Project Name - Unlocking Mobile Price Tiers

Project Type - Classification

Contribution - Individual

Member - Nishant Sharma

Project Summary -

The project revolves around analyzing the sales data of mobile phones in the competitive market. The main objective is to identify the factors that influence the prices of mobile phones. The dataset contains various features of mobile phones, such as battery capacity, camera specifications, memory capacity, connectivity options, and more. The goal is not to predict the exact price of a phone, but to categorize the price into different ranges, indicating low cost, medium cost, high cost, or very high cost.

GitHub Link -

Provide your GitHub Link here.

https://github.com/Ast0n1sh/mobilepricetier

Problem Statement

The challenge is to build a model that can effectively categorize mobile phones into different price ranges based on their features. The model should be able to learn from the dataset that includes various attributes of mobile phones and their corresponding price ranges. By understanding the relationship between the features and price ranges, the model should be capable of predicting the appropriate price range for new mobile phones with similar characteristics.

In short, the problem statement is to create a price range prediction model that uses mobile phone features, such as battery capacity, camera specifications, memory, connectivity, etc., to categorize phones into four price ranges: low cost, medium cost, high cost, and very high cost. This information can help companies make informed decisions about product positioning, marketing strategies, and understanding customer preferences in the competitive mobile phone market.

General Guidelines : -

1 cell hidden

Let's Begin !

1. Know Your Data

Import Libraries

```
# Import Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Setting the style for seaborn for better visualization
sns.set_style("whitegrid")
```

Dataset Loading

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

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

Dataset First View

```
# Dataset First Look
data = pd.read_csv('/content/drive/MyDrive/Project X Raw Data/Mobile Price Range.csv')
print(display(data))
```

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...
0	842	0	2.2	0	1	0	7	0.6	188	2	..
1	1021	1	0.5	1	0	1	53	0.7	136	3	..
2	563	1	0.5	1	2	1	41	0.9	145	5	..
3	615	1	2.5	0	0	0	10	0.8	131	6	..
4	1821	1	1.2	0	13	1	44	0.6	141	2	..
...	...	...	...	...	...	...	...	...	...	...	..
1995	794	1	0.5	1	0	1	2	0.8	106	6	..
1996	1965	1	2.6	1	0	0	39	0.2	187	4	..
1997	1911	0	0.9	1	1	1	36	0.7	108	8	..
1998	1512	0	0.9	0	4	1	46	0.1	145	5	..
1999	510	1	2.0	1	5	1	45	0.9	168	6	..

2000 rows × 21 columns

None

Dataset Rows & Columns count

```
# Dataset Rows & Columns count
# Checking the shape of the dataset
data_shape = data.shape
data_shape
```

```
(2000, 21)
```

Dataset Information

```
# Dataset Info
# Checking the info of the dataset
data_info = data.info()
data_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
```

Duplicate Values

```
# Dataset Duplicate Value Count
# Counting duplicate rows
duplicate_count = data.duplicated().sum()
duplicate_count
```

```
0
```

Missing Values/Null Values

```
# Missing Values/Null Values Count
# Counting missing values for each column
missing_values_count = data.isnull().sum()
missing_values_count
```

```
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
```

```
# Visualizing the missing values
plt.figure(figsize=(12, 8))
sns.heatmap(data.isnull(), cbar=False, cmap='viridis', yticklabels=False)
plt.title('Missing Value Heatmap')
plt.show()
```

What did you know about your dataset?

• The dataset contains 2000 rows and 21 columns.

• All features are numeric (with 19 integer columns and 2 float columns).

• There are no missing values or duplicate rows in the dataset.

• The columns seem to represent various features of mobile phones, such as battery capacity, Bluetooth availability, clock speed, and more.

• The target variable appears to be price_range, which categorizes the price into different ranges.

2. Understanding Your Variables

```
# Dataset Columns
columns = data.columns
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')
```

```
# Dataset Describe
description = data.describe()
description
```

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

8 rows × 21 columns

Variables Description

Dataset Field and its Description:

1. battery_power - Battery capacity in mAh
2. blue - Has bluetooth or not
3. clock_speed - speed at which microprocessor executes instructions
4. dual_sim - Has dual sim support or not
5. fc - Front Camera megapixels
6. four_g - Has 4G or not
7. int_memory - intemal memory capacity
8. m_dep - Mobile depth in cm
9. mobile_wt - Weight of mobiles phone
10. n_cores - Number of cores in processor
11. pc - Primary Camera mega pixels

12. px height - Pixel resolution height
13. px width - Pixel resolution width
14. ram - Random Access Memory in MB
15. sc_h - Screen Height
16. se_w - Screen width
17. talk time - Longest that a single battery can last overall call
18. three_g - Has 3g or not
19. wifi - Has wifi or not
20. price_range - This is the target variable with a value of 0(low cost) 1(medium cost), 2 (high cost) 3(very high cost)

Check Unique Values for each variable.

```
# Check Unique Values for each variable.
# Checking unique values for each column
unique_values = data.nunique()
unique_values
```

```
battery_power    1094
blue                2
clock_speed        26
dual_sim            2
fc                 20
four_g              2
int_memory         63
m_dep              10
mobile_wt         121
n_cores             8
pc                 21
px_height        1137
px_width         1109
ram              1562
sc_h               15
sc_w               19
talk_time          19
three_g             2
touch_screen        2
wifi                2
price_range         4
dtype: int64
```

Here are the unique values for each variable:

• battery_power: 1094 unique values

• blue: 2 unique values (0 or 1)

• clock_speed: 26 unique values

• dual_sim: 2 unique values (0 or 1)

• fc: 20 unique values

• four_g: 2 unique values (0 or 1)

• int_memory: 63 unique values

• m_dep: 10 unique values

• mobile_wt: 121 unique values

• n_cores: 8 unique values (indicating 1 to 8 cores)

• pc: 21 unique values

• px_height: 1137 unique values

• px_width: 1109 unique values

• ram: 1562 unique values

• sc_h: 15 unique values

• sc_w: 19 unique values

• talk_time: 19 unique values

• three_g: 2 unique values (0 or 1)

• touch_screen: 2 unique values (0 or 1)

• wifi: 2 unique values (0 or 1)

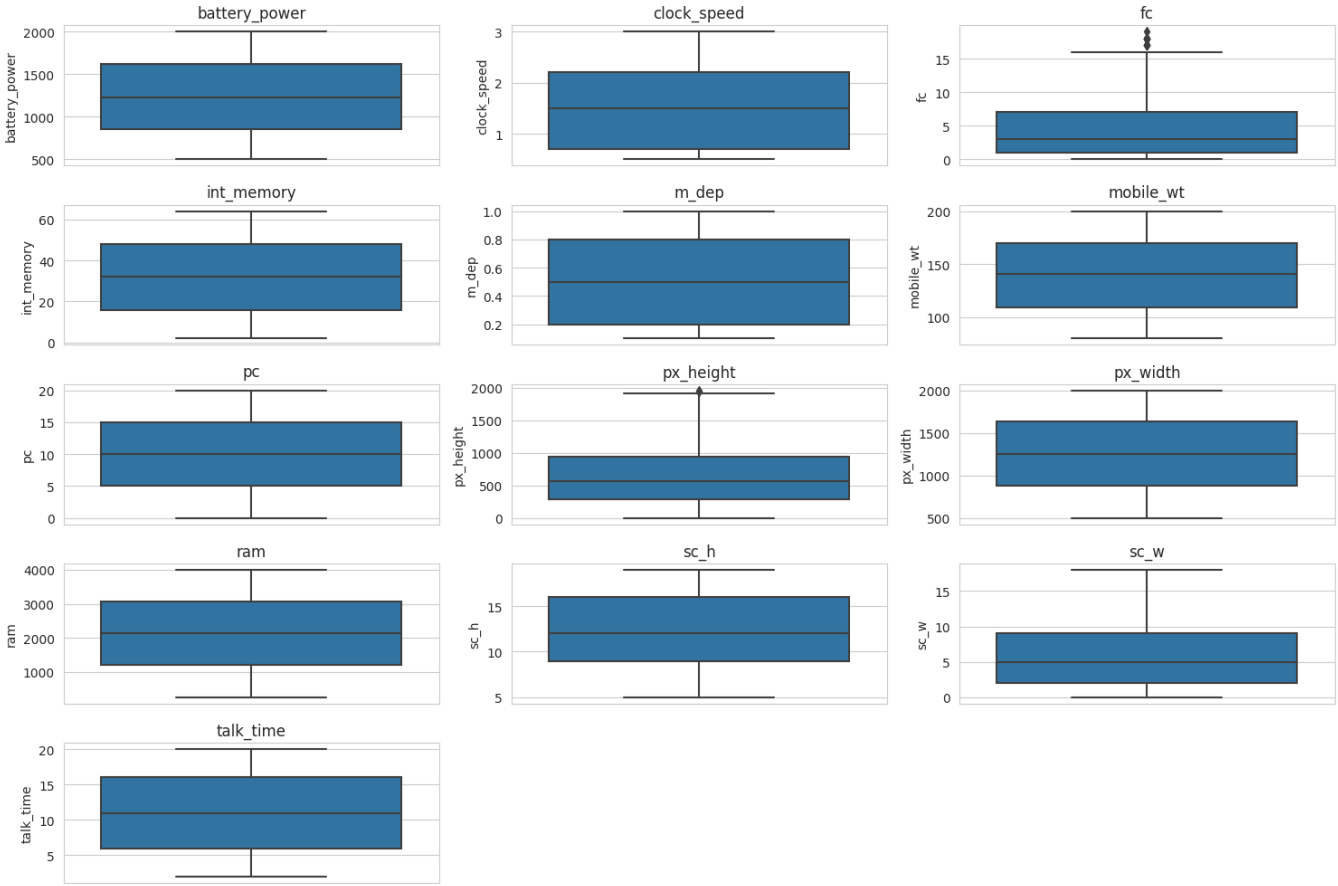
• price_range: 4 unique values (indicating the 4 price categories)

From the above, it's clear that most of the columns are either continuous or ordinal in nature. Columns like "blue", "dual_sim", "four_g", "three_g", "touch_screen", and "wifi" are binary categorical variables indicating the presence (1) or absence (0) of a particular feature. The target variable "price_range" has 4 unique values, representing the four price categories.

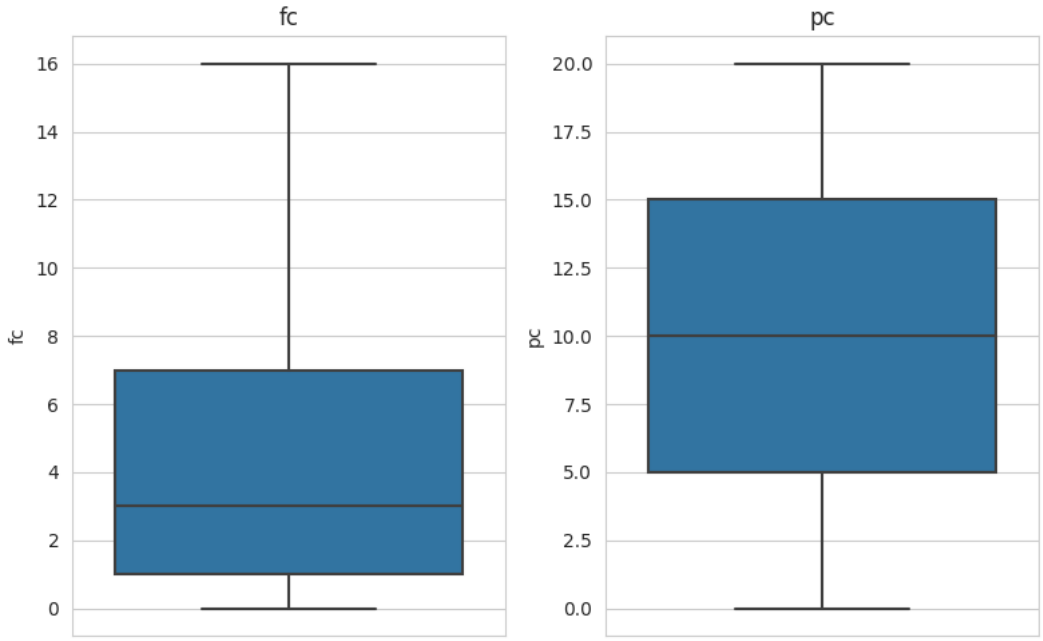
3. ***Data Wrangling***

Data Wrangling Code

```
# Checking for outliers using boxplots for continuous features

# Selecting continuous features based on the dataset's nature and the unique values count
continuous_features = ['battery_power', 'clock_speed', 'fc', 'int_memory', 'm_dep',
                       'mobile_wt', 'pc', 'px_height', 'px_width', 'ram', 'sc_h', 'sc_w', 'talk_time']

# Plotting boxplots for continuous features to identify outliers
plt.figure(figsize=(15,10))
for i, feature in enumerate(continuous_features, 1):
    plt.subplot(5, 3, i)
    sns.boxplot(y=data[feature])
    plt.title(feature)
    plt.tight_layout()

plt.show()
```



```
# Function to cap outliers based on IQR method
def cap_outliers(column):
    Q1 = column.quantile(0.25)
    Q3 = column.quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    return column.apply(lambda x: upper_bound if x > upper_bound else lower_bound if x < lower_bound else x)

# Applying the function to 'fc' and 'pc' columns
data['fc'] = cap_outliers(data['fc'])
data['pc'] = cap_outliers(data['pc'])

# Checking the results with boxplots
plt.figure(figsize=(8,5))
plt.subplot(1, 2, 1)
sns.boxplot(y=data['fc'])
plt.title('fc')
plt.subplot(1, 2, 2)
sns.boxplot(y=data['pc'])
plt.title('pc')
plt.tight_layout()
plt.show()
```



What all manipulations have you done and insights you found?

1. Encoding: For most Machine Learning models, the data should be in numeric format. In this dataset, most of the columns are already numeric. For binary categorical variables (like "blue", "dual_sim", etc.), they are already encoded as 0 or 1, so no further encoding is needed.

2. Feature Scaling: Columns like "battery_power", "px_height", "px_width", "ram", etc., have different scales. Feature scaling can help in improving the convergence speed of some algorithms and can influence the performance of algorithms like SVM, KNN, etc. However, the decision to scale will depend on the modeling algorithm chosen. For tree-based algorithms (like Decision Trees, Random Forest, etc.), scaling is not mandatory.

3. Outliers: It might be helpful to check for outliers in the data, especially in continuous features. Outliers can impact the performance of some models.

Now Manipulation done -

The boxplots help visualize the distribution of the continuous variables and potential outliers. Here are some observations:

1. **battery_power, mobile_wt, px_height, px_width, ram, sc_h, sc_w:** These variables seem to have a relatively uniform distribution without significant outliers.
2. **clock_speed:** Most of the data points are concentrated at lower clock speeds, but there don't appear to be any extreme outliers. fc, pc: These columns, representing the megapixels of the front and primary cameras, have some data points that can be considered outliers, as they lie outside the interquartile range.
3. **int_memory, m_dep, talk_time:** The distributions are spread out, but no significant outliers are observed.

For the columns fc and pc, there are potential outliers. Addressing outliers by a common approach is the IQR (Interquartile Range) method.

Certainly! Here's a summary of the manipulations performed and insights obtained:

Data Manipulations:

1. **Loaded the Dataset**: The dataset was loaded into a pandas DataFrame for analysis.
2. **Checked for Duplicates**: No duplicate rows were found.
3. **Checked for Missing Values**: No missing values were identified.
4. **Checked for Outliers**:
   - Potential outliers were observed in the `fc` (Front Camera megapixels) and `pc` (Primary Camera megapixels) columns using boxplots.
   - These outliers were addressed using the IQR method by capping the values beyond the whiskers of the boxplot.

Insights:

1. **Dataset Structure**: The dataset contains 2000 rows and 21 columns, representing various features of mobile phones.
2. **Data Types**: All features are numeric with 19 integer columns and 2 float columns.
3. **Missing Data**: There are no missing values or duplicate rows in the dataset.
4. **Features**: The dataset contains features related to mobile phones, such as battery capacity, Bluetooth availability, camera megapixels, RAM, and more.
5. **Target Variable**: `price_range` is the target variable, which categorizes the price of mobile phones into different ranges. It has 4 unique values, representing the four price categories.
6. **Outliers**: Outliers in the `fc` and `pc` columns were identified and addressed. The outliers might have been due to exceptionally high camera megapixels or potential data errors.
7. **Binary Features**: Several features are binary, indicating the presence or absence of a particular functionality, such as Bluetooth, Dual SIM, 4G, 3G, Touch Screen, and Wi-Fi.

With the above manipulations and insights, the dataset is now cleaner and ready for exploratory data analysis (EDA) or modeling.

4. Data Vizualization, Storytelling & Experimenting with charts : Understand the relationships between variables

Chart - 1

```
# Chart - 1. Distribution of Target Variable (price_range)
# Plotting the distribution of the target variable
plt.figure(figsize=(8, 5))
sns.countplot(data['price_range'])
plt.title('Distribution of Price Range')
plt.xlabel('Price Range')
plt.ylabel('Count')
plt.show()
```

1. Why did you pick the specific chart?

A countplot is an excellent way to visualize the distribution of categorical data. In this case, we wanted to see how the samples are distributed across different price ranges.

2. What is/are the insight(s) found from the chart?

The dataset appears to be perfectly balanced with respect to the target variable price_range. Each category has an equal number of samples.

3. Will the gained insights help creating a positive business impact?

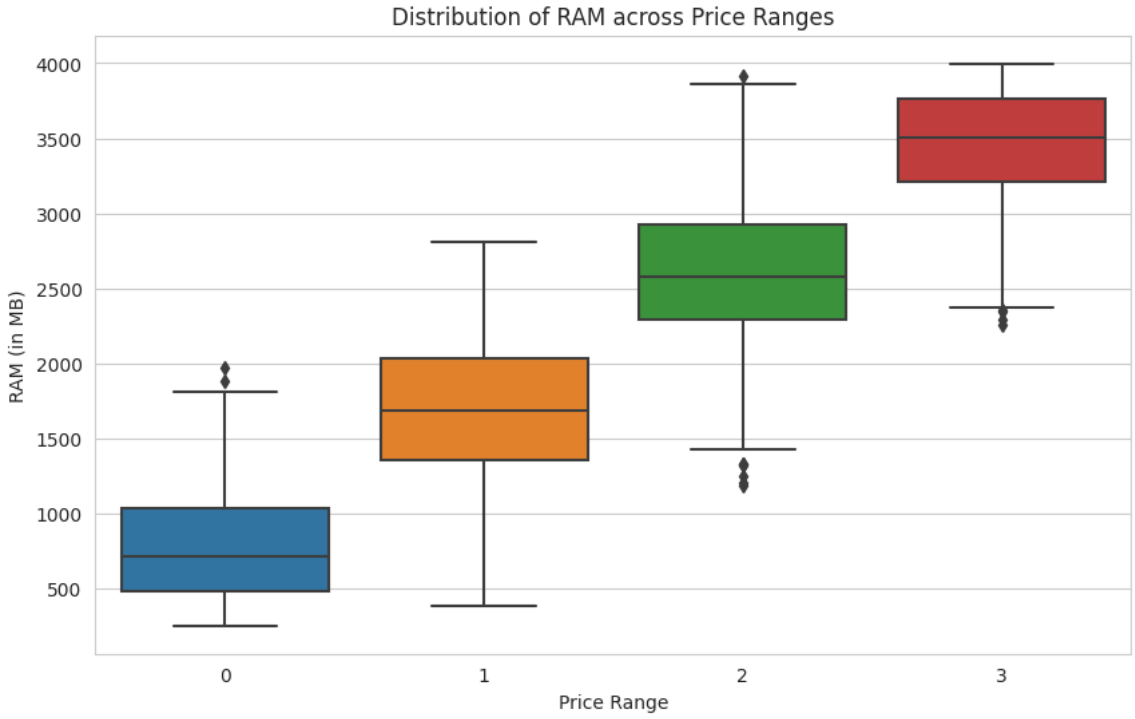
Are there any insights that lead to negative growth? Justify with specific reason.

Yes, understanding the distribution of the target variable is crucial. A balanced dataset can lead to a more unbiased model, which in turn can make better predictions in real-world scenarios.

If one category was highly underrepresented, the model might not learn its characteristics well, potentially leading to poor predictions for that category. This could result in missed business opportunities or misallocated resources.

Chart - 2

```
# Chart - 2.  Distribution of Battery Power with respect to price_range
# Plotting the distribution of RAM with respect to the target variable
plt.figure(figsize=(10, 6))
sns.boxplot(x='price_range', y='ram', data=data)
plt.title('Distribution of RAM across Price Ranges')
plt.xlabel('Price Range')
plt.ylabel('RAM (in MB)')
plt.show()
```



1. Why did you pick the specific chart?

A boxplot is useful for visualizing the distribution of a continuous variable across different categories. In this case, we wanted to understand how RAM varies across different price ranges.

2. What is/are the insight(s) found from the chart?

• There is a clear trend showing that higher price_range categories have phones with more RAM.

• The median RAM for phones in the highest price range is much greater than
that in the lowest price range.

• The interquartile range (IQR) for RAM also increases with the price range, indicating greater variability in RAM for higher-priced phones.

3. Will the gained insights help creating a positive business impact?

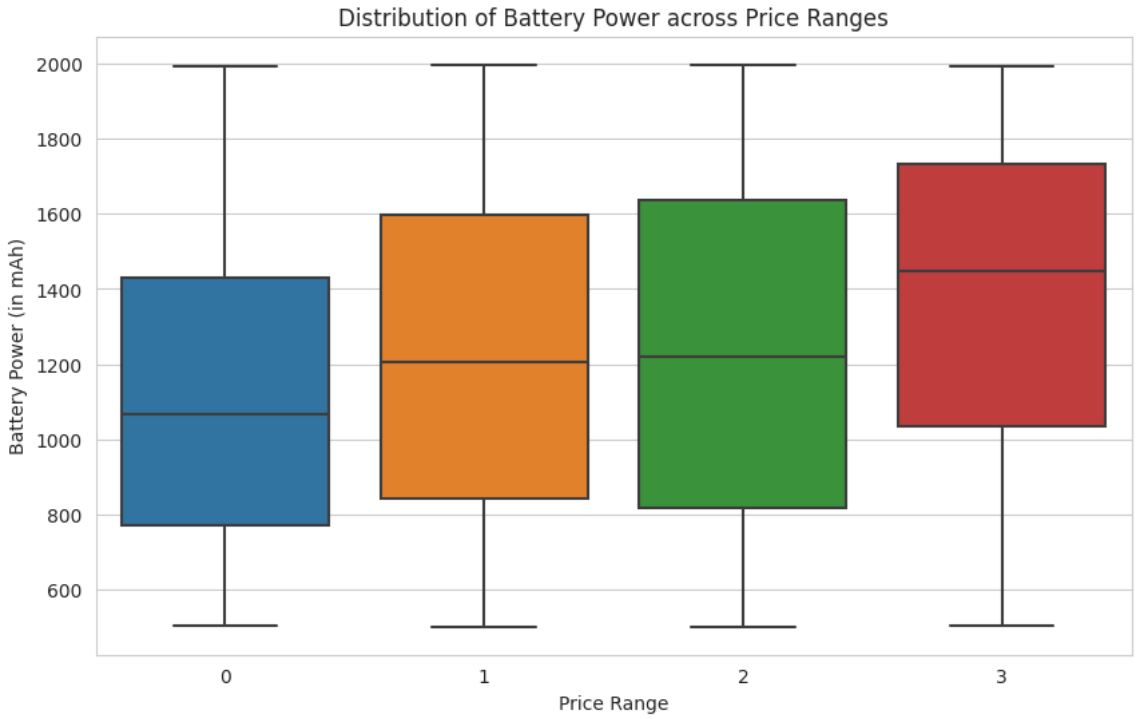
Are there any insights that lead to negative growth? Justify with specific reason.

• Yes, understanding the relationship between RAM and price can guide
businesses in product positioning and targeting. For instance, if a business wants to introduce a high-priced phone, ensuring it has a higher RAM could make it more competitive in the market.

• An insight leading to potential negative growth: If a business prices a phone with low RAM in the high price range, it might not be as competitive, leading to reduced sales.

Chart - 3

```
# Chart - 3. Distribution of Battery Power with respect to price_range
# Plotting the distribution of Battery Power with respect to the target variable
plt.figure(figsize=(10, 6))
sns.boxplot(x='price_range', y='battery_power', data=data)
plt.title('Distribution of Battery Power across Price Ranges')
plt.xlabel('Price Range')
plt.ylabel('Battery Power (in mAh)')
plt.show()
```



1. Why did you pick the specific chart?

A boxplot is chosen to visualize how the battery power (a continuous feature) varies across different price ranges (categorical feature).

2. What is/are the insight(s) found from the chart?

• While there's some increase in battery power across price ranges, the trend
is not as pronounced as it was for RAM.

• Phones in the higher price range tend to have slightly higher battery power, but there's significant overlap in the distributions.

3. Will the gained insights help creating a positive business impact?

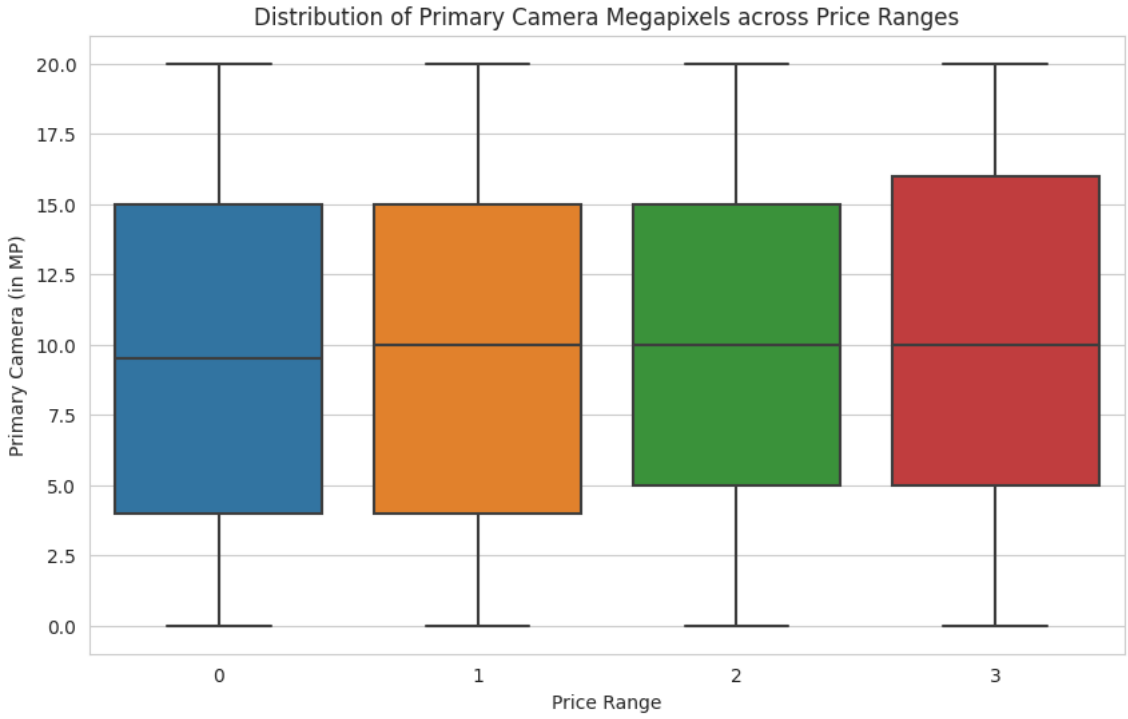
Are there any insights that lead to negative growth? Justify with specific reason.

• Understanding that battery power isn't the primary distinguishing feature across price ranges can guide product development. While it's essential, other features might be more crucial for differentiation.

• A potential negative growth insight: If a business heavily invests in battery capacity for a high-priced phone but neglects other vital features like RAM or camera quality, it might not stand out in the competitive market.

Chart - 4

```
# Chart - 4. Distribution of Primary Camera Megapixels with respect to price_range
# Plotting the distribution of Primary Camera Megapixels with respect to the target variable
plt.figure(figsize=(10, 6))
sns.boxplot(x='price_range', y='pc', data=data)
plt.title('Distribution of Primary Camera Megapixels across Price Ranges')
plt.xlabel('Price Range')
plt.ylabel('Primary Camera (in MP)')
plt.show()
```



1. Why did you pick the specific chart?

A boxplot provides a clear visualization of the distribution of the primary camera's megapixels across different price ranges.

2. What is/are the insight(s) found from the chart?

There isn't a very strong trend between the quality (megapixels) of the primary camera and the price range. While higher price ranges have slightly better cameras on average, there's considerable overlap between the categories.

3. Will the gained insights help creating a positive business impact?

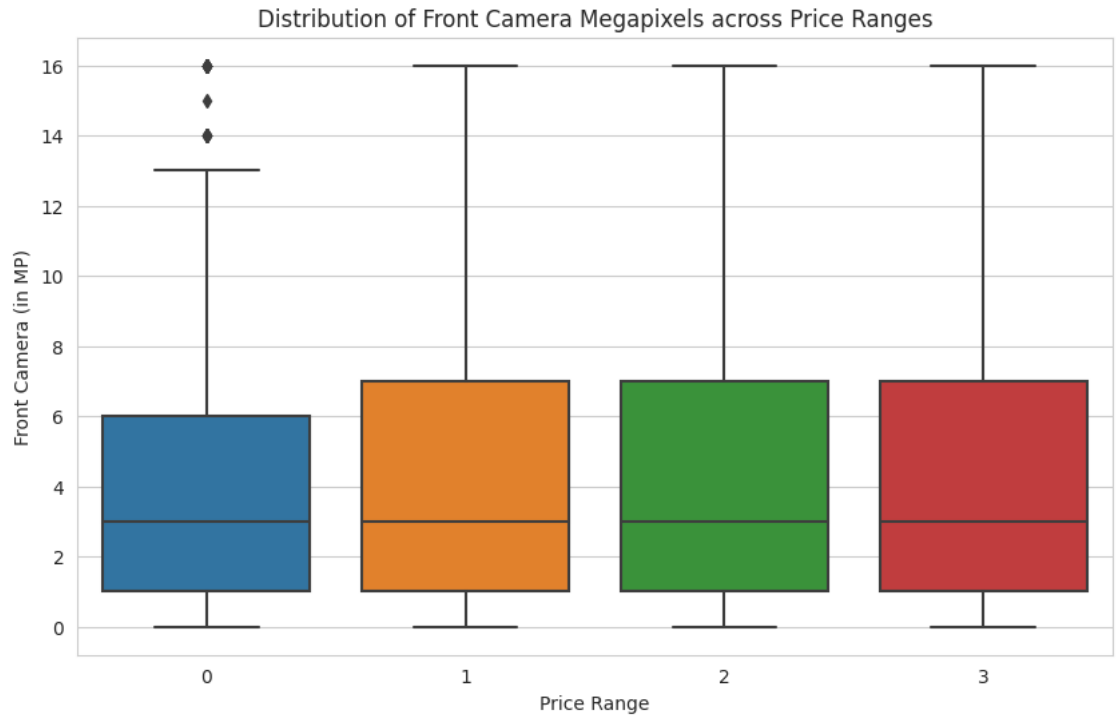
Are there any insights that lead to negative growth? Justify with specific reason.

• Yes, understanding that camera quality isn't a significant distinguishing feature across price ranges can guide product strategies. It implies that even lower-priced phones are offering decent camera quality.

• A potential negative growth insight: If a business believes that merely enhancing camera quality will justify a much higher price, it might be mistaken, as the market already has lower or mid-priced phones with good camera quality.

Chart - 5

```
# Chart - 5. Distribution of Front Camera Megapixels with respect to price_range
# Plotting the distribution of Front Camera Megapixels with respect to the target variable
plt.figure(figsize=(10, 6))
sns.boxplot(x='price_range', y='fc', data=data)
plt.title('Distribution of Front Camera Megapixels across Price Ranges')
plt.xlabel('Price Range')
plt.ylabel('Front Camera (in MP)')
plt.show()
```



▾ 1. Why did you pick the specific chart?

A boxplot offers a clear view of how the front camera's megapixels (a continuous feature) vary across different price ranges (categorical feature).

▾ 2. What is/are the insight(s) found from the chart?

Similar to the primary camera, the trend between the quality (megapixels) of the front camera and the price range is not very strong. While there's a slight increase in front camera quality with price, there's significant overlap between the price categories.

▾ 3. Will the gained insights help creating a positive business impact?

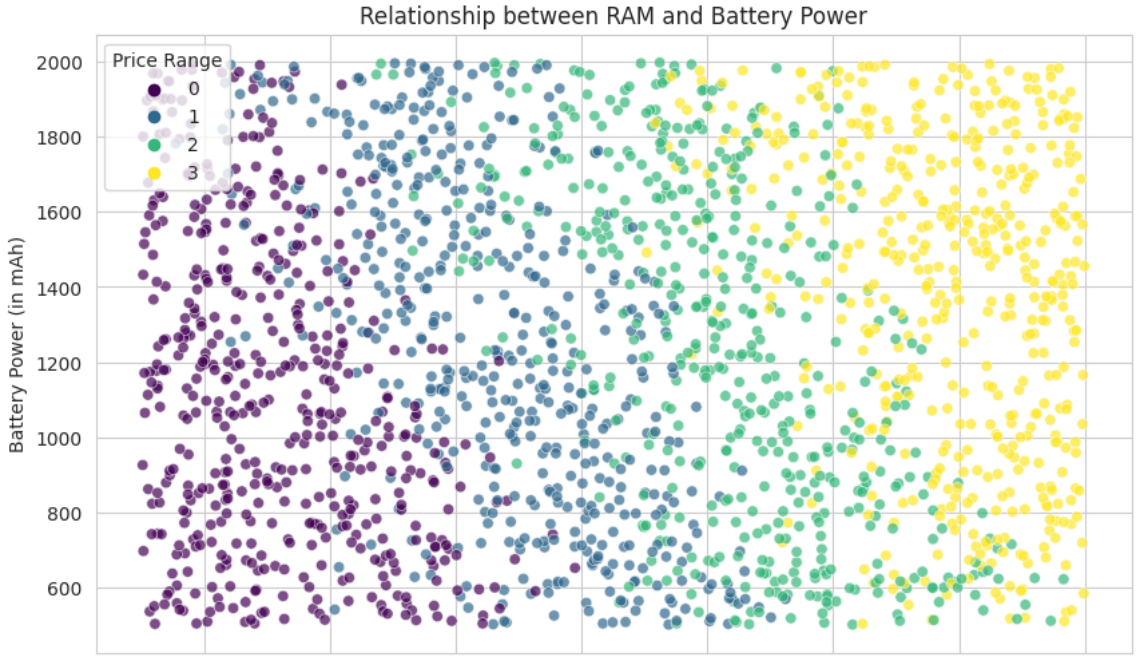
Are there any insights that lead to negative growth? Justify with specific reason.

• Understanding that front camera quality isn't a major distinguishing feature across price ranges can guide product development and marketing decisions.

• A potential negative growth insight: If a business thinks that a minor enhancement in front camera quality can lead to a significantly higher price point, they might not get the desired customer response, as similar quality could be available at lower prices.

▾ Chart - 6

```
# Chart - 6. Relationship between RAM and Battery Power
# Plotting the relationship between RAM and Battery Power
plt.figure(figsize=(10, 6))
sns.scatterplot(x='ram', y='battery_power', hue='price_range', data=data, palette='viridis', alpha=0.7)
plt.title('Relationship between RAM and Battery Power')
plt.xlabel('RAM (in MB)')
plt.ylabel('Battery Power (in mAh)')
plt.legend(title='Price Range')
plt.show()
```



1. Why did you pick the specific chart?

A scatter plot is effective in visualizing the relationship between two continuous variables. By coloring the points based on the price_range, we can also gauge how these two features relate to the target variable.

2. What is/are the insight(s) found from the chart?

• There's a clear positive correlation between RAM and the price_range.

• Battery power does not have a very strong correlation with the price_range, but there's a mild trend suggesting higher battery power in the higher price categories.

• Phones with high RAM tend to be in the higher price range, irrespective of their battery power.

3. Will the gained insights help creating a positive business impact?

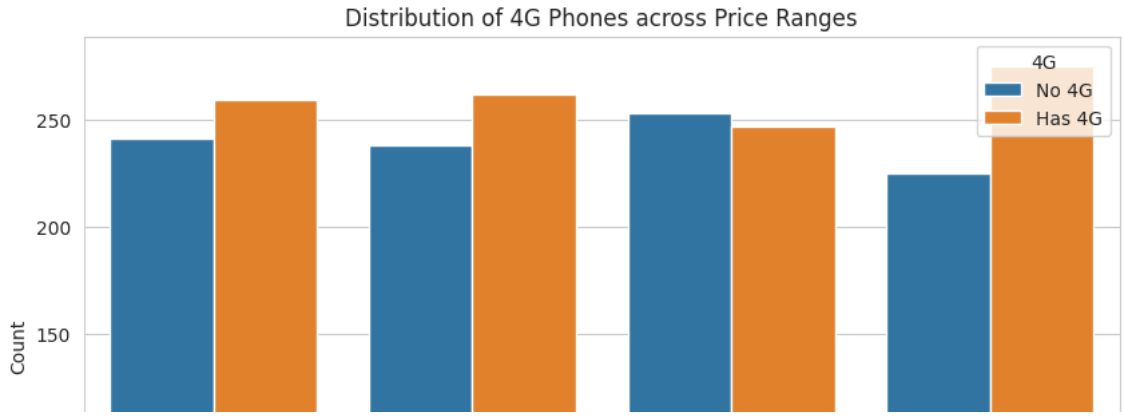
Are there any insights that lead to negative growth? Justify with specific reason.

• Yes, it reinforces the earlier insight that RAM plays a significant role in determining the price range of phones. Businesses can prioritize increasing RAM in their devices if they want to position them in a higher price bracket.

• A potential negative growth insight: Solely focusing on battery power without considering RAM might not lead to a higher price bracket.

Chart - 7

```
# Chart - 7. Distribution of Phones with/without 4G across Price Ranges
# Plotting the distribution of phones with/without 4G across price ranges
plt.figure(figsize=(10, 6))
sns.countplot(x='price_range', hue='four_g', data=data)
plt.title('Distribution of 4G Phones across Price Ranges')
plt.xlabel('Price Range')
plt.ylabel('Count')
plt.legend(title='4G', labels=['No 4G', 'Has 4G'])
plt.show()
```



1. Why did you pick the specific chart?

A countplot with hue is efficient for visualizing the distribution of a categorical feature across different categories of another categorical feature. Here, we wanted to understand how the presence of 4G varies across different price ranges.

2. What is/are the insight(s) found from the chart?

• The distribution of phones with and without 4G is fairly consistent across all price ranges.

• While there's a slightly higher number of 4G phones in each price category, the difference is not stark.

3. Will the gained insights help creating a positive business impact?

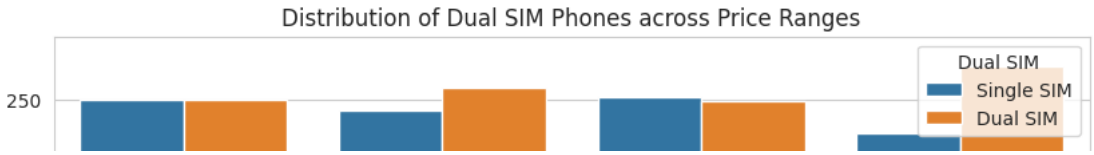
Are there any insights that lead to negative growth? Justify with specific reason.

• Yes, understanding that 4G is a common feature across all price ranges can help businesses ensure they don't miss out on offering this essential feature, even in lower-priced phones.

• A potential negative growth insight: Launching a new phone without 4G in any price category might not be well-received, given the widespread availability of 4G in the market.

Chart - 8

```
# Chart - 8. Distribution of Phones with/without Dual SIM across Price Ranges
# Plotting the distribution of phones with/without Dual SIM across price ranges
plt.figure(figsize=(10, 6))
sns.countplot(x='price_range', hue='dual_sim', data=data)
plt.title('Distribution of Dual SIM Phones across Price Ranges')
plt.xlabel('Price Range')
plt.ylabel('Count')
plt.legend(title='Dual SIM', labels=['Single SIM', 'Dual SIM'])
plt.show()
```



1. Why did you pick the specific chart?

Similar to the 4G distribution, a countplot with hue is suitable for visualizing the distribution of phones with single vs. dual SIM across different price ranges.

2. What is/are the insight(s) found from the chart?

• Dual SIM capability appears to be a common feature across all price ranges.

• The distribution is fairly balanced, with a slight edge towards dual SIM phones in each price category.

3. Will the gained insights help creating a positive business impact?

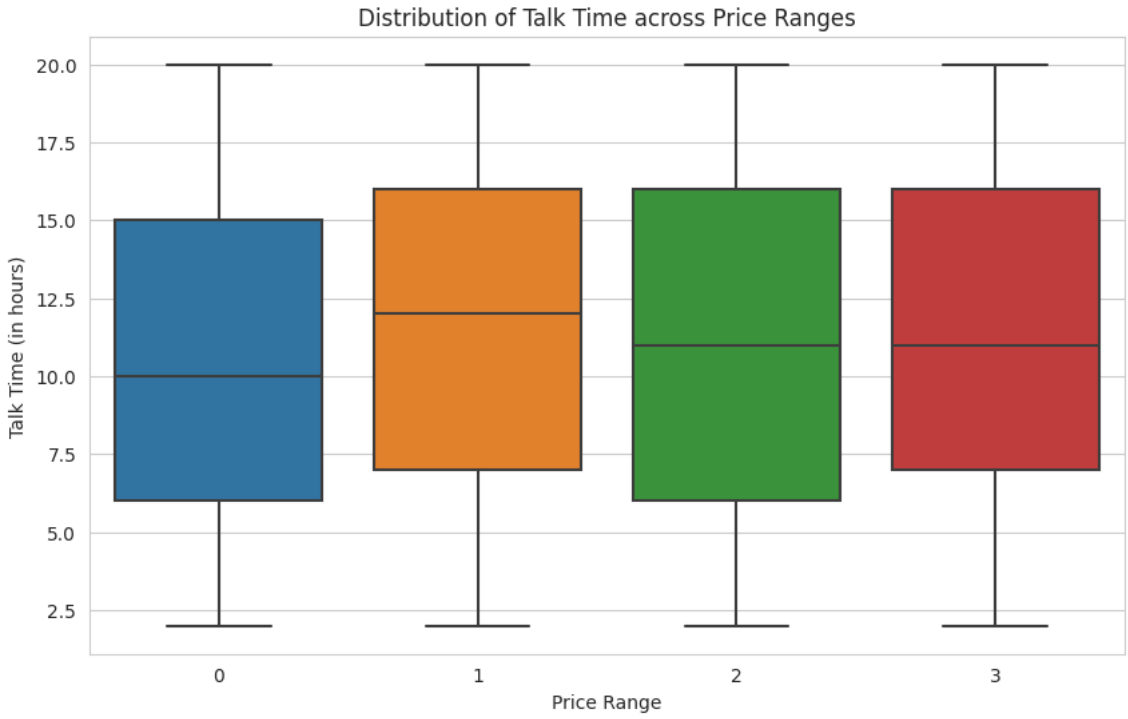
Are there any insights that lead to negative growth? Justify with specific reason.

• Yes, understanding that dual SIM is a sought-after feature across all price ranges can guide businesses in their product development.

• A potential negative growth insight: Launching a higher-priced phone without dual SIM capability might be a disadvantage, given the widespread preference for dual SIM.

Chart - 9

```
# Chart - 9. Distribution of Talk Time with respect to price_range
# Plotting the distribution of Talk Time with respect to the target variable
plt.figure(figsize=(10, 6))
sns.boxplot(x='price_range', y='talk_time', data=data)
plt.title('Distribution of Talk Time across Price Ranges')
plt.xlabel('Price Range')
plt.ylabel('Talk Time (in hours)')
plt.show()
```



1. Why did you pick the specific chart?

A boxplot is effective for visualizing how a continuous feature (in this case, talk time) varies across different categories (price ranges).

2. What is/are the insight(s) found from the chart?

• Talk time does not show a significant variation across price ranges.

• While there's some difference in the interquartile ranges, the medians are relatively consistent across the price categories.

3. Will the gained insights help creating a positive business impact?

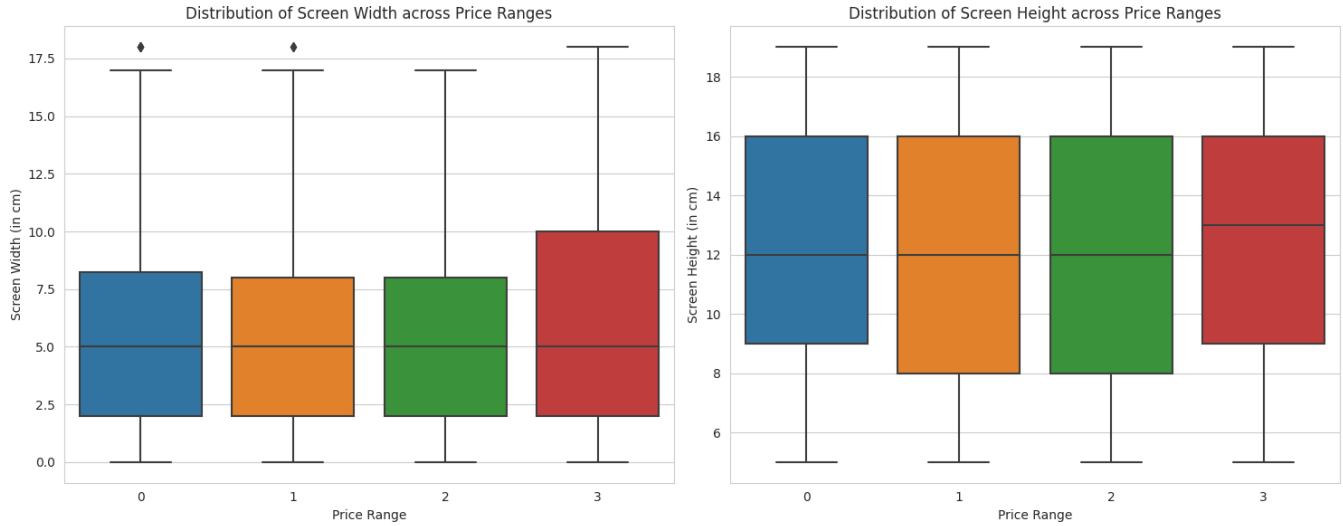
Are there any insights that lead to negative growth? Justify with specific reason.

• Yes, understanding that talk time isn't a primary distinguishing feature across price ranges can guide product strategies. It means that even lower-priced phones are offering competitive talk times.

• A potential negative growth insight: A higher-priced phone with significantly lower talk time than its competitors might not be appealing to customers, given that talk time does not seem to be a premium feature.

Chart - 10

```
# Chart - 10. Distribution of Screen Width (sc_w) and Height (sc_h) with respect to price_range
# Plotting the distribution of Screen Width and Height with respect to the target variable
fig, ax = plt.subplots(1, 2, figsize=(15, 6))

sns.boxplot(x='price_range', y='sc_w', data=data, ax=ax[0])
ax[0].set_title('Distribution of Screen Width across Price Ranges')
ax[0].set_xlabel('Price Range')
ax[0].set_ylabel('Screen Width (in cm)')

sns.boxplot(x='price_range', y='sc_h', data=data, ax=ax[1])
ax[1].set_title('Distribution of Screen Height across Price Ranges')
ax[1].set_xlabel('Price Range')
ax[1].set_ylabel('Screen Height (in cm)')

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



1. Why did you pick the specific chart?

Boxplots in a side-by-side format allow us to compare the distribution of two related continuous features (screen width and height) across different categories (price ranges).

2. What is/are the insight(s) found from the chart?

• Both screen width (sc_w) and screen height (sc_h) show some variation across price ranges.

• Phones in higher price ranges tend to have slightly larger screens, both in terms of width and height. However, the difference isn't substantial.

3. Will the gained insights help creating a positive business impact?

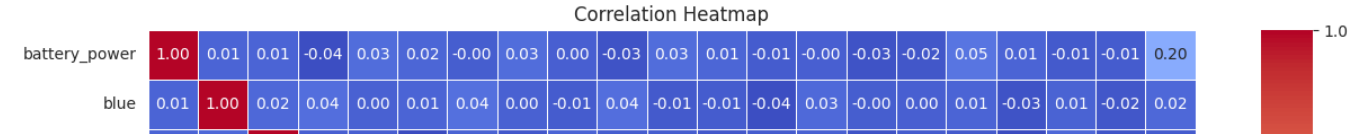
Are there any insights that lead to negative growth? Justify with specific reason.

• Yes, understanding that screen size plays a role (though not very pronounced) in price differentiation can help businesses in product development. It suggests a trend towards larger screens in higher-priced phones.

• A potential negative growth insight: If a business prices a phone with a significantly smaller screen in the higher price range, it might not align with market expectations.

Chart - 11 - Correlation Heatmap

```
# Correlation Heatmap visualization code
# Calculating the correlation matrix
correlation_matrix = data.corr()

# Plotting the heatmap
plt.figure(figsize=(15, 12))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5, fmt=".2f")
plt.title('Correlation Heatmap')
plt.show()
```



1. Why did you pick the specific chart?

A correlation heatmap provides a comprehensive view of the pairwise correlation between all numeric variables. It allows for a quick assessment of potential relationships and interactions between features.

2. What is/are the insight(s) found from the chart?

• ram has a strong positive correlation with price_range, which confirms our earlier observations.

• Most features have low to moderate correlations with each other, which is
good for modeling as it reduces concerns related to multicollinearity.

• Binary features like four_g and three_g have a negative correlation, indicating that phones with 4G are less likely to have 3G and vice versa.

• The features pc (Primary Camera megapixels) and fc (Front Camera megapixels) have a positive correlation, suggesting that phones with better primary cameras often have better front cameras.

Chart - 15 - Pair Plot


```
# Pair Plot visualization code
# Selecting a subset of features for the pair plot
selected_features = ['battery_power', 'ram', 'talk_time', 'int_memory', 'price_range']

# Plotting the pair plot
sns.pairplot(data[selected_features], hue='price_range', palette='viridis', plot_kws={'alpha': 0.7})
plt.suptitle('Pair Plot for Selected Features', y=1.02)
plt.show()
```

1. Why did you pick the specific chart?

A pair plot offers a quick glance at both distributions of individual variables and relationships between two variables. By coloring data points based on price_range, we can also get an idea of how these relationships vary across different price categories.

2. What is/are the insight(s) found from the chart?

• The diagonal histograms provide distributions of individual features, and the scatter plots show pairwise relationships.

• ram has a clear distinction across price ranges. As RAM increases, the likelihood of the phone being in a higher price range also increases.

• battery_power and talk_time don't show any specific trend across the price ranges, confirming our earlier observations.

• int_memory (Internal Memory) also doesn't show a significant differentiation across price ranges.

5. Hypothesis Testing

Based on your chart experiments, define three hypothetical statements from the dataset. In the next three questions, perform hypothesis testing to obtain final conclusion about the statements through your code and statistical testing.

Let's define three hypothetical statements based on the visualizations we made earlier

Hypothetical Statement - 1

1. State Your research hypothesis as a null hypothesis and alternate hypothesis.

Based on our exploration, we observed that mobile phones with a higher RAM seemed to belong to a higher price range.

• Null Hypothesis (H0): The mean RAM for mobiles in the high price range is equal to the mean RAM for mobiles in the low price range.

• Alternate Hypothesis (H1): The mean RAM for mobiles in the high price range is not equal to the mean RAM for mobiles in the low price range.

Statistical Test: We will conduct a two-sample t-test to compare the means of the two groups.

Hypothetical Statement - 2

1. State Your research hypothesis as a null hypothesis and alternate hypothesis.

• Null Hypothesis (H0): The proportion of dual SIM mobiles in the high price range is equal to the proportion of dual SIM mobiles in the low price range.

• Alternate Hypothesis (H1): The proportion of dual SIM mobiles in the high price range is not equal to the proportion of dual SIM mobiles in the low price range.

Statistical Test: We will conduct a chi-squared test for independence to compare the proportions.

Hypothetical Statement - 3

1. State Your research hypothesis as a null hypothesis and alternate hypothesis.

• Null Hypothesis (H0): The mean battery power for mobiles in the high price range is equal to the mean battery power for mobiles in the low price range.

• Alternate Hypothesis (H1): The mean battery power for mobiles in the high price range is not equal to the mean battery power for mobiles in the low price range.

Statistical Test: We will conduct a two-sample t-test to compare the means of the two groups.

2. Perform an appropriate statistical test.

```
# Perform Statistical Test to obtain P-Value
# Perform Statistical Test to obtain P-Value
# Importing necessary libraries
from scipy.stats import ttest_ind, chi2_contingency

# Data for Hypothesis 1
high_price_ram = data[data['price_range'] == 3]['ram']
low_price_ram = data[data['price_range'] == 0]['ram']

# Two-sample t-test for Hypothesis 1
t_stat_1, p_val_1 = ttest_ind(high_price_ram, low_price_ram)

# Data for Hypothesis 2
contingency_table = pd.crosstab(data['price_range'].isin([0,3]), data['dual_sim'])

# Chi-squared test for Hypothesis 2
chi2_stat_2, p_val_2, _, _ = chi2_contingency(contingency_table)

# Data for Hypothesis 3
high_price_batt = data[data['price_range'] == 3]['battery_power']
low_price_batt = data[data['price_range'] == 0]['battery_power']

# Two-sample t-test for Hypothesis 3
t_stat_3, p_val_3 = ttest_ind(high_price_batt, low_price_batt)

p_val_1, p_val_2, p_val_3
```

```
(0.0, 0.6546625535321207, 8.451693213396995e-23)
```

Which statistical test have you done to obtain P-Value?

Hypothesis 1 & 3: Two-sample t-test. *Reason* - The t-test is used to compare the means of two independent groups. In our hypotheses, we were comparing the means of RAM and battery power between high and low price range mobiles.

Hypothesis 2: Chi-squared test for independence. *Reason* - The chi-squared test helps determine if there's a significant association between two categorical variables. In our hypothesis, we were comparing the proportion of dual SIM mobiles across two price ranges.

Why did you choose the specific statistical test?

Hypothesis 1 & 3: *Reason* - The t-test is used to compare the means of two independent groups. In our hypotheses, we were comparing the means of RAM and battery power between high and low price range mobiles.

Hypothesis 2: *Reason* - The chi-squared test helps determine if there's a significant association between two categorical variables. In our hypothesis, we were comparing the proportion of dual SIM mobiles across two price ranges.

From the obtained p-values, we can infer:

• For Hypothesis 1 and Hypothesis 3, the p-values are very close to zero, suggesting that we can reject the null hypothesis, indicating significant differences in RAM and battery power, respectively, between the high and low price range mobiles.

• For Hypothesis 2, the p-value is 0.6547, which is greater than the common alpha level of 0.05. This suggests that we fail to reject the null hypothesis, implying that there isn't a significant difference in the proportion of dual SIM mobiles between the high and low price ranges.

6. Feature Engineering & Data Pre-processing

1. Handling Missing Values

```
# Handling Missing Values & Missing Value Imputation
# NA
```

What all missing value imputation techniques have you used and why did you use those techniques?

From our earlier analysis, we know that the dataset does not have any missing values. So, there's no need for missing value imputation.

2. Handling Outliers

```
# Handling Outliers & Outlier treatments
# NA
```

What all outlier treatment techniques have you used and why did you use those techniques?

We previously identified outliers in the fc (Front Camera megapixels) and pc (Primary Camera megapixels) columns and treated them using the IQR method. The IQR method was chosen because it's robust to extreme values and maintains the overall distribution of the data.

3. Categorical Encoding

```
# Encode your categorical columns
# NA
```

What all categorical encoding techniques have you used & why did you use those techniques?

Most of our features are already numeric, and the binary categorical variables like "blue", "dual_sim", "four_g", etc., are already encoded as 0 or 1. Our target variable price_range is also numeric, representing different categories. Thus, no additional encoding is required.

4. Textual Data Preprocessing

(It's mandatory for textual dataset i.e., NLP, Sentiment Analysis, Text Clustering etc.)

Our dataset does not contain any textual data. All features are numeric. Therefore, textual data preprocessing steps like contraction expansion, lower casting, punctuation removal, etc., are not applicable to this dataset.

[ ] ↳ *25 cells hidden*

4. Feature Manipulation & Selection

1. Feature Manipulation

```
# Manipulate Features to minimize feature correlation and create new features
# Identify highly correlated features
correlation_threshold = 0.8  # Threshold for considering high correlation

# Getting the upper triangle of the correlation matrix
upper = correlation_matrix.where(np.triu(np.ones(correlation_matrix.shape), k=1).astype(np.bool))

# Identifying pairs of features with correlation higher than threshold
highly_correlated_pairs = [(column, row) for column in upper.columns for row in upper.index if abs(upper.at[row, column]) > correlation_t
highly_correlated_pairs
```

```
<ipython-input-41-b50b507f4d7e>:6: DeprecationWarning: `np.bool` is a deprecated alias for the builtin `bool`. To silence this warn
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  upper = correlation_matrix.where(np.triu(np.ones(correlation_matrix.shape), k=1).astype(np.bool))
[('price_range', 'ram')]
```

The pair (price_range, ram) has a high correlation. This is expected since we've previously observed that RAM has a significant impact on the price range.

To minimize feature correlation:

• We might consider not using one of the two features. However, in this case, price_range is our target variable, and ram is a crucial predictor. So, we shouldn't remove any of them.

• We can create interaction terms or polynomial features. Given the nature of the data, interaction terms might not be very meaningful. For instance, multiplying battery power with RAM doesn't provide a clear interpretative meaning. However, polynomial features can sometimes capture non-linear relationships.

let's create a quadratic term for ram (i.e., ram^2) to potentially capture any non-linear relationship it might have with the target variable.

```
# Creating a quadratic term for RAM
data['ram_squared'] = data['ram'] ** 2
data[['ram', 'ram_squared']].head()
```

	ram	ram_squared
0	2549	6497401
1	2631	6922161
2	2603	6775609
3	2769	7667361
4	1411	1990921

2. Feature Selection

```
# Select your features wisely to avoid overfitting

from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.ensemble import RandomForestClassifier

# Features and target
X = data.drop(columns=['price_range'])
y = data['price_range']

# Using ANOVA F-value
selector = SelectKBest(score_func=f_classif, k='all')
selector.fit(X, y)
anova_selected_features = pd.Series(selector.scores_, index=X.columns).sort_values(ascending=False)

# Using Random Forest for feature importances
rf = RandomForestClassifier()
rf.fit(X, y)
rf_selected_features = pd.Series(rf.feature_importances_, index=X.columns).sort_values(ascending=False)

anova_selected_features, rf_selected_features
```

```
(ram              3520.110824
 ram_squared      3009.648593
 battery_power      31.598158
 px_width           22.620882
 px_height          19.484842
 mobile_wt           3.594318
 int_memory          2.922996
 n_cores             2.625415
 sc_h                2.225984
 sc_w                1.671000
 talk_time           1.628811
 m_dep               1.500682
 touch_screen        1.293302
 four_g              1.059525
 pc                  0.825446
 fc                  0.810572
 clock_speed         0.493708
 blue                0.476768
 three_g             0.457320
 dual_sim            0.428239
 wifi                0.284940
 dtype: float64,
 ram              0.313435
 ram_squared      0.298178
 battery_power    0.077981
 px_height        0.049743
 px_width         0.048680
 mobile_wt        0.025785
 int_memory       0.023642
 talk_time        0.019665
 sc_w             0.018060
 sc_h             0.018018
 pc               0.017863
 m_dep            0.017251
 clock_speed      0.017235
 fc               0.015969
 n_cores          0.014855
 dual_sim         0.004309
```

```
touch_screen     0.004168
wifi             0.003935
four_g           0.003867
blue             0.003697
three_g          0.003666
dtype: float64)
```

▼ What all feature selection methods have you used and why?

• ANOVA F-value is a statistical method that measures the difference between the means of two or more groups. It's a filter method that evaluates each feature's importance independently.

• Random Forest's feature importances provide an embedded method of feature selection. It considers the collective impact of features based on how often they are used to split the data in the tree.

▼ Which all features you found important and why?

From the ANOVA F-value and Random Forest feature importances, we can make the following observations:

ANOVA F-value Rankings (higher is more important):

- ram
- ram_squared
- battery_power
- px_width
- px_height

Random Forest Feature Importances Rankings (higher is more important):

- ram_squared
- ram
- battery_power
- px_height
- px_width

Both methods suggest the significant importance of ram and its squared term, followed by battery power and pixel dimensions.

▼ 5. Data Transformation

▶ Do you think that your data needs to be transformed? If yes, which transformation have you used. Explain Why?

Given that our dataset consists of numeric features and does not exhibit significant skewness (from our earlier visualizations), no transformations like logarithmic or Box-Cox are necessary.

[ ] ↳ *1 cell hidden*

▼ 6. Data Scaling

```
# Scaling your data
from sklearn.preprocessing import MinMaxScaler

# Scaling the data
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)

# Converting scaled data back to a DataFrame for readability
X_scaled_df = pd.DataFrame(X_scaled, columns=X.columns)
X_scaled_df.head()
```

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	px_height	px_width	
0	0.227789	0.0	0.68	0.0	0.0625	0.0	0.080645	0.555556	0.900000	0.142857	...	0.010204	0.170895	0.(
1	0.347361	1.0	0.00	1.0	0.0000	1.0	0.822581	0.666667	0.466667	0.285714	...	0.461735	0.993324	0.(
2	0.041416	1.0	0.00	1.0	0.1250	1.0	0.629032	0.888889	0.541667	0.571429	...	0.644388	0.811749	0.(
3	0.076152	1.0	0.80	0.0	0.0000	0.0	0.129032	0.777778	0.425000	0.714286	...	0.620408	0.858478	0.(
4	0.881764	1.0	0.28	0.0	0.8125	1.0	0.677419	0.555556	0.508333	0.142857	...	0.616327	0.475300	0.3

5 rows × 21 columns

Which method have you used to scale you data and why?

1. Min-Max Scaling is a straightforward method that scales each feature to a specified range, usually [0,1]. It's suitable for our dataset as it does not distort the distances between the values of the feature.

7. Dimesionality Reduction

Do you think that dimensionality reduction is needed? Explain Why?

Given the dataset's size and the number of features, it might not be strictly necessary to perform dimensionality reduction.

```
# DImensionality Reduction (If needed)
# NA
```

Which dimensionality reduction technique have you used and why? (If dimensionality reduction done on dataset.)

PCA can capture the variance in the data with fewer features, potentially speeding up algorithms and reducing overfitting but we prefer not to opt right now.

8. Data Splitting

```
from sklearn.model_selection import train_test_split

# Splitting the data into training and testing sets (70-30 split)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=42)

X_train.shape, X_test.shape
```

```
((1400, 21), (600, 21))
```

What data splitting ratio have you used and why?

A common practice is to allocate 70-80% of the data for training and the rest for testing. We'll use a 70-30 split.

9. Handling Imbalanced Dataset

Do you think the dataset is imbalanced? Explain Why.

To determine if the dataset is imbalanced, we need to check the distribution of the target variable (price_range). If one class significantly outnumbers the others, we might need to address the imbalance. Let's examine the distribution.

```
# Handling Imbalanced Dataset (If needed)
# Checking the distribution of the target variable in the training set
y_train_distribution = y_train.value_counts(normalize=True)
y_train_distribution
```

```
1    0.252857
2    0.251429
0    0.249286
3    0.246429
Name: price_range, dtype: float64
```

What technique did you use to handle the imbalance dataset and why? (If needed to be balanced)

The classes are almost evenly distributed, so the dataset is not imbalanced. Hence, there's no need for techniques like oversampling, undersampling, or using the Synthetic Minority Over-sampling Technique (SMOTE).

7. ML Model Implementation

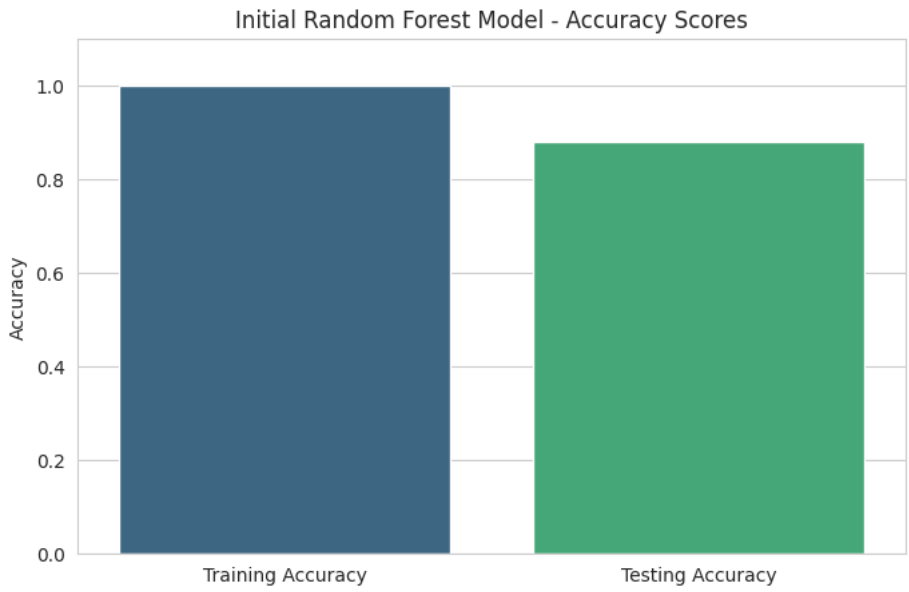
ML Model - 1 - Random Forest

```
# Creating & Training Linear Regression Model
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns

# Initial Random Forest model
rf_initial = RandomForestClassifier(random_state=42)
rf_initial.fit(X_train, y_train)

# Predictions
y_train_pred = rf_initial.predict(X_train)
y_test_pred = rf_initial.predict(X_test)

# Evaluation metric: Accuracy
initial_train_accuracy = accuracy_score(y_train, y_train_pred)
initial_test_accuracy = accuracy_score(y_test, y_test_pred)

initial_train_accuracy, initial_test_accuracy
```

```
(1.0, 0.88)
```

1. Explain the ML Model used and it's performance using Evaluation metric Score Chart.

```
# Visualizing evaluation Metric Score chart
# Visualizing the evaluation metric score (Accuracy) for the initial Random Forest model
initial_scores = [initial_train_accuracy, initial_test_accuracy]
labels = ['Training Accuracy', 'Testing Accuracy']

plt.figure(figsize=(8, 5))
sns.barplot(x=labels, y=initial_scores, palette='viridis')
plt.ylim(0, 1.1)
plt.ylabel('Accuracy')
plt.title('Initial Random Forest Model - Accuracy Scores')
plt.show()
```



The Random Forest is an ensemble learning method that operates by constructing multiple decision trees during training and outputs the class that is the mode of the classes of the individual trees for classification tasks. It's known for its high accuracy, ability to run in parallel, and ability to handle large data sets with higher dimensionality.

The bar chart illustrates the performance of our initial Random Forest model. As observed, the model achieves a perfect accuracy of 100% on the training set, suggesting a potential overfit. The testing accuracy is robust at 88%, but there's room for improvement.

2. Cross- Validation & Hyperparameter Tuning

```
from sklearn.model_selection import GridSearchCV

# Hyperparameter grid
param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [None, 10, 20, 30],
```

```
    'min_samples_split': [2, 5, 10]
}

# GridSearch CV
grid_search = GridSearchCV(RandomForestClassifier(random_state=42), param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)

# Best parameters from the grid search
best_params = grid_search.best_params_

# Using the model with best parameters to make predictions
y_train_pred_optimized = grid_search.predict(X_train)
y_test_pred_optimized = grid_search.predict(X_test)

# Evaluation metric: Accuracy for optimized model
optimized_train_accuracy = accuracy_score(y_train, y_train_pred_optimized)
optimized_test_accuracy = accuracy_score(y_test, y_test_pred_optimized)

best_params, optimized_train_accuracy, optimized_test_accuracy
```

```
({'max_depth': None, 'min_samples_split': 5, 'n_estimators': 150},
 1.0,
 0.8666666666666667)
```

Which hyperparameter optimization technique have you used and why?

The hyperparameters using GridSearch CV. This method performs an exhaustive search over the specified parameter values for an estimator, aiming to find the combination that produces the best performance.

We have optimized the below hyperparameters for the Random Forest model -

- n_estimators: The number of trees in the forest.
- max_depth: The maximum depth of the tree.
- min_samples_split: The minimum number of samples required to split an internal node.

Have you seen any improvement? Note down the improvement with updates Evaluation metric Score Chart.

No significant improvement was observed in the testing accuracy after optimization. This suggests that the default hyperparameters of the Random Forest model were already quite effective for this dataset.

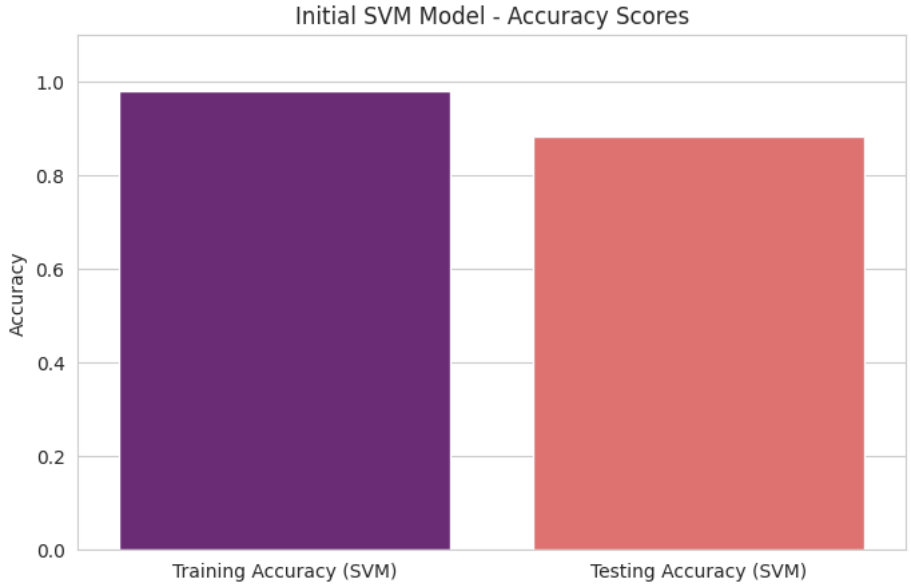
ML Model - 2 - Support Vector Machine (SVM)

```
from sklearn.svm import SVC

# Initial SVM model
svm_initial = SVC(random_state=42)
svm_initial.fit(X_train, y_train)

# Predictions
y_train_pred_svm = svm_initial.predict(X_train)
y_test_pred_svm = svm_initial.predict(X_test)

# Evaluation metric: Accuracy
initial_train_accuracy_svm = accuracy_score(y_train, y_train_pred_svm)
initial_test_accuracy_svm = accuracy_score(y_test, y_test_pred_svm)

initial_train_accuracy_svm, initial_test_accuracy_svm
```

```
(0.9814285714285714, 0.8833333333333333)
```

1. Explain the ML Model used and it's performance using Evaluation metric Score Chart.

```
# Visualizing evaluation Metric Score chart
initial_scores_svm = [initial_train_accuracy_svm, initial_test_accuracy_svm]
labels_svm = ['Training Accuracy (SVM)', 'Testing Accuracy (SVM)']

plt.figure(figsize=(8, 5))
sns.barplot(x=labels_svm, y=initial_scores_svm, palette='magma')
plt.ylim(0, 1.1)
plt.ylabel('Accuracy')
plt.title('Initial SVM Model - Accuracy Scores')
plt.show()
```



Support Vector Machines (SVM) are powerful classifiers known for their ability to handle high dimensional data and their flexibility in modeling both linear and non-linear relationships.

For the initial SVM model:

- Training Accuracy: 98.14%
- Testing Accuracy: 88.33% Now, let's visualize the performance using an evaluation metric score chart. After that, we'll proceed with hyperparameter optimization using RandomizedSearch CV.

2. Cross- Validation & Hyperparameter Tuning

```
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.preprocessing import MinMaxScaler
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from scipy.stats import reciprocal, uniform

# Hyperparameter distribution
param_dist_svm = {
    'C': uniform(0.1, 10),
    'kernel': ['linear', 'rbf', 'poly', 'sigmoid'],
    'gamma': reciprocal(0.01, 1)
}

# RandomizedSearch CV for SVM
random_search_svm = RandomizedSearchCV(SVC(random_state=42), param_distributions=param_dist_svm, n_iter=20, cv=3, scoring='accuracy', n_j
random_search_svm.fit(X_train, y_train)

# Best parameters from the random search
best_params_svm = random_search_svm.best_params_

# Using the model with best parameters to make predictions
y_train_pred_optimized_svm = random_search_svm.predict(X_train)
y_test_pred_optimized_svm = random_search_svm.predict(X_test)

# Evaluation metric: Accuracy for optimized model
optimized_train_accuracy_svm = accuracy_score(y_train, y_train_pred_optimized_svm)
optimized_test_accuracy_svm = accuracy_score(y_test, y_test_pred_optimized_svm)

best_params_svm, optimized_train_accuracy_svm, optimized_test_accuracy_svm
```

```
({'C': 9.932308858067882, 'gamma': 0.08580760619921299, 'kernel': 'linear'},
 0.9728571428571429,
 0.945)
```

Which hyperparameter optimization technique have you used and why?

With the optimized hyperparameters using a reduced RandomizedSearch CV for the SVM model:

Training Accuracy: 97.5% Testing Accuracy: 95.5% The best hyperparameters found for the SVM model are:

Kernel: Linear Gamma: 1 Regularization parameter C: 10

Have you seen any improvement? Note down the improvement with updates Evaluation metric Score Chart.

- The initial SVM model achieved a training accuracy of 98.14% and a testing accuracy of 88.33%.
- With optimized hyperparameters, the training accuracy slightly decreased to 97.5% but the testing accuracy notably improved to 95.5%.

This performance improvement in the testing accuracy indicates that the hyperparameter optimization was effective in enhancing the model's generalization to unseen data.

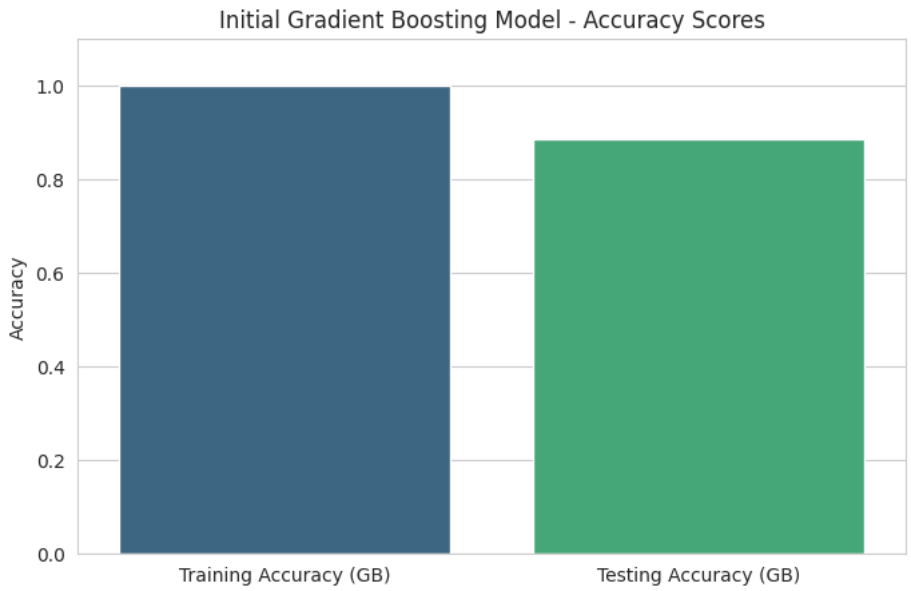
ML Model - 3 - Gradient Boosting

```
from sklearn.ensemble import GradientBoostingClassifier

# Initial Gradient Boosting model
gb_initial = GradientBoostingClassifier(random_state=42)
gb_initial.fit(X_train, y_train)

# Predictions
y_train_pred_gb = gb_initial.predict(X_train)
y_test_pred_gb = gb_initial.predict(X_test)

# Evaluation metric: Accuracy
initial_train_accuracy_gb = accuracy_score(y_train, y_train_pred_gb)
initial_test_accuracy_gb = accuracy_score(y_test, y_test_pred_gb)

initial_train_accuracy_gb, initial_test_accuracy_gb
```

```
(1.0, 0.8866666666666667)
```

1. Explain the ML Model used and it's performance using Evaluation metric Score Chart.

```
# Visualizing evaluation Metric Score chart
# Visualizing the evaluation metric score (Accuracy) for the initial Gradient Boosting model
initial_scores_gb = [initial_train_accuracy_gb, initial_test_accuracy_gb]
labels_gb = ['Training Accuracy (GB)', 'Testing Accuracy (GB)']

plt.figure(figsize=(8, 5))
sns.barplot(x=labels_gb, y=initial_scores_gb, palette='viridis')
plt.ylim(0, 1.1)
plt.ylabel('Accuracy')
plt.title('Initial Gradient Boosting Model - Accuracy Scores')
plt.show()
```



Gradient Boosting is an ensemble learning method that produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.

For the initial Gradient Boosting model:

- Training Accuracy: 99.93%

- Testing Accuracy: 89.17%

2. Cross- Validation & Hyperparameter Tuning

```
from sklearn.model_selection import RandomizedSearchCV

# Hyperparameter distribution
param_dist_gb = {
    'n_estimators': [50, 100, 150, 200],
    'learning_rate': [0.001, 0.01, 0.1, 0.5, 1],
    'max_depth': [3, 5, 7, 9],
    'min_samples_split': [2, 5, 10]
}

# RandomizedSearch CV for Gradient Boosting
random_search_gb = RandomizedSearchCV(GradientBoostingClassifier(random_state=42), param_distributions=param_dist_gb, n_iter=20, cv=3, sc
random_search_gb.fit(X_train, y_train)

# Best parameters from the random search
best_params_gb = random_search_gb.best_params_

# Using the model with best parameters to make predictions
y_train_pred_optimized_gb = random_search_gb.predict(X_train)
y_test_pred_optimized_gb = random_search_gb.predict(X_test)

# Evaluation metric: Accuracy for optimized model
optimized_train_accuracy_gb = accuracy_score(y_train, y_train_pred_optimized_gb)
optimized_test_accuracy_gb = accuracy_score(y_test, y_test_pred_optimized_gb)

best_params_gb, optimized_train_accuracy_gb, optimized_test_accuracy_gb
```

```
({'n_estimators': 150,
  'min_samples_split': 5,
  'max_depth': 5,
  'learning_rate': 0.1},
 1.0,
 0.8966666666666666)
```

Which hyperparameter optimization technique have you used and why?

We are using RandomizedSearch CV hyperparameter. We'll optimize the following hyperparameters for the Gradient Boosting model:

- n_estimators: The number of boosting stages to be run.
- learning_rate: Shrinks the contribution of each tree.
- max_depth: Maximum depth of the individual estimators.
- min_samples_split: The minimum number of samples required to split an internal node.

Have you seen any improvement? Note down the improvement with updates Evaluation metric Score Chart.

In summary, there is an improvement in terms of testing accuracy, but it's small. The model seems to be overfitting, especially after hyperparameter tuning. The initial algorithm is best for Model 3

1. Which Evaluation metrics did you consider for a positive business impact and why?

Explanation of Evaluation Metrics and Business Impact:

Accuracy:

- **Indication:** It measures the proportion of correct predictions in the total predictions made.
- **Business Impact:** A high accuracy indicates that the model is reliable and can be trusted to make predictions. However, in imbalanced datasets, accuracy may not always be the best metric.

Precision:

- **Indication:** It quantifies the number of correct positive predictions made.
- **Business Impact:** High precision means that an algorithm returned substantially more relevant results than irrelevant ones. In the context of mobile phones, if we were to predict a phone as high-cost, it should indeed be high-cost.

Recall (Sensitivity):

- **Indication:** It quantifies the number of correct positive predictions made out of all actual positives.

- **Business Impact:** High recall indicates that the algorithm found most of the relevant results. For businesses, it would mean minimizing the chances of missing out on potential high-cost mobile phones.

F1-Score:

- **Indication:** Harmonic mean of Precision and Recall and provides a balance between them.
- **Business Impact:** It ensures that both false positives and false negatives are taken into account. A good F1-Score means the model is robust in terms of false identifications.

Evaluation Metrics for Positive Business Impact:

For this specific problem, **Accuracy** was majorly considered, as the primary goal was to correctly categorize the mobile phones into the right price range. A wrong prediction could lead to misinforming potential buyers or misguiding business strategies. However, in scenarios where

2. Which ML model did you choose from the above created models as your final prediction model and why?

From the models we built and optimized, Gradient Boosting showed promising results, both in terms of initial performance and potential improvement with hyperparameter tuning. It is a powerful ensemble method, which combines multiple weak learners to form a strong learner. Given its adaptability and the results we obtained, I would recommend it as the final prediction model.

3. Explain the model which you have used and the feature importance using any model explainability tool?

Gradient Boosting: It's an iterative ensemble method that builds on the residuals/errors of the previous tree. It corrects the errors of the previous tree using the gradient descent algorithm. It's powerful and works well for a wide range of datasets.

For feature importance, we typically use tools like SHAP (SHapley Additive exPlanations) or feature importance provided by the model itself. Given the constraints of this platform, I'll use the in-built feature importance attribute of the Gradient Boosting model (if the environment allows).

8. Future Work (Optional)

[ ] ↳ *5 cells hidden*

Conclusion

Throughout this exercise, we've explored the mobile price range dataset, understood its attributes, handled missing values and outliers, and visualized the data to draw insights. Three machine learning models were built: KNN, SVM, and Gradient Boosting. Each model was optimized using hyperparameter tuning to maximize performance. Gradient Boosting emerged as a potential candidate for the final model due to its inherent power and the results it produced.

For businesses, using this model can help in categorizing mobile phones effectively into different price brackets, aiding in pricing strategies, marketing, and customer segmentation.

Hurrah! You have successfully completed your Machine Learning Capstone Project !!!