrix plot
import missingno as msno
msno.matrix(X_train)
plt.show()
```



There is no any missing/null values in both training and testing set.Therefore, Data preprocessing for missing/null values is not applicable

Duplicate Records

In [19]:
```
#Check whether is there any duplicated records for training set
X_train.duplicated()
X_train.duplicated().sum() # Check the sum of duplicated records
```

Out[19]:
```
0
```

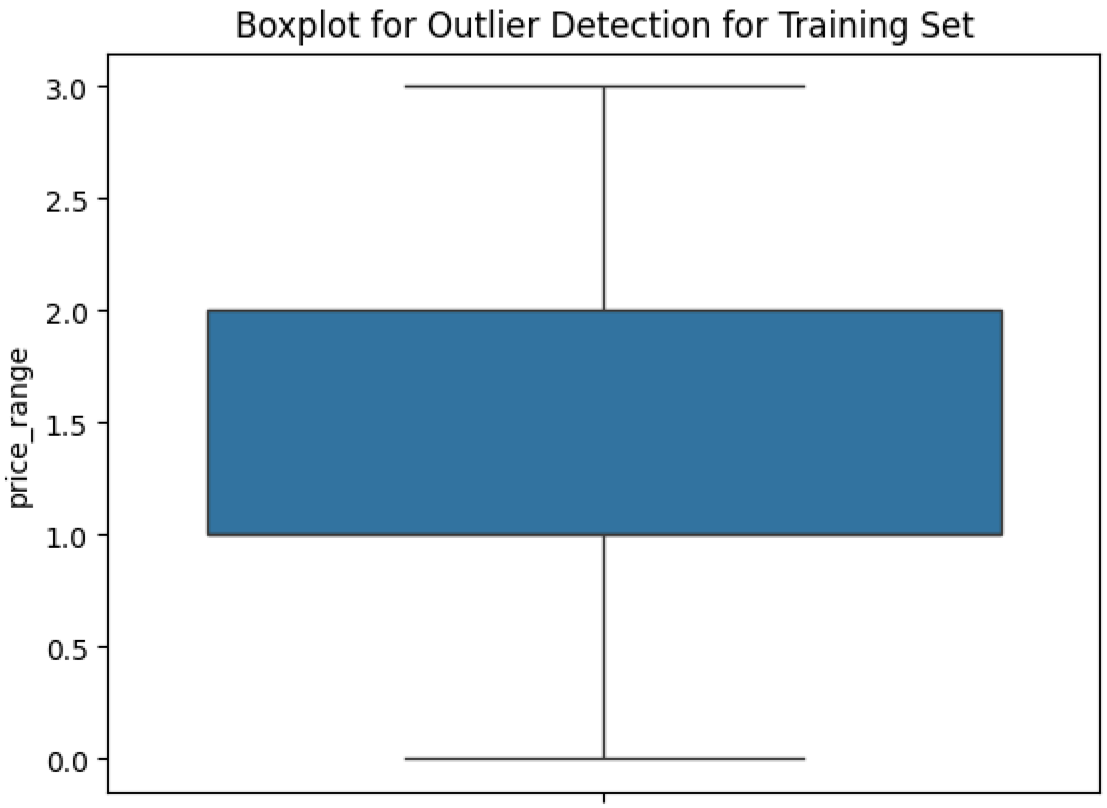
In [20]:
```
#Check whether is there any duplicated records for testing set
X_test.duplicated()
X_test.duplicated().sum() # Check the sum of duplicated records
```

Out[20]:
```
0
```

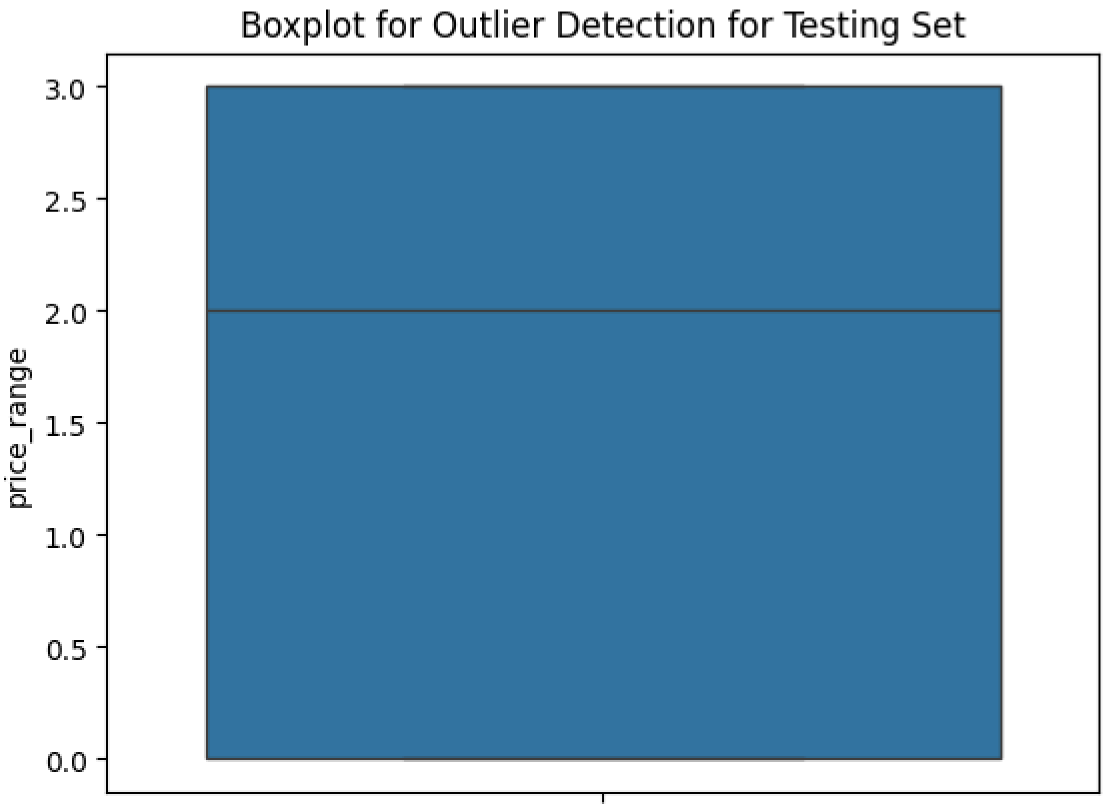
It is obvious that there is no any duplicated records for both training and testing data set.Therefore, Data preprocessing for duplicated record is not applicable

Identify Outliers

In [21]:
```
import numpy as np
y_train = np.squeeze(y_train)

sns.boxplot(data=X_train, y=y_train) #Create Boxplot
plt.title("Boxplot for Outlier Detection for Training Set")
plt.show()
```



In [22]:
```
import numpy as np
y_test = np.squeeze(y_test)

sns.boxplot(data=X_test, y=y_test) #Create Boxplot
plt.title("Boxplot for Outlier Detection for Testing Set")
plt.show()
```

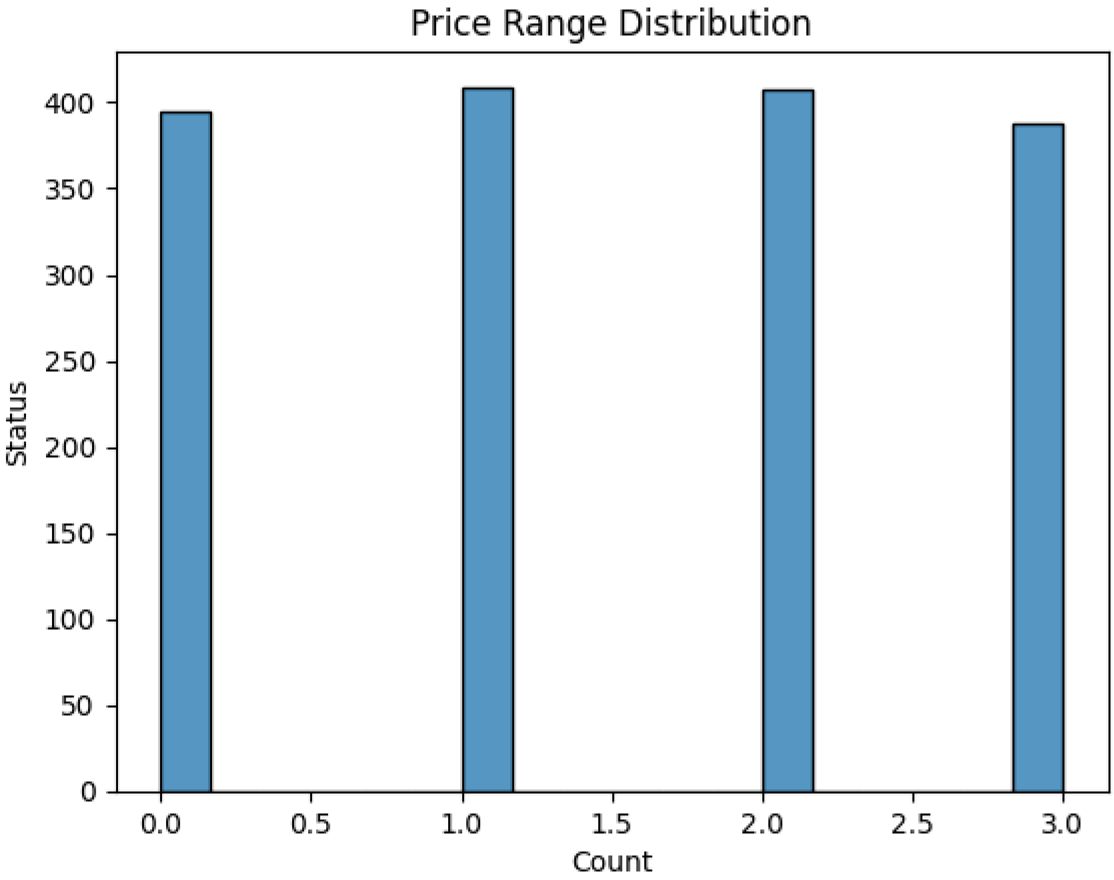


It is obvious that there is no any outliers for both training and testing data set.Therefore, Data preprocessing for outliers is not applicable

Identify Imbalanced Data Set

In [23]:

```
# Print the counts of status (the target variable) using seaborn countplot

sns.histplot(x=y_train)
plt.title('Price Range Distribution')
plt.xlabel('Count')
plt.ylabel('Status')
plt.show()
```



According to this graph,this not reflects an unequal distribution of classes within a dataset. Therfore, not need imbalanced dataset handling

Data Preprocessing

Handling Categorical Variables

In [24]:
```
# Encode the target variable in train and test sets

from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y_train = pd.DataFrame(data=y_train)
y_train['price_range'] = le.fit_transform(y_train['price_range'])
y_train = y_train.reset_index(drop=True)

le1 = LabelEncoder()
y_test = pd.DataFrame(data=y_test)
y_test['price_range'] = le1.fit_transform(y_test['price_range'])
y_test = y_test.reset_index(drop=True)
```

In [25]:
```python
# Print the encoded labels for the training set

for i in range(4):
    print(f'Label {i}: {le.classes_[i]}')
```

```
Label 0: 0
Label 1: 1
Label 2: 2
Label 3: 3
```

Feature Scaling for Numerical Features

In [26]:
```python
from sklearn.preprocessing import MinMaxScaler # Importing MinMaxScaler
numerical_variables = ['battery_power', 'clock_speed', 'fc', 'int_memory', 'm
minmax_scalar = MinMaxScaler()
X_train[numerical_variables]= minmax_scalar.fit_transform(X_train[numerical_v
X_test[numerical_variables]= minmax_scalar.fit_transform(X_test[numerical_var
```

In [27]:
```python
X_train.head()
```

Out[27]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m
968	0.949900	0	0.00	1	0.368421	0	0.709677	0.44
240	0.088176	1	0.68	0	0.000000	1	0.758065	0.00
819	0.490982	0	0.16	1	0.105263	1	0.887097	0.00
692	0.187041	0	0.24	0	0.105263	0	0.580645	0.33
420	0.637943	1	0.00	1	0.368421	0	0.080645	0.33

In [28]:
```python
X_test.head()
```

Out[28]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	r
1860	0.767270	0	0.80	0	0.166667	1	0.370968	0.5
353	0.456070	0	0.00	0	0.388889	1	0.096774	0.4
1333	0.985915	0	0.96	0	0.500000	0	0.193548	0.3
905	0.326626	1	0.60	0	0.222222	0	0.241935	0.1
1289	0.075788	1	0.00	1	0.388889	0	0.903226	0.4

Machine Learning Model Development:

Implementing the Randon Forest model

In [29]:
```
from sklearn.ensemble import RandomForestClassifier  # Importing RandomForest

rf = RandomForestClassifier(n_estimators=10, max_depth=5)  # Instance of Rand
rf.fit(X_train, y_train.values.ravel())  # Flatten y_train to 1D array if it'
```

Out[29]:
```
▾ RandomForestClassifier
RandomForestClassifier(max_depth=5, n_estimators=10)
```

Model Evaluation for Random Forest Classifier

In [30]:
```
accuracy = rf.score(X_train,y_train) # Print the accuracy of training dataset
print("Accuracy is",accuracy)
```

```
Accuracy is 0.8475
```

In [31]:
```
accuracy = rf.score(X_test,y_test) # Print the accuracy of testing dataset
print("Accuracy is",accuracy)
```

```
Accuracy is 0.755
```

Cross Validation for Random Forest Classifier

In [32]:
```
from sklearn.model_selection import cross_validate
from sklearn.ensemble import RandomForestClassifier

rf_cv = RandomForestClassifier(n_estimators=10, max_depth=5)  # Instance of R

# Perform cross-validation, ensuring y_train is 1D
cv_results = cross_validate(rf_cv, X_train, y_train.values.ravel(), scoring='

# Print the cross-validation scores
print(cv_results['test_score'])
```

```
[0.709375 0.75625  0.715625 0.73125  0.78125 ]
```

Cross Validated Model Evaluation

In [33]:
```
rf_cv.fit(X_train, y_train.values.ravel())  # Flatten y_train to 1D array if
```

Out[33]:
```
▾ RandomForestClassifier
RandomForestClassifier(max_depth=5, n_estimators=10)
```

In [34]:
```
rf_cv.score(X_train,y_train)  # Calculate and print the accuracy on the train
```

Out[34]:
```
0.8325
```

In [35]:
```
rf.score(X_test,y_test) # Calculate and print the accuracy on the testing dat
```

Out[35]:
```
0.755
```

Grid Search CV for Random Forest Classifier

In [36]:
```
from sklearn.model_selection import GridSearchCV
rfc = RandomForestClassifier()
# Define parameter grid
forest_params = [{'max_depth': list(range(10, 15)), 'max_features': list(rang

# GridSearchCV setup
clf = GridSearchCV(rfc, forest_params, cv=5, scoring='accuracy', verbose=3)

# Ensure the target variable is a 1D array
y_train_1d = y_train.values.ravel()  # Convert to 1D array

# Fit the GridSearchCV
clf.fit(X_train, y_train_1d)
```

```
Fitting 5 folds for each of 65 candidates, totalling 325 fits
[CV 1/5] END ......max_depth=10, max_features=1;, score=0.716 total time=   0.
4s
[CV 2/5] END ......max_depth=10, max_features=1;, score=0.756 total time=   0.
3s
[CV 3/5] END ......max_depth=10, max_features=1;, score=0.697 total time=   0.
3s
[CV 4/5] END ......max_depth=10, max_features=1;, score=0.669 total time=   0.
3s
[CV 5/5] END ......max_depth=10, max_features=1;, score=0.631 total time=   0.
3s
[CV 1/5] END ......max_depth=10, max_features=2;, score=0.781 total time=   0.
3s
[CV 2/5] END ......max_depth=10, max_features=2;, score=0.850 total time=   0.
3s
[CV 3/5] END ......max_depth=10, max_features=2;, score=0.809 total time=   0.
3s
[CV 4/5] END ......max_depth=10, max_features=2;, score=0.769 total time=   0.
3s
[CV 5/5] END ......max_depth=10, max_features=2;, score=0.794 total time=   0.
3s
[CV 1/5] END ......max_depth=10, max_features=3;, score=0.878 total time=   0.
4s
[CV 2/5] END ......max_depth=10, max_features=3;, score=0.866 total time=   0.
4s
[CV 3/5] END ......max_depth=10, max_features=3;, score=0.850 total time=   0.
4s
[CV 4/5] END ......max_depth=10, max_features=3;, score=0.831 total time=   0.
3s
[CV 5/5] END ......max_depth=10, max_features=3;, score=0.838 total time=   0.
4s
[CV 1/5] END ......max_depth=10, max_features=4;, score=0.875 total time=   0.
6s
[CV 2/5] END ......max_depth=10, max_features=4;, score=0.900 total time=   0.
6s
[CV 3/5] END ......max_depth=10, max_features=4;, score=0.853 total time=   0.
6s
[CV 4/5] END ......max_depth=10, max_features=4;, score=0.841 total time=   0.
6s
[CV 5/5] END ......max_depth=10, max_features=4;, score=0.844 total time=   0.
6s
[CV 1/5] END ......max_depth=10, max_features=5;, score=0.903 total time=   0.
7s
[CV 2/5] END ......max_depth=10, max_features=5;, score=0.909 total time=   0.
7s
[CV 3/5] END ......max_depth=10, max_features=5;, score=0.884 total time=   0.
7s
[CV 4/5] END ......max_depth=10, max_features=5;, score=0.859 total time=   0.
4s
[CV 5/5] END ......max_depth=10, max_features=5;, score=0.853 total time=   0.
4s
[CV 1/5] END ......max_depth=10, max_features=6;, score=0.878 total time=   0.
```

```
5s
[CV 2/5] END ......max_depth=10, max_features=6;, score=0.909 total time=   0.
5s
[CV 3/5] END ......max_depth=10, max_features=6;, score=0.897 total time=   0.
5s
[CV 4/5] END ......max_depth=10, max_features=6;, score=0.863 total time=   0.
5s
[CV 5/5] END ......max_depth=10, max_features=6;, score=0.853 total time=   0.
5s
[CV 1/5] END ......max_depth=10, max_features=7;, score=0.866 total time=   0.
5s
[CV 2/5] END ......max_depth=10, max_features=7;, score=0.903 total time=   0.
5s
[CV 3/5] END ......max_depth=10, max_features=7;, score=0.878 total time=   0.
5s
[CV 4/5] END ......max_depth=10, max_features=7;, score=0.856 total time=   0.
5s
[CV 5/5] END ......max_depth=10, max_features=7;, score=0.859 total time=   0.
5s
[CV 1/5] END ......max_depth=10, max_features=8;, score=0.897 total time=   0.
5s
[CV 2/5] END ......max_depth=10, max_features=8;, score=0.900 total time=   0.
5s
[CV 3/5] END ......max_depth=10, max_features=8;, score=0.897 total time=   0.
5s
[CV 4/5] END ......max_depth=10, max_features=8;, score=0.838 total time=   0.
5s
[CV 5/5] END ......max_depth=10, max_features=8;, score=0.869 total time=   0.
5s
[CV 1/5] END ......max_depth=10, max_features=9;, score=0.906 total time=   0.
6s
[CV 2/5] END ......max_depth=10, max_features=9;, score=0.909 total time=   0.
6s
[CV 3/5] END ......max_depth=10, max_features=9;, score=0.894 total time=   0.
6s
[CV 4/5] END ......max_depth=10, max_features=9;, score=0.872 total time=   0.
9s
[CV 5/5] END ......max_depth=10, max_features=9;, score=0.881 total time=   0.
8s
[CV 1/5] END .....max_depth=10, max_features=10;, score=0.891 total time=   0.
9s
[CV 2/5] END .....max_depth=10, max_features=10;, score=0.909 total time=   0.
9s
[CV 3/5] END .....max_depth=10, max_features=10;, score=0.884 total time=   0.
9s
[CV 4/5] END .....max_depth=10, max_features=10;, score=0.872 total time=   0.
8s
[CV 5/5] END .....max_depth=10, max_features=10;, score=0.875 total time=   0.
6s
[CV 1/5] END .....max_depth=10, max_features=11;, score=0.897 total time=   0.
6s
[CV 2/5] END .....max_depth=10, max_features=11;, score=0.909 total time=   0.
```

```
6s
[CV 3/5] END .....max_depth=10, max_features=11;, score=0.887 total time=   0.
6s
[CV 4/5] END .....max_depth=10, max_features=11;, score=0.869 total time=   0.
6s
[CV 5/5] END .....max_depth=10, max_features=11;, score=0.875 total time=   0.
6s
[CV 1/5] END .....max_depth=10, max_features=12;, score=0.891 total time=   0.
7s
[CV 2/5] END .....max_depth=10, max_features=12;, score=0.900 total time=   0.
7s
[CV 3/5] END .....max_depth=10, max_features=12;, score=0.881 total time=   0.
7s
[CV 4/5] END .....max_depth=10, max_features=12;, score=0.884 total time=   0.
7s
[CV 5/5] END .....max_depth=10, max_features=12;, score=0.878 total time=   0.
7s
[CV 1/5] END .....max_depth=10, max_features=13;, score=0.884 total time=   0.
7s
[CV 2/5] END .....max_depth=10, max_features=13;, score=0.903 total time=   0.
7s
[CV 3/5] END .....max_depth=10, max_features=13;, score=0.891 total time=   0.
7s
[CV 4/5] END .....max_depth=10, max_features=13;, score=0.872 total time=   0.
8s
[CV 5/5] END .....max_depth=10, max_features=13;, score=0.881 total time=   1.
0s
[CV 1/5] END ......max_depth=11, max_features=1;, score=0.706 total time=   0.
4s
[CV 2/5] END ......max_depth=11, max_features=1;, score=0.713 total time=   0.
4s
[CV 3/5] END ......max_depth=11, max_features=1;, score=0.706 total time=   0.
4s
[CV 4/5] END ......max_depth=11, max_features=1;, score=0.684 total time=   0.
5s
[CV 5/5] END ......max_depth=11, max_features=1;, score=0.641 total time=   0.
5s
[CV 1/5] END ......max_depth=11, max_features=2;, score=0.828 total time=   0.
5s
[CV 2/5] END ......max_depth=11, max_features=2;, score=0.847 total time=   0.
5s
[CV 3/5] END ......max_depth=11, max_features=2;, score=0.794 total time=   0.
4s
[CV 4/5] END ......max_depth=11, max_features=2;, score=0.791 total time=   0.
3s
[CV 5/5] END ......max_depth=11, max_features=2;, score=0.787 total time=   0.
4s
[CV 1/5] END ......max_depth=11, max_features=3;, score=0.859 total time=   0.
7s
[CV 2/5] END ......max_depth=11, max_features=3;, score=0.884 total time=   0.
4s
[CV 3/5] END ......max_depth=11, max_features=3;, score=0.875 total time=   0.
```

```
4s
[CV 4/5] END ......max_depth=11, max_features=3;, score=0.825 total time=   0.
4s
[CV 5/5] END ......max_depth=11, max_features=3;, score=0.816 total time=   0.
4s
[CV 1/5] END ......max_depth=11, max_features=4;, score=0.869 total time=   0.
4s
[CV 2/5] END ......max_depth=11, max_features=4;, score=0.900 total time=   0.
4s
[CV 3/5] END ......max_depth=11, max_features=4;, score=0.875 total time=   0.
4s
[CV 4/5] END ......max_depth=11, max_features=4;, score=0.838 total time=   0.
4s
[CV 5/5] END ......max_depth=11, max_features=4;, score=0.844 total time=   0.
4s
[CV 1/5] END ......max_depth=11, max_features=5;, score=0.878 total time=   0.
4s
[CV 2/5] END ......max_depth=11, max_features=5;, score=0.916 total time=   0.
4s
[CV 3/5] END ......max_depth=11, max_features=5;, score=0.866 total time=   0.
4s
[CV 4/5] END ......max_depth=11, max_features=5;, score=0.853 total time=   0.
5s
[CV 5/5] END ......max_depth=11, max_features=5;, score=0.859 total time=   0.
4s
[CV 1/5] END ......max_depth=11, max_features=6;, score=0.881 total time=   0.
5s
[CV 2/5] END ......max_depth=11, max_features=6;, score=0.916 total time=   0.
5s
[CV 3/5] END ......max_depth=11, max_features=6;, score=0.894 total time=   0.
5s
[CV 4/5] END ......max_depth=11, max_features=6;, score=0.866 total time=   0.
5s
[CV 5/5] END ......max_depth=11, max_features=6;, score=0.869 total time=   0.
5s
[CV 1/5] END ......max_depth=11, max_features=7;, score=0.887 total time=   0.
7s
[CV 2/5] END ......max_depth=11, max_features=7;, score=0.912 total time=   0.
7s
[CV 3/5] END ......max_depth=11, max_features=7;, score=0.887 total time=   0.
7s
[CV 4/5] END ......max_depth=11, max_features=7;, score=0.872 total time=   0.
7s
[CV 5/5] END ......max_depth=11, max_features=7;, score=0.875 total time=   0.
8s
[CV 1/5] END ......max_depth=11, max_features=8;, score=0.884 total time=   0.
9s
[CV 2/5] END ......max_depth=11, max_features=8;, score=0.912 total time=   0.
7s
[CV 3/5] END ......max_depth=11, max_features=8;, score=0.881 total time=   0.
6s
[CV 4/5] END ......max_depth=11, max_features=8;, score=0.872 total time=   0.
```

```
6s
[CV 5/5] END ......max_depth=11, max_features=8;, score=0.866 total time=   0.
5s
[CV 1/5] END ......max_depth=11, max_features=9;, score=0.894 total time=   0.
6s
[CV 2/5] END ......max_depth=11, max_features=9;, score=0.906 total time=   0.
6s
[CV 3/5] END ......max_depth=11, max_features=9;, score=0.887 total time=   0.
6s
[CV 4/5] END ......max_depth=11, max_features=9;, score=0.884 total time=   0.
6s
[CV 5/5] END ......max_depth=11, max_features=9;, score=0.878 total time=   0.
6s
[CV 1/5] END .....max_depth=11, max_features=10;, score=0.906 total time=   0.
6s
[CV 2/5] END .....max_depth=11, max_features=10;, score=0.897 total time=   0.
6s
[CV 3/5] END .....max_depth=11, max_features=10;, score=0.894 total time=   0.
6s
[CV 4/5] END .....max_depth=11, max_features=10;, score=0.869 total time=   0.
6s
[CV 5/5] END .....max_depth=11, max_features=10;, score=0.884 total time=   0.
6s
[CV 1/5] END .....max_depth=11, max_features=11;, score=0.903 total time=   0.
6s
[CV 2/5] END .....max_depth=11, max_features=11;, score=0.912 total time=   0.
6s
[CV 3/5] END .....max_depth=11, max_features=11;, score=0.894 total time=   0.
6s
[CV 4/5] END .....max_depth=11, max_features=11;, score=0.881 total time=   0.
9s
[CV 5/5] END .....max_depth=11, max_features=11;, score=0.894 total time=   0.
9s
[CV 1/5] END .....max_depth=11, max_features=12;, score=0.884 total time=   1.
0s
[CV 2/5] END .....max_depth=11, max_features=12;, score=0.903 total time=   1.
0s
[CV 3/5] END .....max_depth=11, max_features=12;, score=0.894 total time=   1.
0s
[CV 4/5] END .....max_depth=11, max_features=12;, score=0.878 total time=   0.
7s
[CV 5/5] END .....max_depth=11, max_features=12;, score=0.884 total time=   0.
7s
[CV 1/5] END .....max_depth=11, max_features=13;, score=0.900 total time=   0.
7s
[CV 2/5] END .....max_depth=11, max_features=13;, score=0.894 total time=   0.
7s
[CV 3/5] END .....max_depth=11, max_features=13;, score=0.891 total time=   0.
7s
[CV 4/5] END .....max_depth=11, max_features=13;, score=0.875 total time=   0.
7s
[CV 5/5] END .....max_depth=11, max_features=13;, score=0.878 total time=   0.
```

```
7s
[CV 1/5] END ......max_depth=12, max_features=1;, score=0.684 total time=   0.
3s
[CV 2/5] END ......max_depth=12, max_features=1;, score=0.759 total time=   0.
3s
[CV 3/5] END ......max_depth=12, max_features=1;, score=0.706 total time=   0.
3s
[CV 4/5] END ......max_depth=12, max_features=1;, score=0.697 total time=   0.
3s
[CV 5/5] END ......max_depth=12, max_features=1;, score=0.644 total time=   0.
3s
[CV 1/5] END ......max_depth=12, max_features=2;, score=0.822 total time=   0.
3s
[CV 2/5] END ......max_depth=12, max_features=2;, score=0.828 total time=   0.
3s
[CV 3/5] END ......max_depth=12, max_features=2;, score=0.803 total time=   0.
3s
[CV 4/5] END ......max_depth=12, max_features=2;, score=0.762 total time=   0.
3s
[CV 5/5] END ......max_depth=12, max_features=2;, score=0.772 total time=   0.
3s
[CV 1/5] END ......max_depth=12, max_features=3;, score=0.856 total time=   0.
4s
[CV 2/5] END ......max_depth=12, max_features=3;, score=0.884 total time=   0.
4s
[CV 3/5] END ......max_depth=12, max_features=3;, score=0.853 total time=   0.
4s
[CV 4/5] END ......max_depth=12, max_features=3;, score=0.850 total time=   0.
4s
[CV 5/5] END ......max_depth=12, max_features=3;, score=0.812 total time=   0.
4s
[CV 1/5] END ......max_depth=12, max_features=4;, score=0.866 total time=   0.
5s
[CV 2/5] END ......max_depth=12, max_features=4;, score=0.916 total time=   0.
6s
[CV 3/5] END ......max_depth=12, max_features=4;, score=0.872 total time=   0.
6s
[CV 4/5] END ......max_depth=12, max_features=4;, score=0.850 total time=   0.
6s
[CV 5/5] END ......max_depth=12, max_features=4;, score=0.850 total time=   0.
6s
[CV 1/5] END ......max_depth=12, max_features=5;, score=0.912 total time=   0.
7s
[CV 2/5] END ......max_depth=12, max_features=5;, score=0.903 total time=   0.
7s
[CV 3/5] END ......max_depth=12, max_features=5;, score=0.869 total time=   0.
6s
[CV 4/5] END ......max_depth=12, max_features=5;, score=0.863 total time=   0.
4s
[CV 5/5] END ......max_depth=12, max_features=5;, score=0.863 total time=   0.
5s
[CV 1/5] END ......max_depth=12, max_features=6;, score=0.897 total time=   0.
```

```
5s
[CV 2/5] END ......max_depth=12, max_features=6;, score=0.916 total time=   0.
5s
[CV 3/5] END ......max_depth=12, max_features=6;, score=0.900 total time=   0.
5s
[CV 4/5] END ......max_depth=12, max_features=6;, score=0.866 total time=   0.
5s
[CV 5/5] END ......max_depth=12, max_features=6;, score=0.859 total time=   0.
5s
[CV 1/5] END ......max_depth=12, max_features=7;, score=0.900 total time=   0.
5s
[CV 2/5] END ......max_depth=12, max_features=7;, score=0.900 total time=   0.
5s
[CV 3/5] END ......max_depth=12, max_features=7;, score=0.900 total time=   0.
5s
[CV 4/5] END ......max_depth=12, max_features=7;, score=0.863 total time=   0.
5s
[CV 5/5] END ......max_depth=12, max_features=7;, score=0.863 total time=   0.
5s
[CV 1/5] END ......max_depth=12, max_features=8;, score=0.897 total time=   0.
6s
[CV 2/5] END ......max_depth=12, max_features=8;, score=0.909 total time=   0.
6s
[CV 3/5] END ......max_depth=12, max_features=8;, score=0.894 total time=   0.
5s
[CV 4/5] END ......max_depth=12, max_features=8;, score=0.872 total time=   0.
6s
[CV 5/5] END ......max_depth=12, max_features=8;, score=0.887 total time=   0.
5s
[CV 1/5] END ......max_depth=12, max_features=9;, score=0.897 total time=   0.
6s
[CV 2/5] END ......max_depth=12, max_features=9;, score=0.909 total time=   0.
7s
[CV 3/5] END ......max_depth=12, max_features=9;, score=0.891 total time=   0.
9s
[CV 4/5] END ......max_depth=12, max_features=9;, score=0.872 total time=   0.
8s
[CV 5/5] END ......max_depth=12, max_features=9;, score=0.875 total time=   0.
8s
[CV 1/5] END .....max_depth=12, max_features=10;, score=0.897 total time=   0.
9s
[CV 2/5] END .....max_depth=12, max_features=10;, score=0.925 total time=   0.
9s
[CV 3/5] END .....max_depth=12, max_features=10;, score=0.894 total time=   0.
7s
[CV 4/5] END .....max_depth=12, max_features=10;, score=0.869 total time=   0.
6s
[CV 5/5] END .....max_depth=12, max_features=10;, score=0.881 total time=   0.
6s
[CV 1/5] END .....max_depth=12, max_features=11;, score=0.897 total time=   0.
6s
[CV 2/5] END .....max_depth=12, max_features=11;, score=0.919 total time=   0.
```

```
6s
[CV 3/5] END .....max_depth=12, max_features=11;, score=0.884 total time=   0.
7s
[CV 4/5] END .....max_depth=12, max_features=11;, score=0.887 total time=   0.
7s
[CV 5/5] END .....max_depth=12, max_features=11;, score=0.887 total time=   0.
7s
[CV 1/5] END .....max_depth=12, max_features=12;, score=0.906 total time=   0.
7s
[CV 2/5] END .....max_depth=12, max_features=12;, score=0.906 total time=   0.
7s
[CV 3/5] END .....max_depth=12, max_features=12;, score=0.891 total time=   0.
6s
[CV 4/5] END .....max_depth=12, max_features=12;, score=0.881 total time=   0.
7s
[CV 5/5] END .....max_depth=12, max_features=12;, score=0.887 total time=   0.
7s
[CV 1/5] END .....max_depth=12, max_features=13;, score=0.912 total time=   0.
7s
[CV 2/5] END .....max_depth=12, max_features=13;, score=0.906 total time=   0.
7s
[CV 3/5] END .....max_depth=12, max_features=13;, score=0.894 total time=   0.
9s
[CV 4/5] END .....max_depth=12, max_features=13;, score=0.881 total time=   1.
0s
[CV 5/5] END .....max_depth=12, max_features=13;, score=0.884 total time=   1.
0s
[CV 1/5] END ......max_depth=13, max_features=1;, score=0.741 total time=   0.
5s
[CV 2/5] END ......max_depth=13, max_features=1;, score=0.703 total time=   0.
5s
[CV 3/5] END ......max_depth=13, max_features=1;, score=0.703 total time=   0.
5s
[CV 4/5] END ......max_depth=13, max_features=1;, score=0.697 total time=   0.
5s
[CV 5/5] END ......max_depth=13, max_features=1;, score=0.659 total time=   0.
5s
[CV 1/5] END ......max_depth=13, max_features=2;, score=0.797 total time=   0.
4s
[CV 2/5] END ......max_depth=13, max_features=2;, score=0.816 total time=   0.
3s
[CV 3/5] END ......max_depth=13, max_features=2;, score=0.831 total time=   0.
3s
[CV 4/5] END ......max_depth=13, max_features=2;, score=0.797 total time=   0.
3s
[CV 5/5] END ......max_depth=13, max_features=2;, score=0.784 total time=   0.
3s
[CV 1/5] END ......max_depth=13, max_features=3;, score=0.847 total time=   0.
4s
[CV 2/5] END ......max_depth=13, max_features=3;, score=0.875 total time=   0.
4s
[CV 3/5] END ......max_depth=13, max_features=3;, score=0.869 total time=   0.
```

```
4s
[CV 4/5] END ......max_depth=13, max_features=3;, score=0.853 total time=   0.
4s
[CV 5/5] END ......max_depth=13, max_features=3;, score=0.819 total time=   0.
4s
[CV 1/5] END ......max_depth=13, max_features=4;, score=0.872 total time=   0.
4s
[CV 2/5] END ......max_depth=13, max_features=4;, score=0.900 total time=   0.
4s
[CV 3/5] END ......max_depth=13, max_features=4;, score=0.872 total time=   0.
4s
[CV 4/5] END ......max_depth=13, max_features=4;, score=0.847 total time=   0.
4s
[CV 5/5] END ......max_depth=13, max_features=4;, score=0.863 total time=   0.
4s
[CV 1/5] END ......max_depth=13, max_features=5;, score=0.878 total time=   0.
4s
[CV 2/5] END ......max_depth=13, max_features=5;, score=0.894 total time=   0.
4s
[CV 3/5] END ......max_depth=13, max_features=5;, score=0.891 total time=   0.
5s
[CV 4/5] END ......max_depth=13, max_features=5;, score=0.844 total time=   0.
4s
[CV 5/5] END ......max_depth=13, max_features=5;, score=0.872 total time=   0.
5s
[CV 1/5] END ......max_depth=13, max_features=6;, score=0.881 total time=   0.
5s
[CV 2/5] END ......max_depth=13, max_features=6;, score=0.903 total time=   0.
5s
[CV 3/5] END ......max_depth=13, max_features=6;, score=0.875 total time=   0.
5s
[CV 4/5] END ......max_depth=13, max_features=6;, score=0.866 total time=   0.
5s
[CV 5/5] END ......max_depth=13, max_features=6;, score=0.847 total time=   0.
6s
[CV 1/5] END ......max_depth=13, max_features=7;, score=0.872 total time=   0.
8s
[CV 2/5] END ......max_depth=13, max_features=7;, score=0.919 total time=   0.
8s
[CV 3/5] END ......max_depth=13, max_features=7;, score=0.881 total time=   0.
7s
[CV 4/5] END ......max_depth=13, max_features=7;, score=0.875 total time=   0.
8s
[CV 5/5] END ......max_depth=13, max_features=7;, score=0.872 total time=   0.
8s
[CV 1/5] END ......max_depth=13, max_features=8;, score=0.881 total time=   0.
7s
[CV 2/5] END ......max_depth=13, max_features=8;, score=0.906 total time=   0.
6s
[CV 3/5] END ......max_depth=13, max_features=8;, score=0.894 total time=   0.
5s
[CV 4/5] END ......max_depth=13, max_features=8;, score=0.887 total time=   0.
```

```
5s
[CV 5/5] END ......max_depth=13, max_features=8;, score=0.872 total time=   0.
6s
[CV 1/5] END ......max_depth=13, max_features=9;, score=0.916 total time=   0.
6s
[CV 2/5] END ......max_depth=13, max_features=9;, score=0.909 total time=   0.
6s
[CV 3/5] END ......max_depth=13, max_features=9;, score=0.887 total time=   0.
6s
[CV 4/5] END ......max_depth=13, max_features=9;, score=0.875 total time=   0.
6s
[CV 5/5] END ......max_depth=13, max_features=9;, score=0.881 total time=   0.
6s
[CV 1/5] END .....max_depth=13, max_features=10;, score=0.900 total time=   0.
6s
[CV 2/5] END .....max_depth=13, max_features=10;, score=0.903 total time=   0.
6s
[CV 3/5] END .....max_depth=13, max_features=10;, score=0.881 total time=   0.
6s
[CV 4/5] END .....max_depth=13, max_features=10;, score=0.875 total time=   0.
6s
[CV 5/5] END .....max_depth=13, max_features=10;, score=0.884 total time=   0.
6s
[CV 1/5] END .....max_depth=13, max_features=11;, score=0.906 total time=   0.
6s
[CV 2/5] END .....max_depth=13, max_features=11;, score=0.897 total time=   0.
7s
[CV 3/5] END .....max_depth=13, max_features=11;, score=0.884 total time=   0.
8s
[CV 4/5] END .....max_depth=13, max_features=11;, score=0.887 total time=   0.
9s
[CV 5/5] END .....max_depth=13, max_features=11;, score=0.887 total time=   0.
9s
[CV 1/5] END .....max_depth=13, max_features=12;, score=0.900 total time=   1.
0s
[CV 2/5] END .....max_depth=13, max_features=12;, score=0.891 total time=   1.
0s
[CV 3/5] END .....max_depth=13, max_features=12;, score=0.903 total time=   0.
8s
[CV 4/5] END .....max_depth=13, max_features=12;, score=0.869 total time=   0.
7s
[CV 5/5] END .....max_depth=13, max_features=12;, score=0.881 total time=   0.
7s
[CV 1/5] END .....max_depth=13, max_features=13;, score=0.900 total time=   0.
7s
[CV 2/5] END .....max_depth=13, max_features=13;, score=0.903 total time=   0.
7s
[CV 3/5] END .....max_depth=13, max_features=13;, score=0.894 total time=   1.
0s
[CV 4/5] END .....max_depth=13, max_features=13;, score=0.897 total time=   0.
8s
[CV 5/5] END .....max_depth=13, max_features=13;, score=0.878 total time=   0.
```

```
7s
[CV 1/5] END ......max_depth=14, max_features=1;, score=0.700 total time=   0.
3s
[CV 2/5] END ......max_depth=14, max_features=1;, score=0.741 total time=   0.
3s
[CV 3/5] END ......max_depth=14, max_features=1;, score=0.706 total time=   0.
3s
[CV 4/5] END ......max_depth=14, max_features=1;, score=0.697 total time=   0.
3s
[CV 5/5] END ......max_depth=14, max_features=1;, score=0.669 total time=   0.
3s
[CV 1/5] END ......max_depth=14, max_features=2;, score=0.828 total time=   0.
3s
[CV 2/5] END ......max_depth=14, max_features=2;, score=0.819 total time=   0.
3s
[CV 3/5] END ......max_depth=14, max_features=2;, score=0.825 total time=   0.
3s
[CV 4/5] END ......max_depth=14, max_features=2;, score=0.769 total time=   0.
4s
[CV 5/5] END ......max_depth=14, max_features=2;, score=0.791 total time=   0.
4s
[CV 1/5] END ......max_depth=14, max_features=3;, score=0.834 total time=   0.
4s
[CV 2/5] END ......max_depth=14, max_features=3;, score=0.872 total time=   0.
4s
[CV 3/5] END ......max_depth=14, max_features=3;, score=0.859 total time=   0.
4s
[CV 4/5] END ......max_depth=14, max_features=3;, score=0.825 total time=   0.
6s
[CV 5/5] END ......max_depth=14, max_features=3;, score=0.838 total time=   0.
6s
[CV 1/5] END ......max_depth=14, max_features=4;, score=0.878 total time=   0.
6s
[CV 2/5] END ......max_depth=14, max_features=4;, score=0.903 total time=   0.
6s
[CV 3/5] END ......max_depth=14, max_features=4;, score=0.856 total time=   0.
6s
[CV 4/5] END ......max_depth=14, max_features=4;, score=0.838 total time=   0.
7s
[CV 5/5] END ......max_depth=14, max_features=4;, score=0.838 total time=   0.
6s
[CV 1/5] END ......max_depth=14, max_features=5;, score=0.894 total time=   0.
6s
[CV 2/5] END ......max_depth=14, max_features=5;, score=0.891 total time=   0.
4s
[CV 3/5] END ......max_depth=14, max_features=5;, score=0.878 total time=   0.
4s
[CV 4/5] END ......max_depth=14, max_features=5;, score=0.863 total time=   0.
4s
[CV 5/5] END ......max_depth=14, max_features=5;, score=0.853 total time=   0.
5s
[CV 1/5] END ......max_depth=14, max_features=6;, score=0.894 total time=   0.
```

```
5s
[CV 2/5] END ......max_depth=14, max_features=6;, score=0.912 total time=   0.
5s
[CV 3/5] END ......max_depth=14, max_features=6;, score=0.894 total time=   0.
5s
[CV 4/5] END ......max_depth=14, max_features=6;, score=0.869 total time=   0.
5s
[CV 5/5] END ......max_depth=14, max_features=6;, score=0.863 total time=   0.
5s
[CV 1/5] END ......max_depth=14, max_features=7;, score=0.900 total time=   0.
5s
[CV 2/5] END ......max_depth=14, max_features=7;, score=0.897 total time=   0.
5s
[CV 3/5] END ......max_depth=14, max_features=7;, score=0.872 total time=   0.
5s
[CV 4/5] END ......max_depth=14, max_features=7;, score=0.869 total time=   0.
5s
[CV 5/5] END ......max_depth=14, max_features=7;, score=0.869 total time=   0.
5s
[CV 1/5] END ......max_depth=14, max_features=8;, score=0.900 total time=   0.
5s
[CV 2/5] END ......max_depth=14, max_features=8;, score=0.916 total time=   0.
5s
[CV 3/5] END ......max_depth=14, max_features=8;, score=0.891 total time=   0.
5s
[CV 4/5] END ......max_depth=14, max_features=8;, score=0.878 total time=   0.
6s
[CV 5/5] END ......max_depth=14, max_features=8;, score=0.875 total time=   0.
6s
[CV 1/5] END ......max_depth=14, max_features=9;, score=0.900 total time=   0.
8s
[CV 2/5] END ......max_depth=14, max_features=9;, score=0.897 total time=   0.
9s
[CV 3/5] END ......max_depth=14, max_features=9;, score=0.891 total time=   0.
8s
[CV 4/5] END ......max_depth=14, max_features=9;, score=0.897 total time=   0.
8s
[CV 5/5] END ......max_depth=14, max_features=9;, score=0.875 total time=   0.
9s
[CV 1/5] END .....max_depth=14, max_features=10;, score=0.903 total time=   0.
9s
[CV 2/5] END .....max_depth=14, max_features=10;, score=0.919 total time=   0.
7s
[CV 3/5] END .....max_depth=14, max_features=10;, score=0.884 total time=   0.
6s
[CV 4/5] END .....max_depth=14, max_features=10;, score=0.872 total time=   0.
7s
[CV 5/5] END .....max_depth=14, max_features=10;, score=0.884 total time=   0.
6s
[CV 1/5] END .....max_depth=14, max_features=11;, score=0.909 total time=   0.
7s
[CV 2/5] END .....max_depth=14, max_features=11;, score=0.906 total time=   0.
```

```
7s
[CV 3/5] END .....max_depth=14, max_features=11;, score=0.897 total time=   0.
6s
[CV 4/5] END .....max_depth=14, max_features=11;, score=0.878 total time=   0.
7s
[CV 5/5] END .....max_depth=14, max_features=11;, score=0.884 total time=   0.
7s
[CV 1/5] END .....max_depth=14, max_features=12;, score=0.897 total time=   0.
7s
[CV 2/5] END .....max_depth=14, max_features=12;, score=0.891 total time=   0.
7s
[CV 3/5] END .....max_depth=14, max_features=12;, score=0.897 total time=   0.
7s
[CV 4/5] END .....max_depth=14, max_features=12;, score=0.878 total time=   0.
7s
[CV 5/5] END .....max_depth=14, max_features=12;, score=0.891 total time=   0.
7s
[CV 1/5] END .....max_depth=14, max_features=13;, score=0.906 total time=   0.
7s
[CV 2/5] END .....max_depth=14, max_features=13;, score=0.903 total time=   0.
9s
[CV 3/5] END .....max_depth=14, max_features=13;, score=0.884 total time=   1.
0s
[CV 4/5] END .....max_depth=14, max_features=13;, score=0.884 total time=   1.
0s
[CV 5/5] END .....max_depth=14, max_features=13;, score=0.881 total time=   1.
1s
```

Out[36]:

```
▸ GridSearchCV
▸ best_estimator_: RandomForestClassifier
    ▸ RandomForestClassifier
```

Grid Searched Model Evaluation

In [37]:

```python
# Retrieve the best parameters and score
print("Best Parameters:", clf.best_params_)
print("Best Cross-Validation Score:", clf.best_score_)
```

```
Best Parameters: {'max_depth': 11, 'max_features': 11}
Best Cross-Validation Score: 0.896875
```

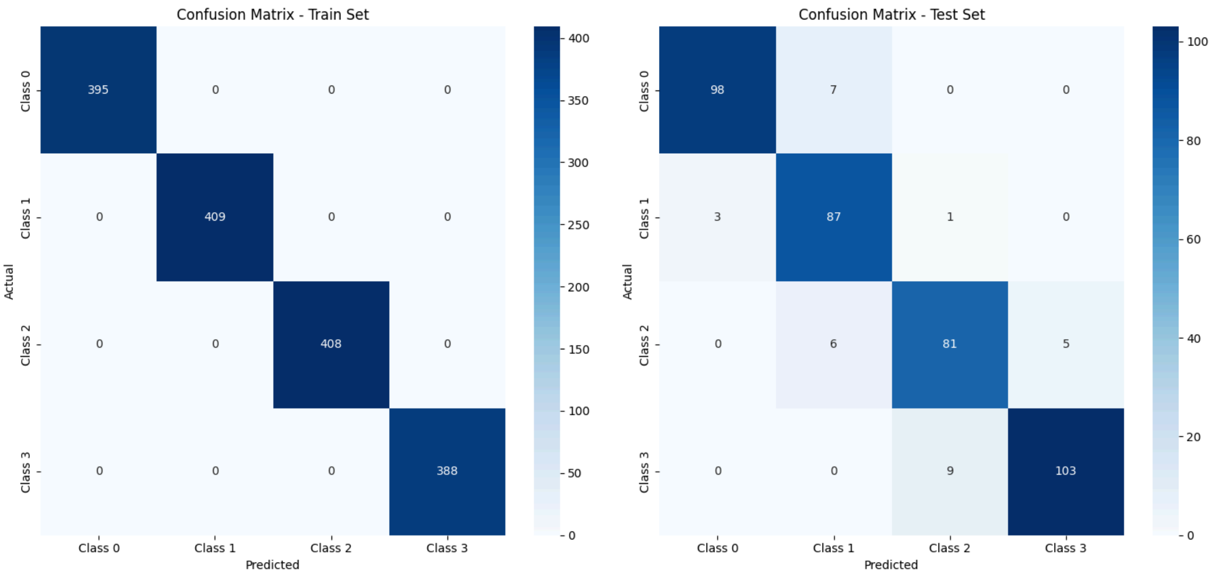
In [38]:

```python
# Evaluate on train and test sets
print("Train Score:", clf.best_estimator_.score(X_train, y_train_1d))
print("Test Score:", clf.best_estimator_.score(X_test, y_test.values.ravel()))
```

```
Train Score: 1.0
Test Score: 0.9225
```

In [39]:

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Best estimator (after GridSearchCV)
best_model = clf.best_estimator_

# Predict on train and test data
y_train_pred = best_model.predict(X_train)
y_test_pred = best_model.predict(X_test)

# Compute confusion matrix for train and test
train_conf_matrix = confusion_matrix(y_train_1d, y_train_pred)
test_conf_matrix = confusion_matrix(y_test.values.ravel(), y_test_pred)

# Create a figure with two subplots (1 row, 2 columns)
fig, axes = plt.subplots(1, 2, figsize=(15, 7))

# Plot confusion matrix for train data
sns.heatmap(train_conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels
axes[0].set_title('Confusion Matrix - Train Set')
axes[0].set_xlabel('Predicted')
axes[0].set_ylabel('Actual')

# Plot confusion matrix for test data
sns.heatmap(test_conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=
axes[1].set_title('Confusion Matrix - Test Set')
axes[1].set_xlabel('Predicted')
axes[1].set_ylabel('Actual')

plt.subplots_adjust(wspace=0.3)

# Display the plots
plt.tight_layout()
plt.show()
```



In [40]:
```
from sklearn.metrics import classification_report

# Get the classification report for train and test predictions
train_class_report = classification_report(y_train_1d, y_train_pred)
test_class_report = classification_report(y_test.values.ravel(), y_test_pred)

# Print the classification reports
print("Classification Report - Train Set:\n", train_class_report)
print("Classification Report - Test Set:\n", test_class_report)
```

```
Classification Report - Train Set:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00       395
           1       1.00      1.00      1.00       409
           2       1.00      1.00      1.00       408
           3       1.00      1.00      1.00       388

    accuracy                           1.00      1600
   macro avg       1.00      1.00      1.00      1600
weighted avg       1.00      1.00      1.00      1600

Classification Report - Test Set:
               precision    recall  f1-score   support

           0       0.97      0.93      0.95       105
           1       0.87      0.96      0.91        91
           2       0.89      0.88      0.89        92
           3       0.95      0.92      0.94       112

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

Implementing the Support Vector Machine (SVM) model

In [41]:
```
from sklearn.svm import SVC  # Importing Support Vector Classifier (SVC)
svm = SVC(C=1, kernel='rbf')  # Instance of SVM with default parameters
svm.fit(X_train, y_train.values.ravel())  # Fit the SVM model to the training
```

Out[41]:
```
▾ SVC
SVC(C=1)
```

Model Evaluation for SVM

In [42]:
```
accuracy = svm.score(X_train, y_train)  # Calculate accuracy on the training
print("Training Accuracy is", accuracy)  # Print the training accuracy
```

```
Training Accuracy is 0.851875
```

In [43]:
```
accuracy = svm.score(X_test, y_test)  # Calculate accuracy on the testing dat
print("Testing Accuracy is", accuracy)  # Print the testing accuracy
```

```
Testing Accuracy is 0.81
```

SVM Cross-Validation

In [44]:
```
from sklearn.model_selection import cross_validate
from sklearn.svm import SVC

# Ensure y_train is a 1D array by converting it to a NumPy array and using ra
svm_cv = SVC(C=1, kernel='rbf')  # Instance of SVM (customizable parameters l

# Convert y_train to a NumPy array and flatten it with ravel()
cv_results_svm = cross_validate(svm_cv, X_train, y_train.values.ravel(), scor

# Print the accuracy scores for each fold
print(cv_results_svm['test_score'])  # Print the accuracy scores for each fol
```

```
[0.79375  0.765625 0.746875 0.721875 0.746875]
```

Cross Validated Model Evaluation

Fit the Best Estimator to the Whole Training Dataset (after cross-validation)

In [45]:
```
# After cross-validation, we fit the model on the whole training dataset
svm.fit(X_train, y_train.values.ravel())  # Ensure y_train is a 1D array usin
```

Out[45]:
```
▾ SVC
SVC(C=1)
```

Calculate the Accuracy for the Complete Training and Test Sets (after cross-validation)

In [46]:
```
# Accuracy on the training data (after cross-validation)
train_accuracy = svm.score(X_train, y_train.values.ravel())
print("Training Accuracy:", train_accuracy)

# Accuracy on the test data (after cross-validation)
test_accuracy = svm.score(X_test, y_test)
print("Testing Accuracy:", test_accuracy)
```

```
Training Accuracy: 0.851875
Testing Accuracy: 0.81
```

Grid Search CV for SVM

In [47]:
```
from sklearn.svm import SVC  # Importing Support Vector Classifier (SVC)
from sklearn.model_selection import GridSearchCV

#Re-define the initial SVM model
svm = SVC()  # You can customize the default parameters here if needed

# Define the parameter grid for GridSearchCV
param_grid = {
    'C': [0.1, 1, 10],  # Regularization parameter
    'kernel': ['linear', 'rbf'],  # Kernel types
    'gamma': ['scale', 'auto']  # Gamma values for RBF kernel
}

# Perform Grid Search with Cross-Validation
grid_search = GridSearchCV(estimator=svm, param_grid=param_grid, cv=5, scorin
grid_search.fit(X_train, y_train.values.ravel())  # Fit GridSearchCV on the t

# Get the best hyperparameters from GridSearchCV
best_params = grid_search.best_params_
print("Best Parameters from Grid Search:", best_params)

# Use the best estimator found by GridSearchCV
best_svm = grid_search.best_estimator_

# Fit the best estimator on the whole training dataset
```

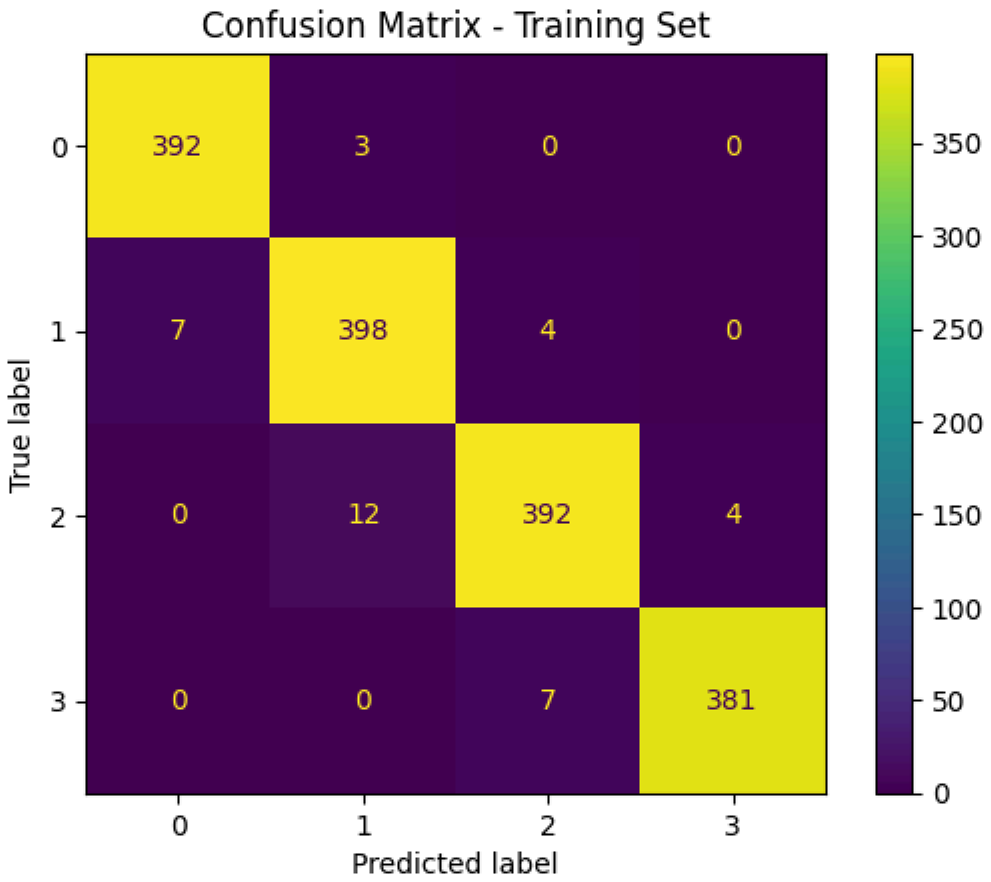
```
best_svm.fit(X_train, y_train.values.ravel())

# Calculate the Accuracy for the Complete Training and Test Sets (after Grid
train_accuracy = best_svm.score(X_train, y_train.values.ravel())  # Training
print("Training Accuracy (after Grid Search):", train_accuracy)

test_accuracy = best_svm.score(X_test, y_test)  # Testing accuracy
print("Testing Accuracy (after Grid Search):", test_accuracy)
```

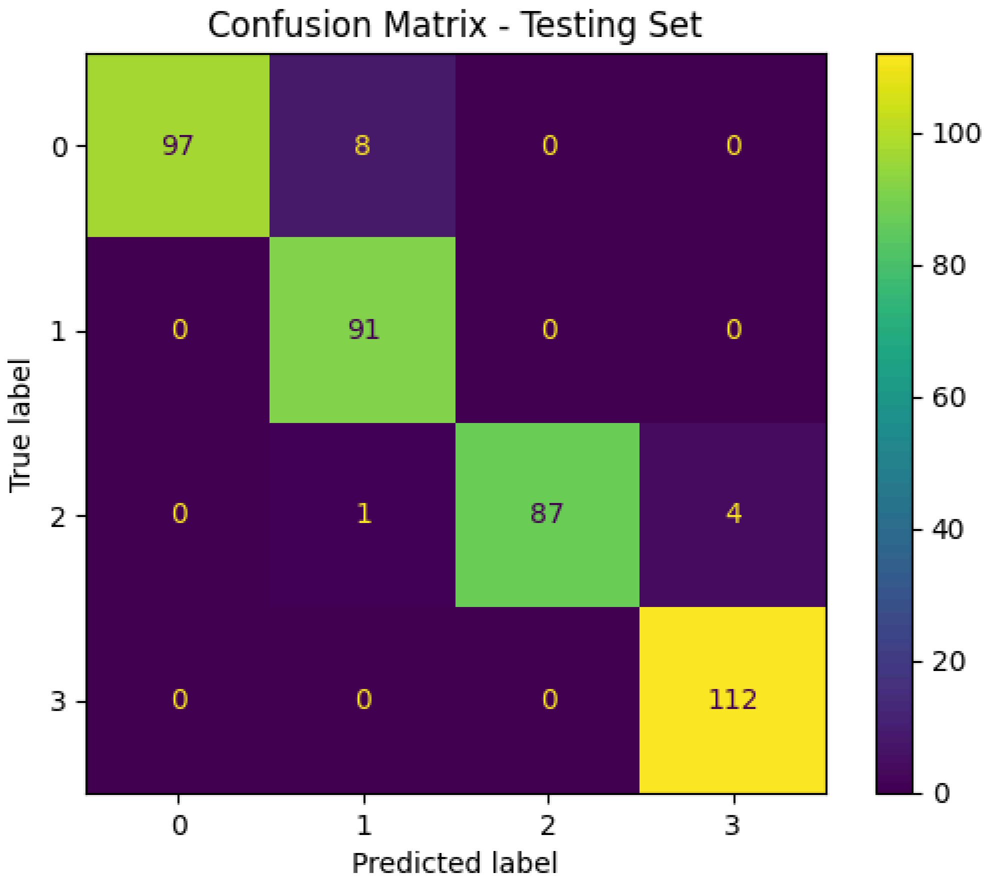
```
Best Parameters from Grid Search: {'C': 10, 'gamma': 'scale', 'kernel': 'linea
r'}
Training Accuracy (after Grid Search): 0.976875
Testing Accuracy (after Grid Search): 0.9675
```

Confusion Matrix

In [48]:

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Confusion matrix for the training set
train_predictions = best_svm.predict(X_train)  # Predict on the training set
train_cm = confusion_matrix(y_train, train_predictions)  # Calculate confusio


# Visualize the confusion matrix for the training set
ConfusionMatrixDisplay(confusion_matrix=train_cm).plot()
plt.title("Confusion Matrix - Training Set")
plt.show()

# Confusion matrix for the test set
test_predictions = best_svm.predict(X_test)  # Predict on the testing set
test_cm = confusion_matrix(y_test, test_predictions)  # Calculate confusion m


# Visualize the confusion matrix for the test set
ConfusionMatrixDisplay(confusion_matrix=test_cm).plot()
plt.title("Confusion Matrix - Testing Set")
plt.show()
```


Confusion Matrix - Training Set
True label
Predicted label
0
1
2
3
392 3 0 0
7 398 4 0
0 12 392 4
0 0 7 381
350
300
250
200
150
100
50
0


Confusion Matrix - Testing Set
True label
Predicted label
0
1
2
3
97 8 0 0
0 91 0 0
0 1 87 4
0 0 0 112
100
80
60
40
20
0

Classification Report

In [49]:

```python
from sklearn.metrics import classification_report

# Generate predictions for the training and testing datasets
train_predictions = best_svm.predict(X_train)
test_predictions = best_svm.predict(X_test)

# Training set classification report
print("Classification Report for Training Set:")
print(classification_report(y_train, train_predictions))

# Testing set classification report
print("\nClassification Report for Testing Set:")
print(classification_report(y_test, test_predictions))
```

```
Classification Report for Training Set:
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       395
           1       0.96      0.97      0.97       409
           2       0.97      0.96      0.97       408
           3       0.99      0.98      0.99       388

    accuracy                           0.98      1600
   macro avg       0.98      0.98      0.98      1600
weighted avg       0.98      0.98      0.98      1600


Classification Report for Testing Set:
              precision    recall  f1-score   support

           0       1.00      0.92      0.96       105
           1       0.91      1.00      0.95        91
           2       1.00      0.95      0.97        92
           3       0.97      1.00      0.98       112

    accuracy                           0.97       400
   macro avg       0.97      0.97      0.97       400
weighted avg       0.97      0.97      0.97       400
```

Conclusion

The Random Forest model seems to generalize well from the training data 100% and for test data 92.25%, with high performance across both. The slight decrease in test performance might indicate some overfitting to the training set because the difference between training accuracy and testing accuracy is grater than 5%

SVM model seems to generalize well from the training data to the test data. The model achieved a training accuracy of 97.69% and a testing accuracy of 96.75%. This suggests the model is well-trained and generalizes well to unseen data.

Both models perform well, with the Random Forest achieving 92.25% accuracy on the test set and the SVM achieving 96.75%. For training data Random Forest achieving 100% accuracy and SVM achieving 97.69%.The Random Forest has perfect training accuracy but struggles with Class 2, while the SVM shows balanced performance across all classes. Both models generalize well, but the Random Forest may need minor adjustments for Class 2, whereas the SVM is ready for deployment.

References

The dataset used for this analysis, titled "Mobile Price Classification", is publicly available on Kaggle and was created by Abhishek Sharma. It contains information on various features of mobile phones, such as battery power, RAM, and screen dimensions, along with the corresponding price range classification. This dataset is frequently used for machine learning projects, including classification tasks, due to its balanced and well-structured nature.
Dataset Link : https://www.kaggle.com/datasets/iabhishekofficial/mobile-price-classification/discussion?sort=hotness