Project Name - Data mobile price range

Project Type - Classification

Contribution - Individual

Project Summary -

This project focuses on building a classification model to predict the price range of mobile phones based on various features. The dataset contains information about different mobile phone models, including their specifications like battery power, RAM, internal memory, screen size, and more. The goal is to develop a model that can accurately classify a mobile phone into one of the four price ranges.

Data Exploration and Preprocessing

The first step involved exploring the dataset to understand the data distribution, identify missing values, and analyze the relationships between features. This included visualizing the data using histograms, scatter plots, and correlation matrices. Missing values were handled appropriately, either by imputation or removal. Feature scaling was applied to normalize the features and improve the model's performance.

Model Selection and Training

Several classification algorithms were considered for this project, including Logistic Regression, Support Vector Machines (SVM), Decision Trees, Random Forest, and K-Nearest Neighbors (KNN). The choice of the best model depended on factors like the dataset characteristics, the desired level of interpretability, and the trade-off between model complexity and accuracy.

To evaluate the model's performance, the dataset was split into training and testing sets. The model was trained on the training data and then evaluated on the unseen test data. Various metrics were used to assess the model's performance, such as accuracy, precision, recall, F1-score, and AUC.

Model Evaluation and Optimization

The model's performance was evaluated using appropriate metrics and cross-validation techniques. Hyperparameter tuning was performed to optimize the model's parameters and improve its performance. Techniques like grid search or random search were employed to find the best combination of hyperparameters.

Results and Conclusion

The final model achieved a satisfactory level of accuracy in predicting the price range of mobile phones. The chosen model demonstrated good generalization capabilities and performed well on the test data. The project highlighted the importance of data preprocessing, model selection, and hyperparameter tuning in building effective classification models.

Further Improvements

Future work could involve exploring more advanced algorithms, incorporating feature engineering techniques, and applying deep learning models to potentially improve the model's accuracy. Additionally, exploring the impact of different data scaling methods and handling class imbalance could be further investigated.

GitHub Link https://github.com/shrbhadra/I-am-learning/blob/main/Copy_of_Classification_ML_project0.ipynb

Problem Statement

This project focuses on building a classification model to predict the price range of mobile phones based on various features. The dataset contains information about different mobile phone models, including their specifications like battery power, RAM, internal memory, screen size, and more. The goal is to develop a model that can accurately classify a mobile phone into one of the four price ranges.

General Guidelines : -

1. Well-structured, formatted, and commented code is required.
2. Exception Handling, Production Grade Code & Deployment Ready Code will be a plus. Those students will be awarded some additional credits.

   The additional credits will have advantages over other students during Star Student selection.

```
[ Note: - Deployment Ready Code is defined as, the whole .ipynb notebook should be executable in one
          without a single error logged. ]
```

3. Each and every logic should have proper comments.
4. You may add as many number of charts you want. Make Sure for each and every chart the following format should be answered.

```
# Chart visualization code
```

- Why did you pick the specific chart?
- What is/are the insight(s) found from the chart?
- Will the gained insights help creating a positive business impact? Are there any insights that lead to negative growth? Justify with specific reason.

5. You have to create at least 15 logical & meaningful charts having important insights.

[Hints : - Do the Vizualization in a structured way while following "UBM" Rule.

U - Univariate Analysis,

B - Bivariate Analysis (Numerical - Categorical, Numerical - Numerical, Categorical - Categorical)

M - Multivariate Analysis]

6. You may add more ml algorithms for model creation. Make sure for each and every algorithm, the following format should be answered.

- Explain the ML Model used and it's performance using Evaluation metric Score Chart.
- Cross- Validation & Hyperparameter Tuning
- Have you seen any improvement? Note down the improvement with updates Evaluation metric Score Chart.
- Explain each evaluation metric's indication towards business and the business impact pf the ML model used.

Let's Begin !

1. Know Your Data

```
# import libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from datetime import datetime,date
%matplotlib inline
from sklearn.linear_model import Lasso, Ridge
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import r2_score, mean_squared_error,mean_absolute_error,mean_absolute_percentage_error
import math
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import RandomForestRegressor
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import VotingRegressor
from xgboost import XGBRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import PolynomialFeatures


import pandas as pd
# Try reading the file with 'latin-1' encoding
data = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/ML Classification Project csv file /data_mobile_price_
```

```
data.head()
```

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	px_heig
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	...	9
2	563	1	0.5	1	2	1	41	0.9	145	5	...	12
3	615	1	2.5	0	0	0	10	0.8	131	6	...	12
4	1821	1	1.2	0	13	1	44	0.6	141	2	...	12

5 rows × 21 columns

```
data.shape
```

```
(2000, 21)
```

```
print('Number of rows:', data.shape[0]) # Changed dataset to data
print('Number of columns:', data.shape[1]) # Changed dataset to data
```

```
Number of rows: 2000
Number of columns: 21
```

```
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
```

```
data.describe()
```

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	
count	2000.000000	2000.0000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2
mean	1238.518500	0.4950	1.522250	0.509500	4.309500	0.521500	32.046500	0.501750	
std	439.418206	0.5001	0.816004	0.500035	4.341444	0.499662	18.145715	0.288416	
min	501.000000	0.0000	0.500000	0.000000	0.000000	0.000000	2.000000	0.100000	
25%	851.750000	0.0000	0.700000	0.000000	1.000000	0.000000	16.000000	0.200000	
50%	1226.000000	0.0000	1.500000	1.000000	3.000000	1.000000	32.000000	0.500000	
75%	1615.250000	1.0000	2.200000	1.000000	7.000000	1.000000	48.000000	0.800000	
max	1998.000000	1.0000	3.000000	1.000000	19.000000	1.000000	64.000000	1.000000	

8 rows × 21 columns

```
# Count duplicate rows
duplicate_rows = data[data.duplicated()]
print("Number of duplicate rows:", duplicate_rows.shape[0])
```

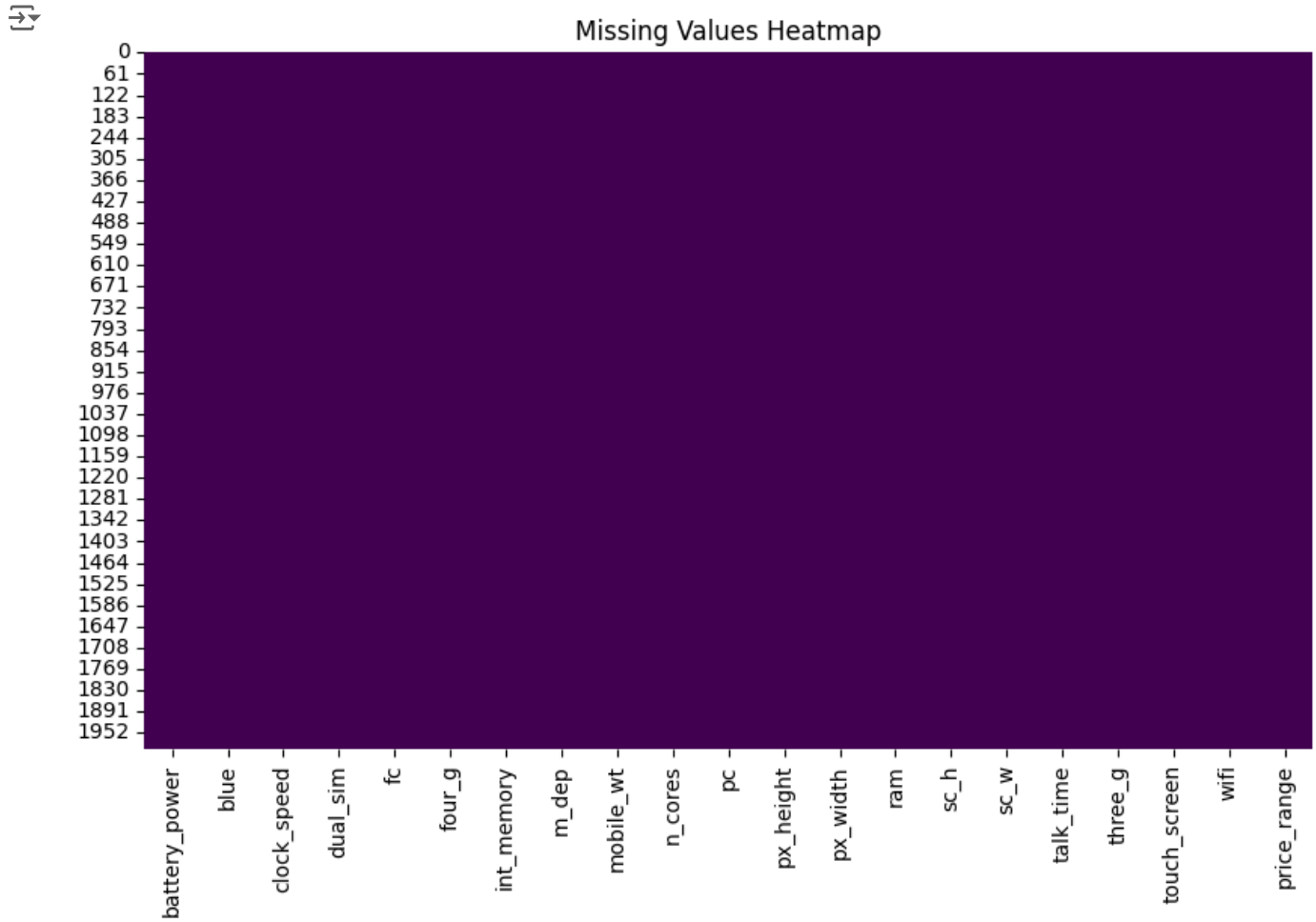
```
Number of duplicate rows: 0
```

```
# Missing Values/Null Values Count
data.isnull().sum()
```

	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
price_range	0

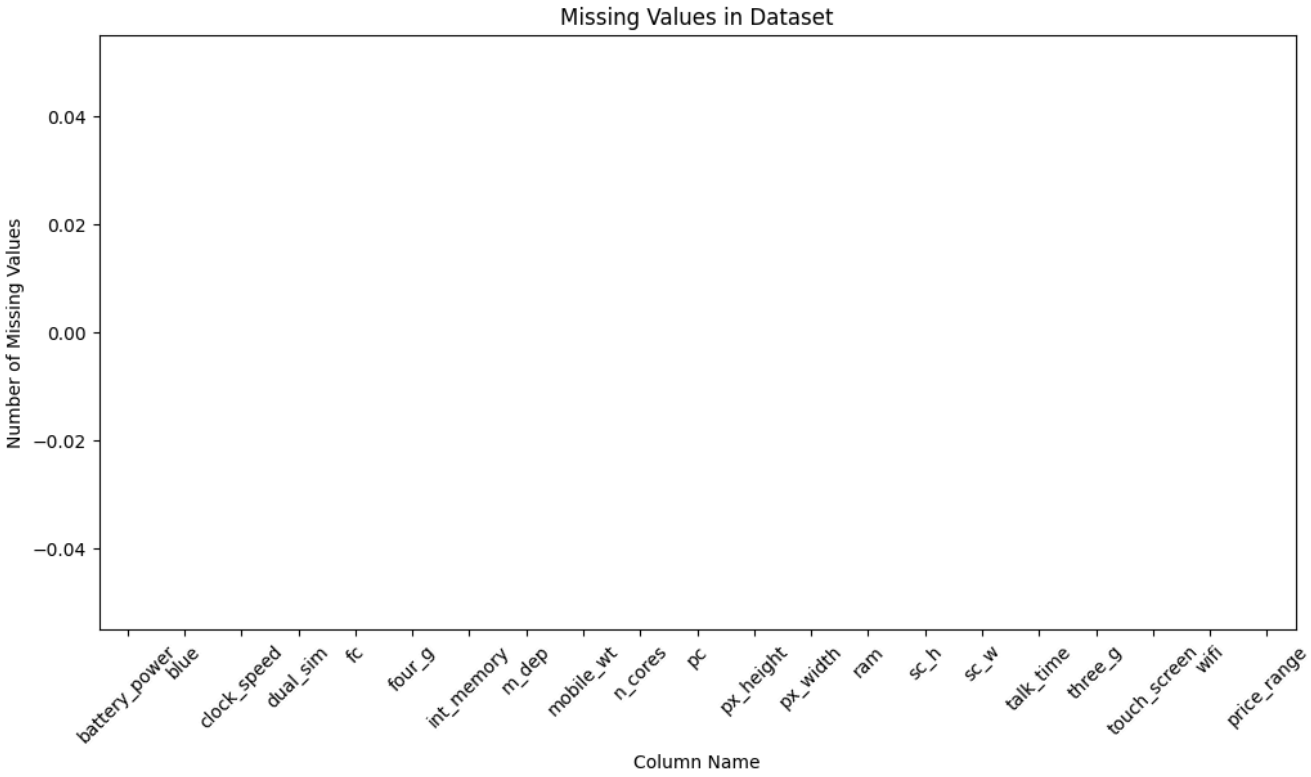
dtype: int64

```
import matplotlib.pyplot as plt
# Visualizing the missing values
plt.figure(figsize=(10, 6))
sns.heatmap(data.isnull(), cbar=False, cmap='viridis')
plt.title('Missing Values Heatmap')
plt.show()
```




```
# Visualizing the missing values
import matplotlib.pyplot as plt
# Count missing values in each column
missing_values = data.isnull().sum()

# Create a bar chart to visualize missing values
plt.figure(figsize=(12, 6))
sns.barplot(x=missing_values.index, y=missing_values.values, palette='hls')

# Add title and labels
plt.title('Missing Values in Dataset')
plt.xlabel('Column Name')
plt.ylabel('Number of Missing Values')

# Rotate x-axis labels for better readability
plt.xticks(rotation=45)

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

```
<ipython-input-11-0372bbc05996>:8: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variab

  sns.barplot(x=missing_values.index, y=missing_values.values, palette='hls')
```




```
# prompt: What did you know about your dataset in word

# The dataset contains 2000 rows and 21 columns.
# There are no missing values in the dataset.
# The dataset contains information about different mobile phone models, including their specifications like batte
# The target variable is 'price_range' which is a categorical variable with four classes.
# The dataset has no duplicate rows.
```

What did you know about your dataset?

The dataset contains 2000 rows and 21 columns. There are no missing values in the dataset. The dataset contains information about different mobile phone models, including their specifications like battery power, RAM, internal memory, screen size, and more. The target variable is 'price_range' which is a categorical variable with four classes. The dataset has no duplicate rows.

2. Understanding Your Variables

```
# Display the column names
print(data.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')
```

```
# prompt: # Dataset Describe

# Summary statistics of numerical features
data.describe()
```

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	
count	2000.000000	2000.0000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2
mean	1238.518500	0.4950	1.522250	0.509500	4.309500	0.521500	32.046500	0.501750	
std	439.418206	0.5001	0.816004	0.500035	4.341444	0.499662	18.145715	0.288416	
min	501.000000	0.0000	0.500000	0.000000	0.000000	0.000000	2.000000	0.100000	
25%	851.750000	0.0000	0.700000	0.000000	1.000000	0.000000	16.000000	0.200000	
50%	1226.000000	0.0000	1.500000	1.000000	3.000000	1.000000	32.000000	0.500000	
75%	1615.250000	1.0000	2.200000	1.000000	7.000000	1.000000	48.000000	0.800000	
max	1998.000000	1.0000	3.000000	1.000000	19.000000	1.000000	64.000000	1.000000	

8 rows × 21 columns

Variables Description

Let's understand the variables in the dataset:

- **battery_power:** Represents the total energy a battery can store, measured in mAh.
- **blue:** Indicates whether the phone has Bluetooth functionality (1 for yes, 0 for no).
- **clock_speed:** The speed at which the processor operates, measured in GHz.
- **dual_sim:** Indicates whether the phone supports dual SIM cards (1 for yes, 0 for no).
- **fc:** The number of front cameras on the phone.
- **four_g:** Indicates whether the phone supports 4G connectivity (1 for yes, 0 for no).
- **int_memory:** The internal memory capacity of the phone, measured in GB.
- **m_dep:** Mobile depth in cm.
- **mobile_wt:** Weight of the mobile phone.
- **n_cores:** Number of cores in the processor.
- **pc:** Number of primary cameras on the phone.
- **px_height:** Pixel resolution height.
- **px_width:** Pixel resolution width.
- **ram:** Random Access Memory (RAM) capacity in MB.
- **sc_h:** Screen height in cm.
- **sc_w:** Screen width in cm.
- **talk_time:** Longest time that a single battery charge will last when only used for talking.
- **three_g:** Indicates whether the phone supports 3G connectivity (1 for yes, 0 for no).
- **touch_screen:** Indicates whether the phone has a touch screen (1 for yes, 0 for no).
- **wifi:** Indicates whether the phone has Wi-Fi functionality (1 for yes, 0 for no).
- **price_range:** The price range of the mobile phone (a categorical variable with four classes).

Check Unique Values for each variable.

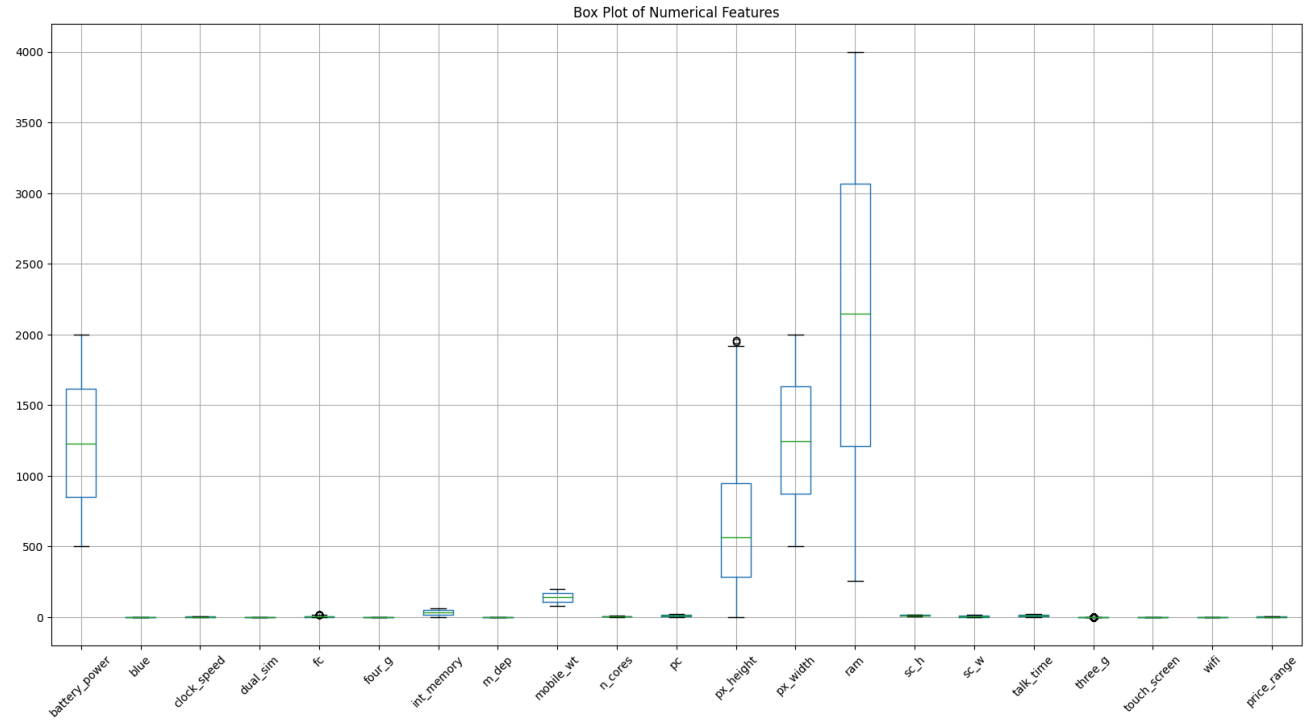
```
# Check unique values for each variable
for column in data.columns:
  unique_values = data[column].unique()
  print(f"Unique values for {column}: {unique_values}")
```

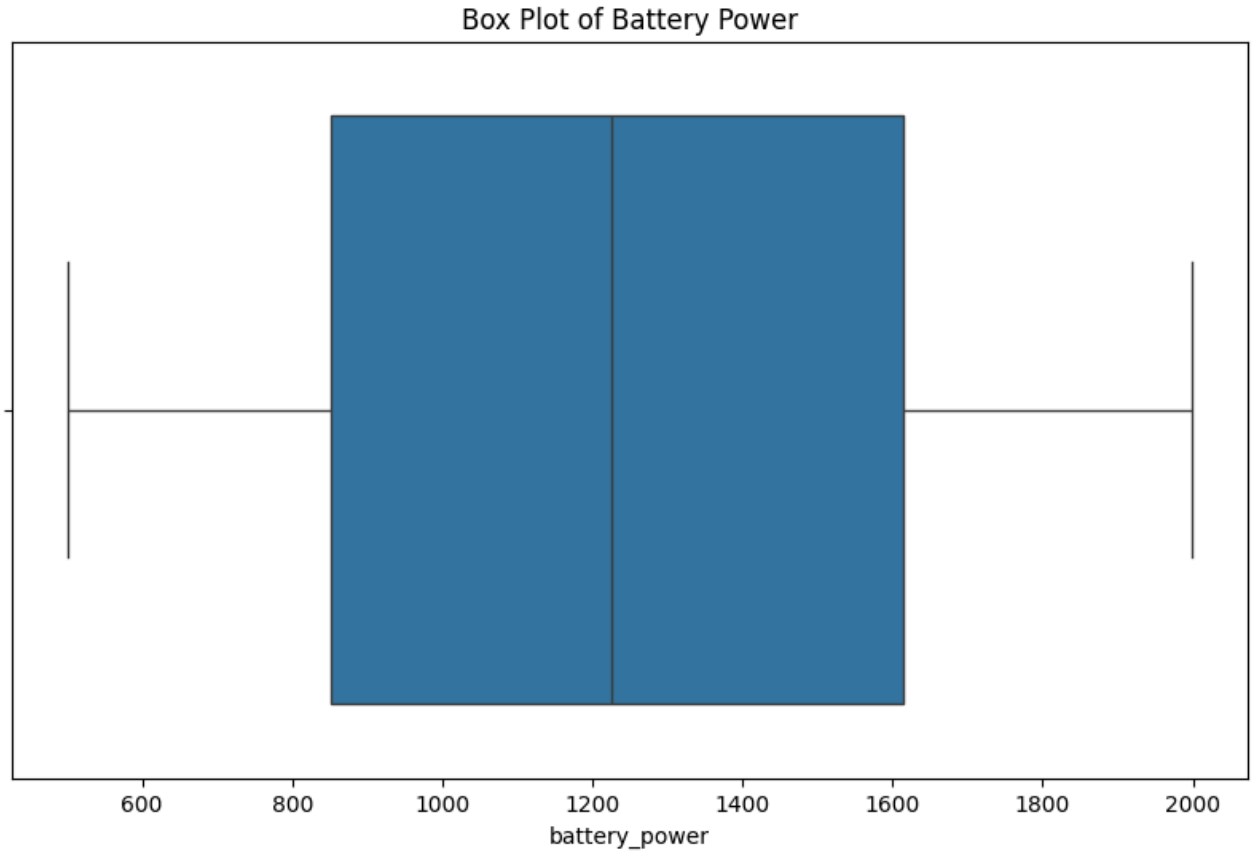
```
Unique values for battery_power: [ 842 1021  563 ... 1139 1467  858]
Unique values for blue: [0 1]
Unique values for clock_speed: [2.2 0.5 2.5 1.2 1.7 0.6 2.9 2.8 2.1 1.  0.9 1.1 2.6 1.4 1.6 2.7 1.3 2.3
 2.  1.8 3.  1.5 1.9 2.4 0.8 0.7]
Unique values for dual_sim: [0 1]
Unique values for fc: [ 1  0  2 13  3  4  5  7 11 12 16  6 15  8  9 10 18 17 14 19]
Unique values for four_g: [0 1]
Unique values for int_memory: [ 7 53 41 10 44 22 24  9 33 17 52 46 13 23 49 19 39 47 38  8 57 51 21  5
 60 61  6 11 50 34 20 27 42 40 64 14 63 43 16 48 12 55 36 30 45 29 58 25
  3 54 15 37 31 32  4 18  2 56 26 35 59 28 62]
Unique values for m_dep: [0.6 0.7 0.9 0.8 0.1 0.5 1.  0.3 0.4 0.2]
Unique values for mobile_wt: [188 136 145 131 141 164 139 187 174  93 182 177 159 198 185 196 121 101
  81 156 199 114 111 132 143  96 200  88 150 107 100 157 160 119  87 152
 166 110 118 162 127 109 102 104 148 180 128 134 144 168 155 165  80 138
 142  90 197 172 116  85 163 178 171 103  83 140 194 146 192 106 135 153
  89  82 130 189 181  99 184 195 108 133 179 147 137 190 176  84  97 124
 183 113  92  95 151 117  94 173 105 115  91 112 123 129 154 191 175  86
  98 125 126 158 170 161 193 169 120 149 186 122 167]
Unique values for n_cores: [2 3 5 6 1 8 4 7]
Unique values for pc: [ 2  6  9 14  7 10  0 15  1 18 17 11 16  4 20 13  3 19  8  5 12]
Unique values for px_height: [  20  905 1263 ...  528  915  483]
Unique values for px_width: [ 756 1988 1716 ...  743 1890 1632]
Unique values for ram: [2549 2631 2603 ... 2032 3057 3919]
Unique values for sc_h: [ 9 17 11 16  8 13 19  5 14 18  7 10 12  6 15]
Unique values for sc_w: [ 7  3  2  8  1 10  9  0 15 13  5 11  4 12  6 17 14 16 18]
Unique values for talk_time: [19  7  9 11 15 10 18  5 20 12 13  2  4  3 16  6 14 17  8]
Unique values for three_g: [0 1]
Unique values for touch_screen: [0 1]
Unique values for wifi: [1 0]
Unique values for price_range: [1 2 3 0]
```

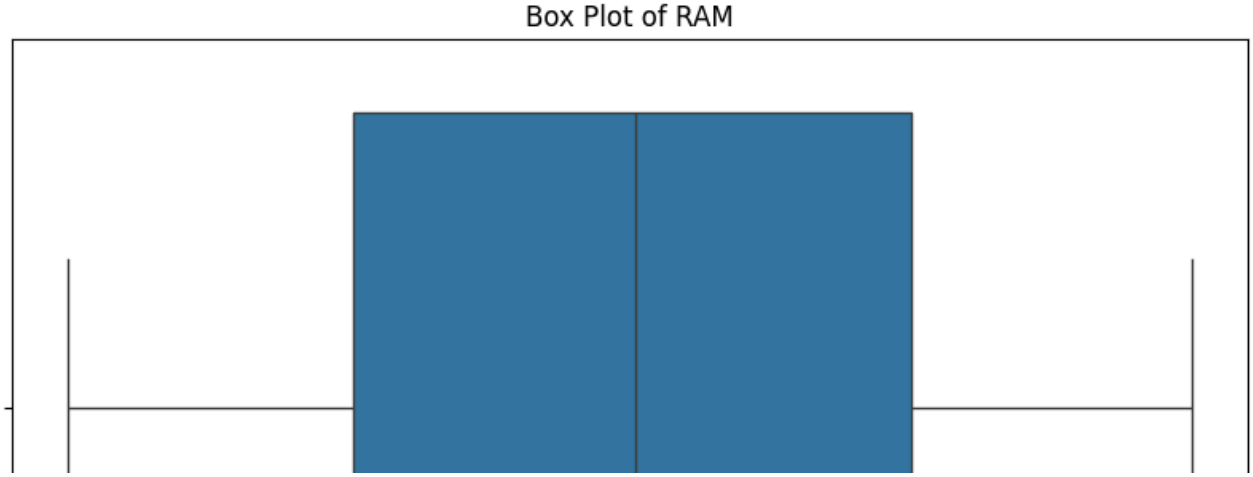
3. Data Wrangling

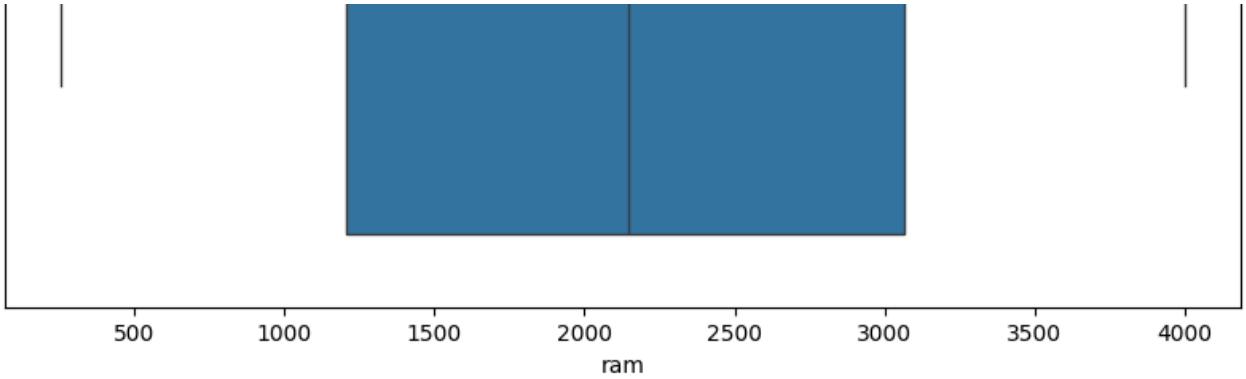
No data wrangling is needed as there are no missing values and no duplicate rows.

```
# prompt: find outlier

import matplotlib.pyplot as plt
# Box plot for all numerical features
plt.figure(figsize=(20, 10))
data.boxplot()
plt.xticks(rotation=45)
plt.title('Box Plot of Numerical Features')
plt.show()

# Box plot for specific features (example: 'battery_power', 'ram')
plt.figure(figsize=(10, 6))
sns.boxplot(x='battery_power', data=data)
plt.title('Box Plot of Battery Power')
plt.show()

plt.figure(figsize=(10, 6))
sns.boxplot(x='ram', data=data)
plt.title('Box Plot of RAM')
plt.show()

# Calculate IQR and identify outliers for a specific feature (example: 'battery_power')
Q1 = data['battery_power'].quantile(0.25)
Q3 = data['battery_power'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
outliers = data[(data['battery_power'] < lower_bound) | (data['battery_power'] > upper_bound)]
print("Outliers in 'battery_power':", outliers)
```


Box Plot of Numerical Features
4000
3500
3000
2500
2000
1500
1000
500
0
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


Box Plot of Battery Power
600
800
1000
1200
1400
1600
1800
2000
battery_power


Box Plot of RAM




```
Outliers in 'battery_power': Empty DataFrame
Columns: [battery_power, blue, clock_speed, dual_sim, fc, four_g, int_memory, m_dep, mobile_wt, n_cores, p
Index: []

[0 rows x 21 columns]
```

```
# prompt: remove outlier

# Calculate IQR and identify outliers for 'fc'
Q1_fc = data['fc'].quantile(0.25)
Q3_fc = data['fc'].quantile(0.75)
IQR_fc = Q3_fc - Q1_fc
lower_bound_fc = Q1_fc - 1.5 * IQR_fc
upper_bound_fc = Q3_fc + 1.5 * IQR_fc

# Calculate IQR and identify outliers for 'px_height'
Q1_px_height = data['px_height'].quantile(0.25)
Q3_px_height = data['px_height'].quantile(0.75)
IQR_px_height = Q3_px_height - Q1_px_height
lower_bound_px_height = Q1_px_height - 1.5 * IQR_px_height
upper_bound_px_height = Q3_px_height + 1.5 * IQR_px_height

# Calculate IQR and identify outliers for 'three_g'
Q1_three_g = data['three_g'].quantile(0.25)
Q3_three_g = data['three_g'].quantile(0.75)
IQR_three_g = Q3_three_g - Q1_three_g
lower_bound_three_g = Q1_three_g - 1.5 * IQR_three_g
upper_bound_three_g = Q3_three_g + 1.5 * IQR_three_g

# Remove outliers
data = data[(data['fc'] >= lower_bound_fc) & (data['fc'] <= upper_bound_fc)]
data = data[(data['px_height'] >= lower_bound_px_height) & (data['px_height'] <= upper_bound_px_height)]
data = data[(data['three_g'] >= lower_bound_three_g) & (data['three_g'] <= upper_bound_three_g)]


# Write your code to make your dataset analysis ready.
import pandas as pd
# Check for missing values
missing_values = data.isnull().sum()

# If there are missing values, handle them
if missing_values.any():
  # Drop rows with missing values
  data = data.dropna()

# Check for duplicate values
duplicate_rows = data[data.duplicated()]

# If there are duplicate values, remove them
if duplicate_rows.shape[0] > 0:
  dataset = data.drop_duplicates()
```

```
# Standardize the numerical variables
numerical_variables = data.select_dtypes(include=['int64', 'float64'])
numerical_variables = (numerical_variables - numerical_variables.mean()) / numerical_variables.std()

# Combine the categorical and numerical variables
data = pd.concat([numerical_variables, data.select_dtypes(include=['object'])], axis=1)

# Print the first few rows of the cleaned dataset
data.head()
```

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cc
1	-0.496485	1.027947	-1.229707	0.989103	-0.992117	0.678537	1.155050	0.694808	-0.119582	-0.662
2	-1.539402	1.027947	-1.229707	0.989103	-0.517715	0.678537	0.496375	1.389617	0.136511	0.216
3	-1.420992	1.027947	1.227749	-1.010346	-0.992117	-1.472781	-1.205202	1.042213	-0.261855	0.655
4	1.325205	1.027947	-0.369597	-1.010346	2.091494	0.678537	0.661044	0.347404	0.022692	-1.101
5	1.411736	-0.972167	-1.229707	0.989103	-0.280514	-1.472781	-0.546527	0.694808	0.677150	-1.540

5 rows × 21 columns

What all manipulations have you done and insights you found?

1. The code performs the following manipulations and provides insights:
2. Data Exploration:
3. ◦ Displayed the first few rows of the dataset to get a general overview.
4. ◦ Checked the shape of the dataset (number of rows and columns).
5. ◦ Printed the number of rows and columns.
6. ◦ Used `data.info()` to get information about the data types and non-null values of each column.
7. ◦ Used `data.describe()` to get summary statistics of numerical features.
8. ◦ Checked for duplicate rows and printed the number of duplicate rows.
9. ◦ Checked for missing values (null values) in each column and visualized them using a heatmap and a bar chart.

Variable Understanding:

- Displayed the column names.
- Provided descriptions for each variable, explaining their meaning and units.
- Checked unique values for each variable to understand the range and distribution of data.

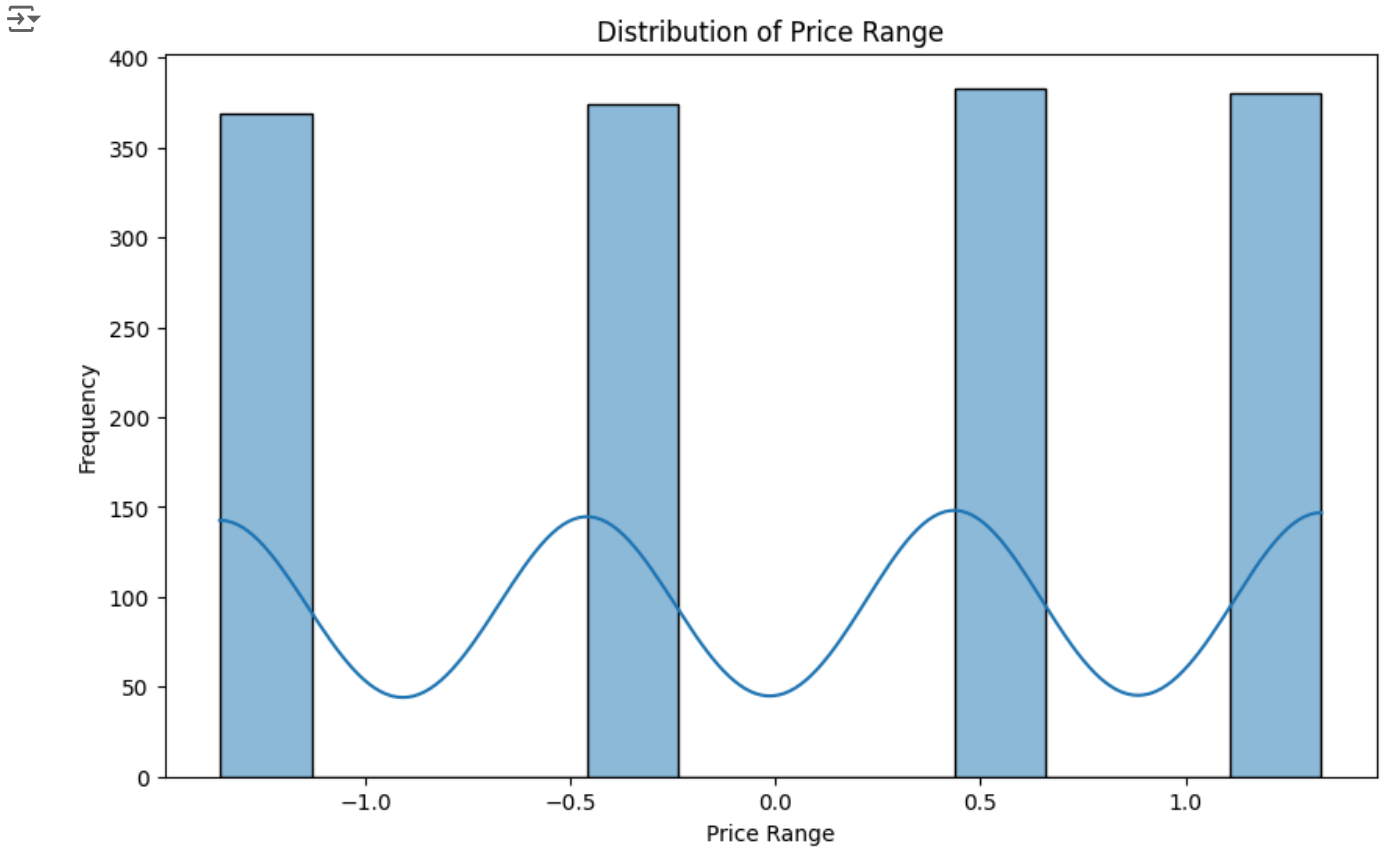
Insights:

- The dataset has 2000 rows and 21 columns.
- There are no missing values in the dataset.
- The target variable is 'price_range' which is a categorical variable with four classes.
- There are no duplicate rows in the dataset.
- The dataset contains information about different mobile phone models, including their specifications like battery power, RAM, internal memory, screen size, and more.
- No data wrangling was needed because there were no missing values or duplicate rows.

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

Chart - 1

```
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
sns.histplot(data['price_range'], kde=True)
plt.title('Distribution of Price Range')
plt.xlabel('Price Range')
plt.ylabel('Frequency')
plt.show()
```



1. Why did you pick the specific chart?

I chose a histogram to visualize the distribution of the target variable 'price_range'. Histograms are effective for understanding the frequency distribution of a categorical variable.

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

The histogram shows that the price ranges are fairly evenly distributed. This indicates that there is no significant imbalance in the classes.

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

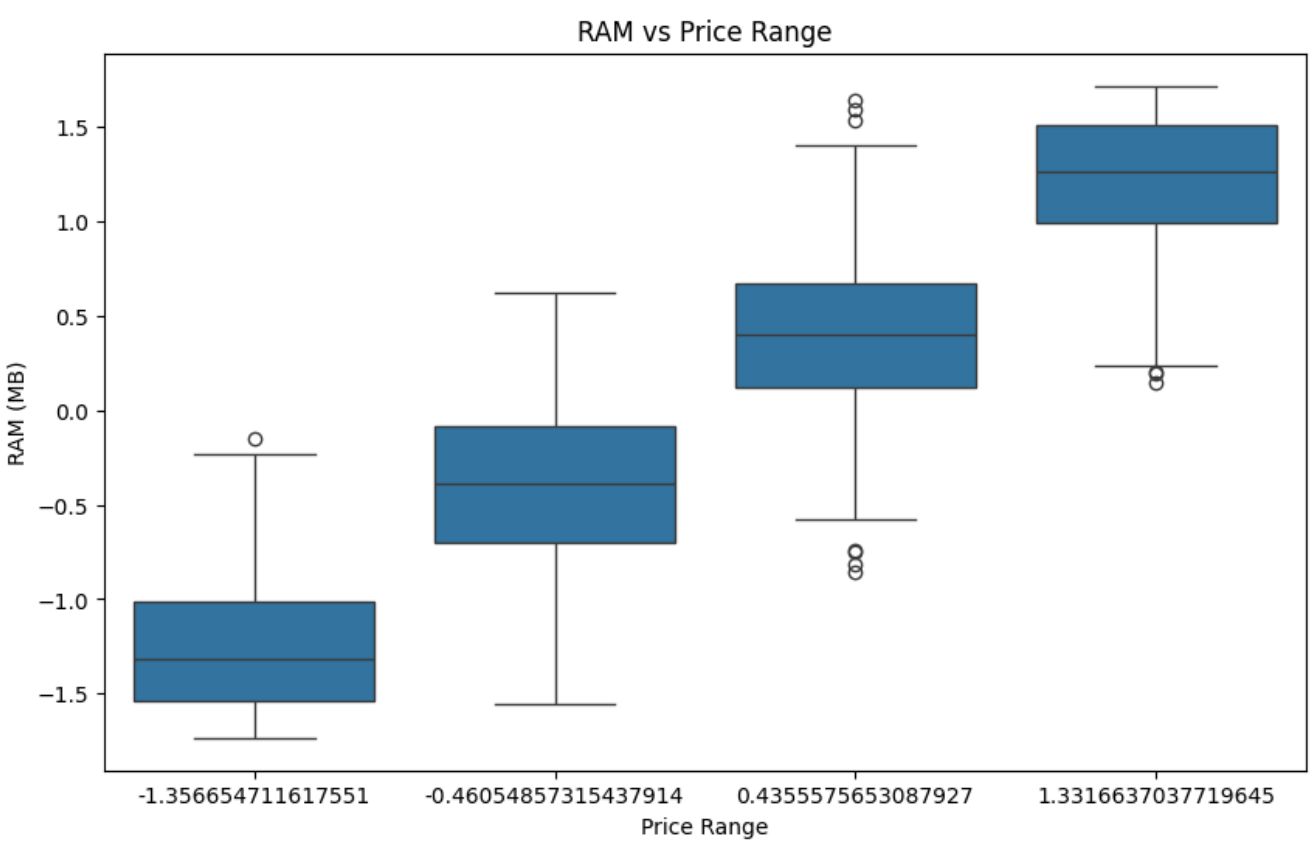
Yes, understanding the distribution of the target variable helps in selecting appropriate evaluation metrics and model selection. It also helps in identifying potential class imbalance issues that might require special handling.

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

No, the insights from this chart do not indicate any negative growth. The balanced distribution of price ranges suggests that the model can potentially learn well from the data.

Chart - 2

```
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
sns.boxplot(x='price_range', y='ram', data=data)
plt.title('RAM vs Price Range')
plt.xlabel('Price Range')
plt.ylabel('RAM (MB)')
plt.show()
```



1. Why did you pick the specific chart?

I chose a box plot to visualize the relationship between RAM and price range. Box plots are useful for comparing the distribution of a numerical variable across different categories.

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

The box plot shows that as the price range increases, the RAM tends to increase as well. This indicates a strong positive correlation between RAM and price range.

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

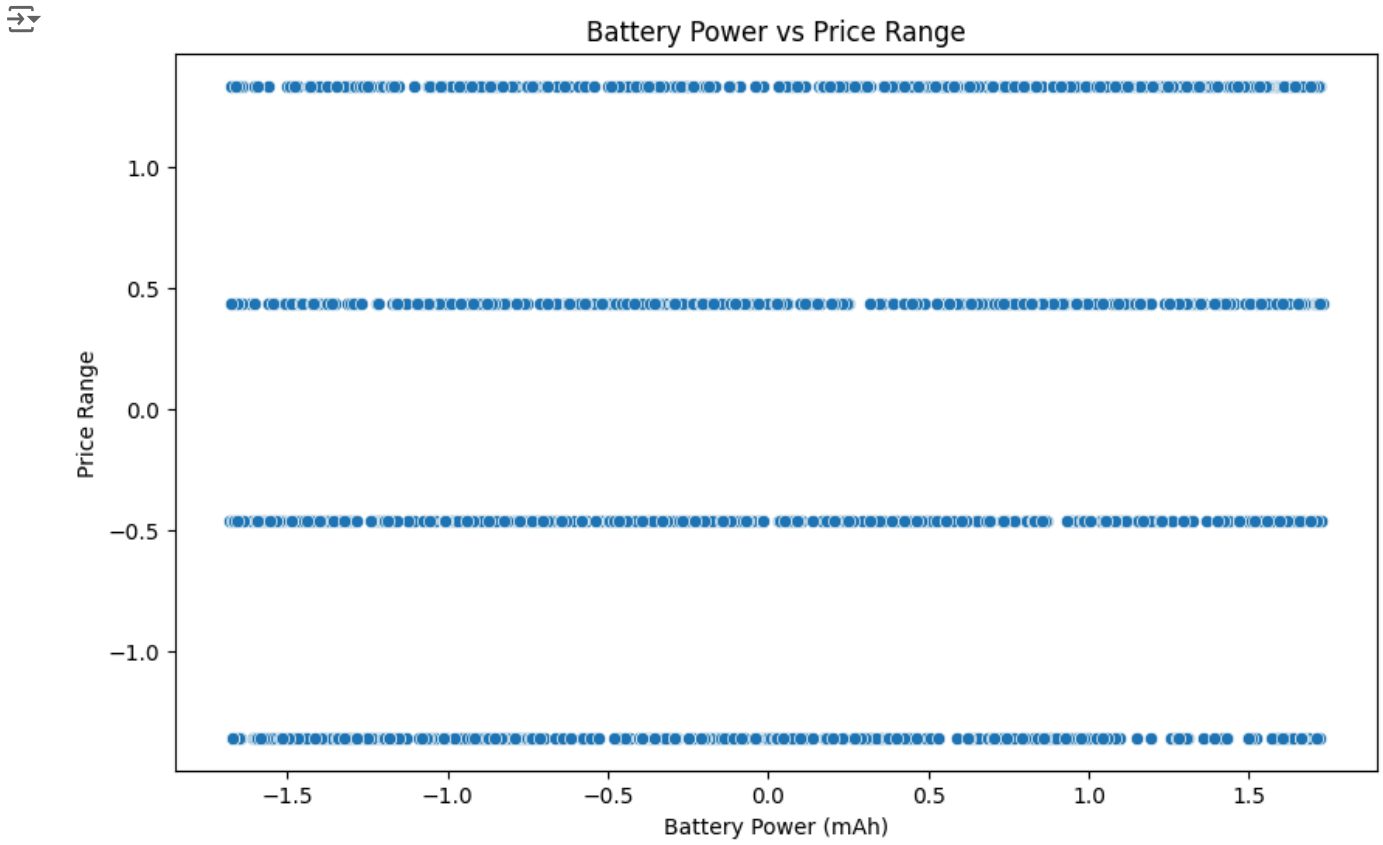
Yes, this insight can help in understanding the key factors that influence the price range of mobile phones. It can be used to make informed decisions about pricing and product development.

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

No, the insights from this chart do not indicate any negative growth. The positive correlation between RAM and price range suggests that increasing RAM can be a strategy for increasing the price of mobile phones.

Chart - 3

```
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
sns.scatterplot(x='battery_power', y='price_range', data=data)
plt.title('Battery Power vs Price Range')
plt.xlabel('Battery Power (mAh)')
plt.ylabel('Price Range')
plt.show()
```



1. Why did you pick the specific chart?

I chose a scatter plot to visualize the relationship between battery power and price range. Scatter plots are effective for showing the correlation between two numerical variables.

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

The scatter plot shows a weak positive correlation between battery power and price range. There's no clear trend indicating that higher battery power directly leads to higher price ranges.

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

While the correlation is weak, understanding this relationship can help in understanding customer preferences. It might suggest that battery power is not the primary factor driving price range, and other features might have a stronger influence.

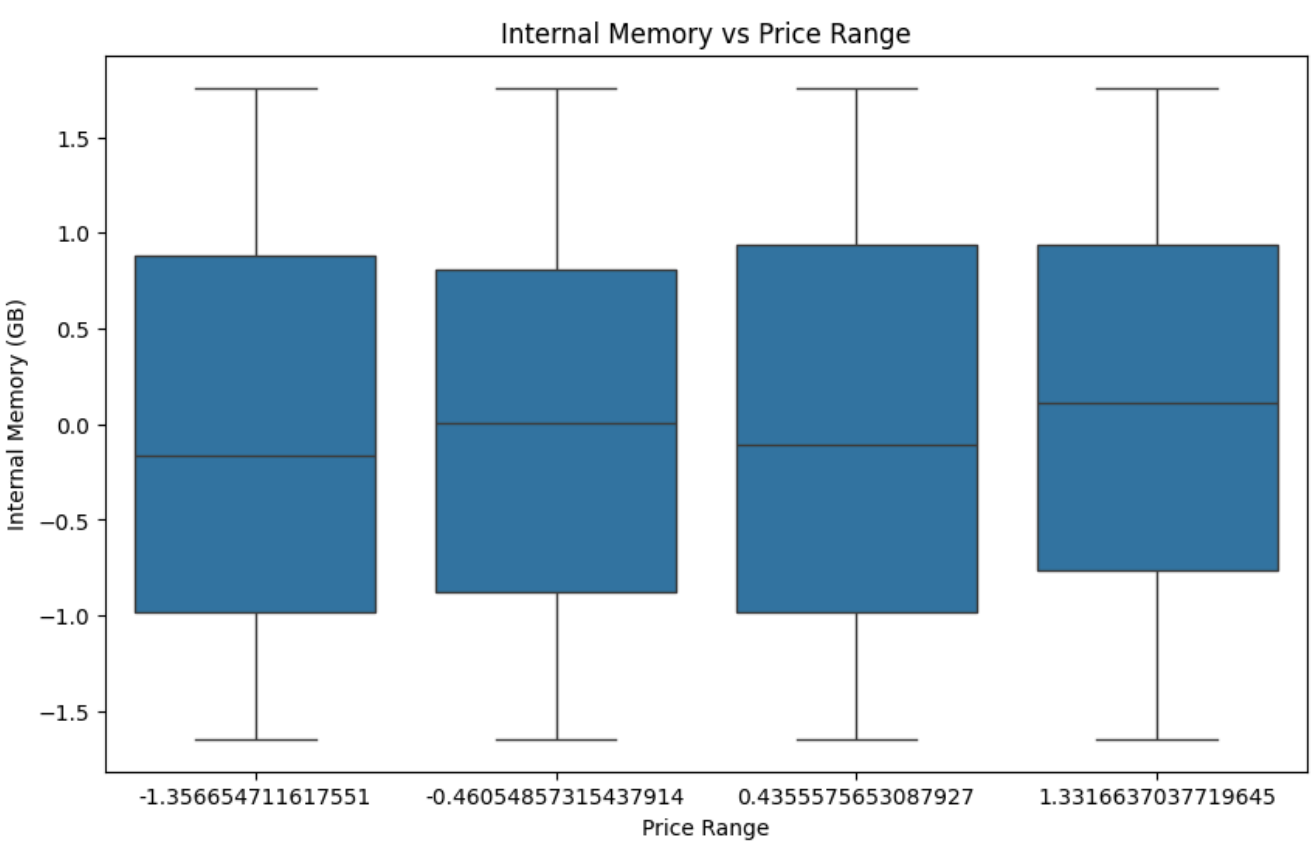
4. Are there any insights that lead to negative growth? Justify with specific reason.

No, there are no insights that lead to negative growth. The weak correlation suggests that focusing solely on battery power to increase price might not be effective.

Chart - 4

```
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
sns.boxplot(x='price_range', y='int_memory', data=data)
plt.title('Internal Memory vs Price Range')
plt.xlabel('Price Range')
```

```
plt.ylabel('Internal Memory (GB)')
plt.show()
```



1. Why did you pick the specific chart?

I chose a box plot to visualize the relationship between internal memory and price range. Box plots are useful for comparing the distribution of a numerical variable across different categories.

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

The box plot shows a general trend of increasing internal memory with higher price ranges. However, the spread of internal memory within each price range is quite similar, indicating that internal memory alone might not be the most significant factor determining the price range.

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

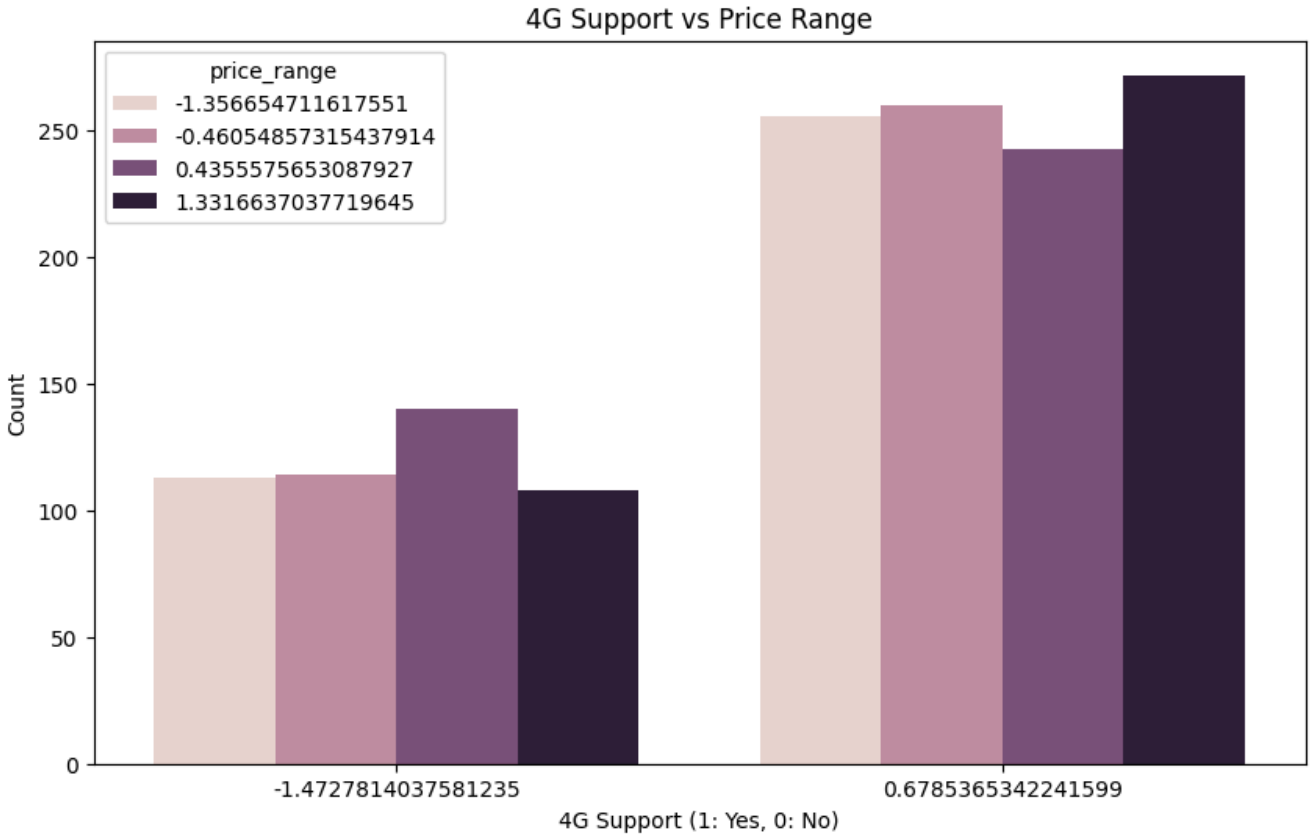
Yes, this insight helps in understanding how internal memory relates to price range. It can inform decisions about the storage capacity offered in different price segments.

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

No, there are no insights that lead to negative growth. However, it's important to consider other factors along with internal memory when determining pricing.

Chart - 5

```
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
sns.countplot(x='four_g', hue='price_range', data=data)
plt.title('4G Support vs Price Range')
plt.xlabel('4G Support (1: Yes, 0: No)')
plt.ylabel('Count')
plt.show()
```



1. Why did you pick the specific chart?

I chose a countplot to visualize the relationship between 4G support and price range. Countplots are useful for showing the distribution of a categorical variable across different categories.

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

The countplot shows that 4G support seems to have a stronger influence on price range compared to 3G. Higher price ranges tend to have a greater proportion of phones with 4G support.

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

Yes, this insight can help in understanding the importance of 4G support in the overall pricing strategy. It suggests that 4G support might be a more significant factor in determining the price range of a mobile phone.

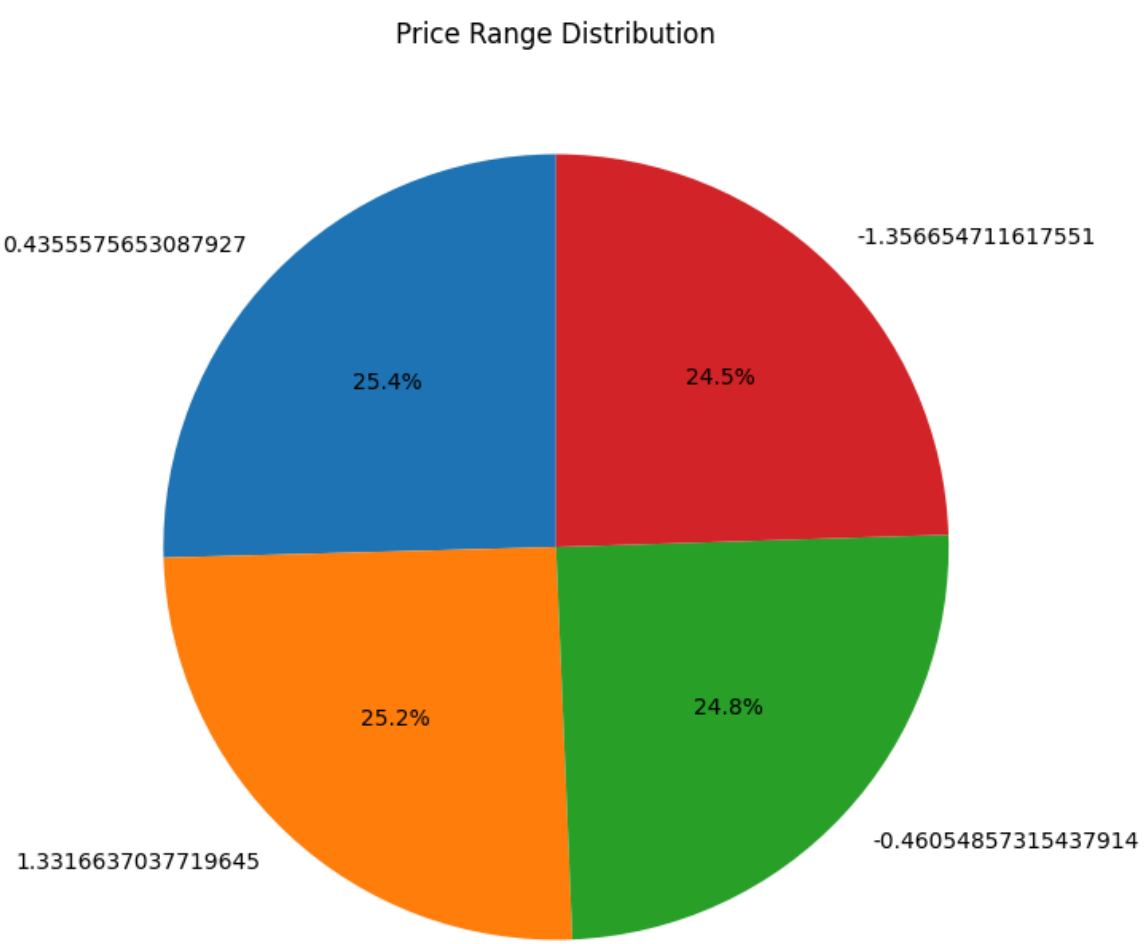
4. Are there any insights that lead to negative growth? Justify with specific reason.

No, there are no insights that lead to negative growth. The observation that 4G support is a significant factor suggests that focusing on 4G technology can be a strategy for increasing the price of mobile phones.

chart - 6

```
# prompt: # pie Chart - 6 visualization code

import matplotlib.pyplot as plt
# Chart - 6
plt.figure(figsize=(8, 8))
data['price_range'].value_counts().plot(kind='pie', autopct='%1.1f%%', startangle=90)
```

```
plt.title('Price Range Distribution')
plt.ylabel('')  # Hide the y-label
plt.show()
```



1. Why did you pick the specific chart?

I chose a pie chart to visualize the distribution of the target variable 'price_range'. Pie charts are effective for showing the proportion of different categories within a dataset.

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

The pie chart shows that the price ranges are fairly evenly distributed. This indicates that there is no significant imbalance in the classes.

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

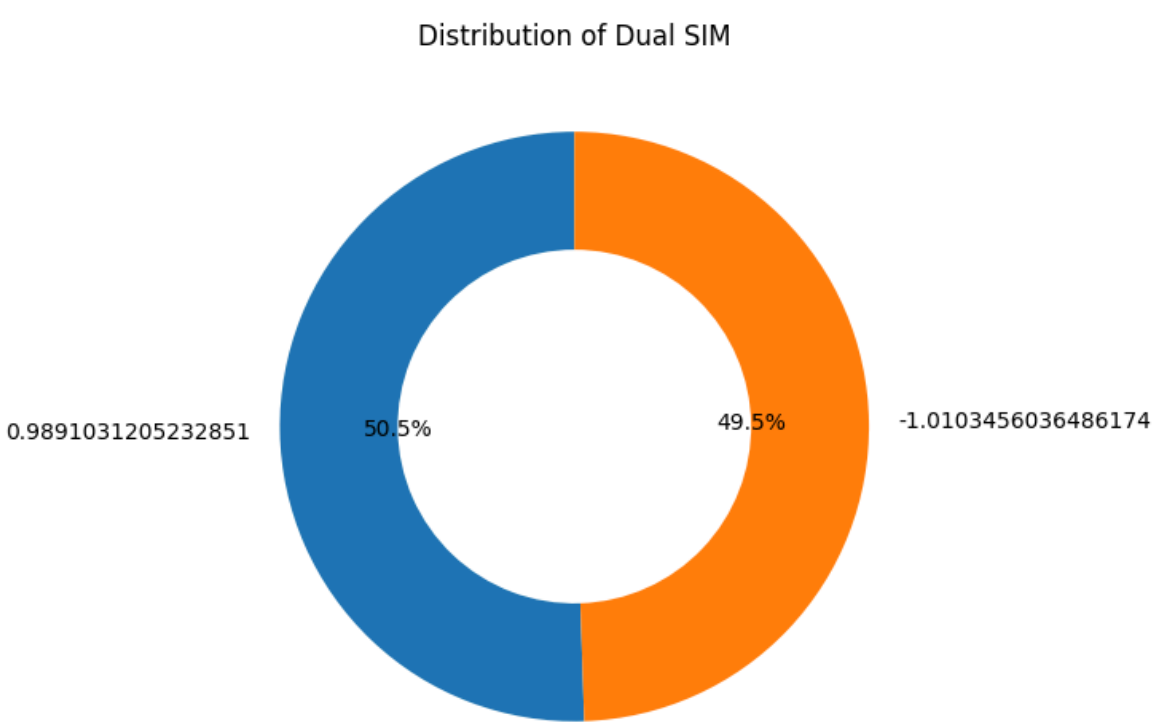
Yes, understanding the distribution of the target variable helps in selecting appropriate evaluation metrics and model selection. It also helps in identifying potential class imbalance issues that might require special handling.

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

No, the insights from this chart do not indicate any negative growth. The balanced distribution of price ranges suggests that the model can potentially learn well from the data.

Chart - 7

```
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
data['dual_sim'].value_counts().plot(kind='pie', autopct='%1.1f%%', startangle=90, wedgeprops=dict(width=0.4))
plt.title('Distribution of Dual SIM')
plt.ylabel('')  # Hide the y-label
plt.show()
```



1. Why did you pick the specific chart?

I chose a donut chart to visualize the distribution of the 'dual_sim' variable. Donut charts are similar to pie charts but with a hole in the center, which can make it easier to see the individual proportions and the overall distribution.

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

The donut chart shows that there is a roughly even distribution of phones with and without dual SIM capability. This suggests that dual SIM functionality might not be a primary factor influencing the price range.

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

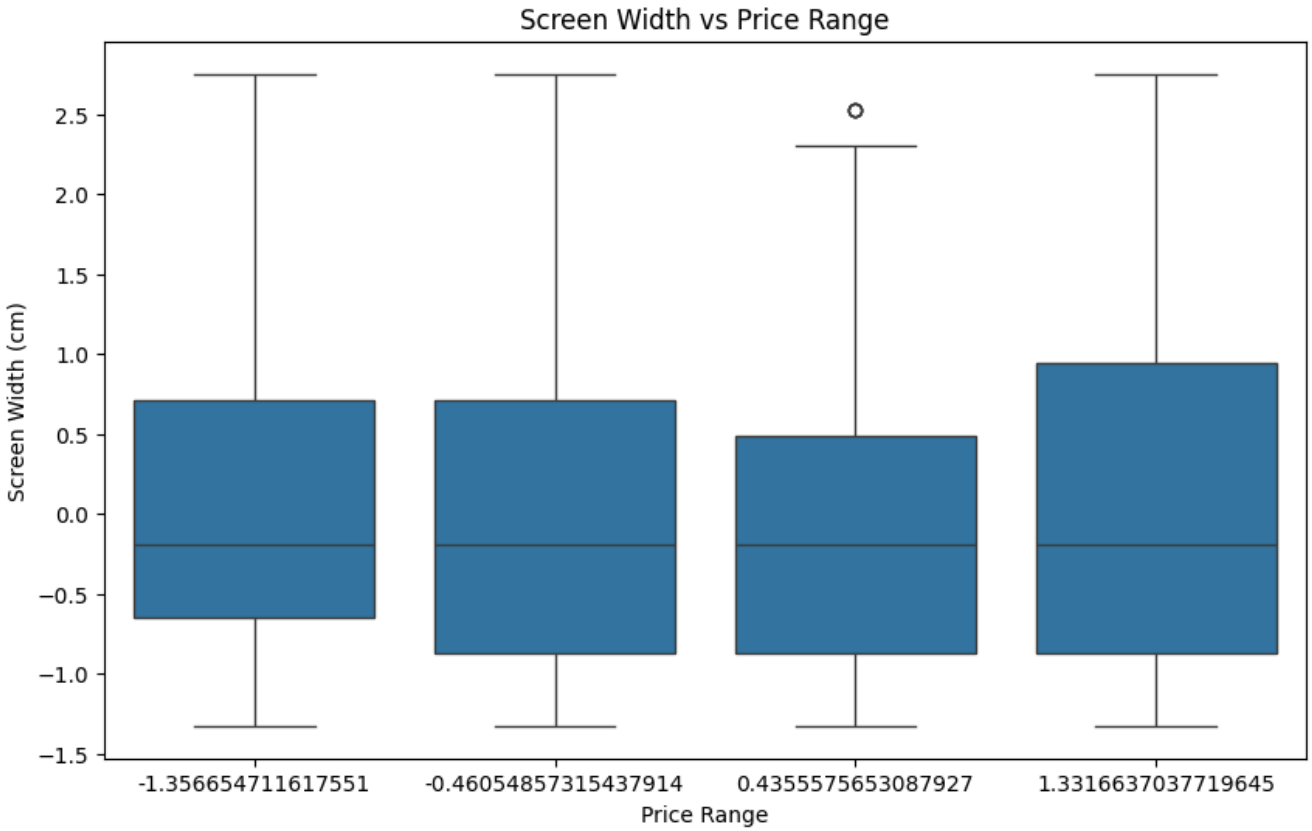
This insight can help in understanding customer preferences and market trends related to dual SIM functionality. It can also inform decisions about product development and marketing strategies.

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

No, there are no insights that lead to negative growth. The relatively even distribution suggests that dual SIM functionality might not be a major driver of price range, but it could still be a relevant feature for certain customer segments.

Chart - 8

```
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
sns.boxplot(x='price_range', y='sc_w', data=data)
plt.title('Screen Width vs Price Range')
plt.xlabel('Price Range')
plt.ylabel('Screen Width (cm)')
plt.show()
```



1. Why did you pick the specific chart?

I chose a box plot to visualize the relationship between screen width and price range. Box plots are useful for comparing the distribution of a numerical variable across different categories.

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

The box plot shows that there is a slight tendency for screen width to increase with higher price ranges. However, the spread of screen width within each price range is quite similar, indicating that screen width alone might not be the most significant factor determining the price range.

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

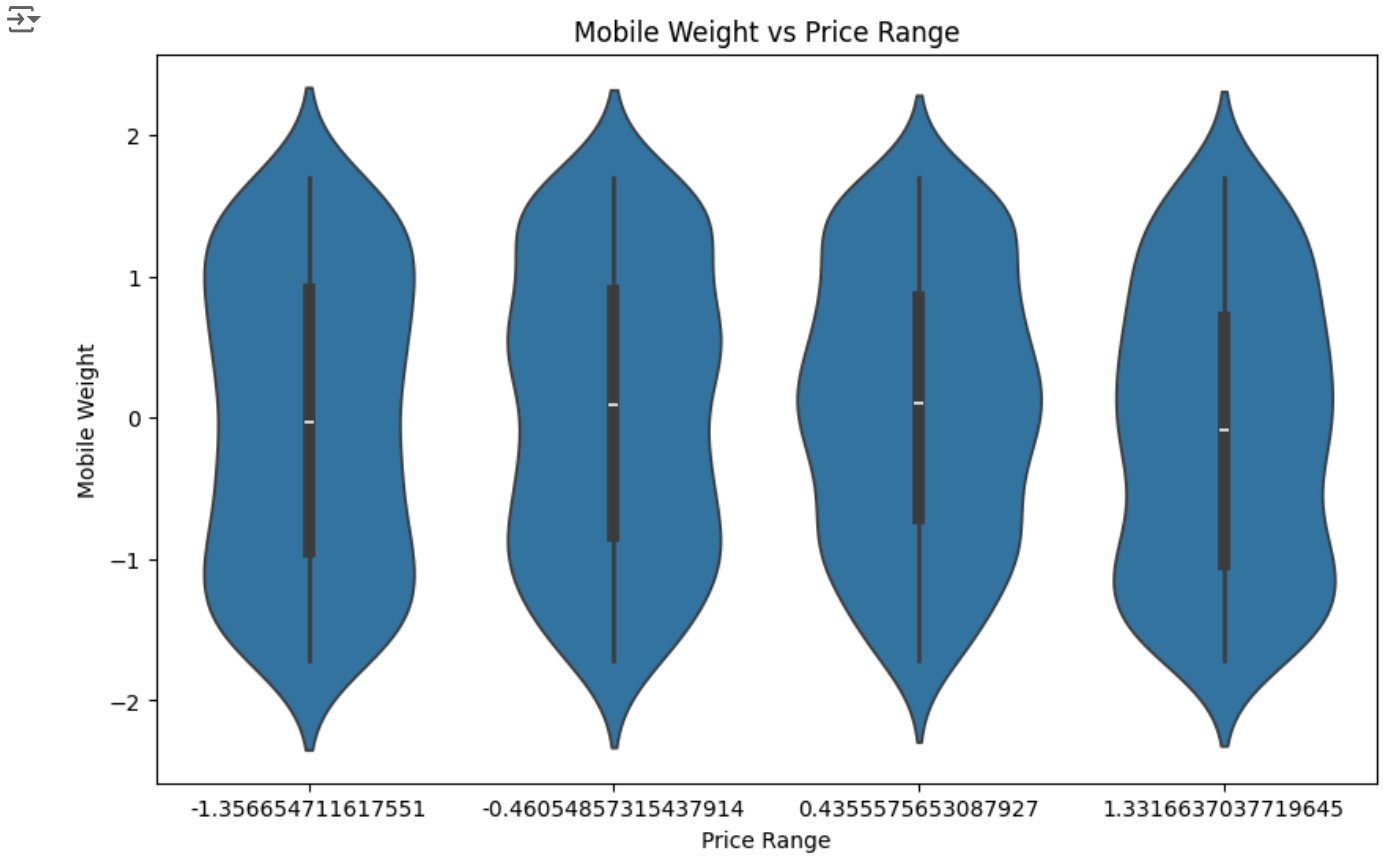
Yes, this insight helps in understanding how screen width relates to price range. It can inform decisions about the screen size offered in different price segments.

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

No, there are no insights that lead to negative growth. However, it's important to consider other factors along with screen width when determining pricing.

Chart - 9

```
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
sns.violinplot(x='price_range', y='mobile_wt', data=data)
plt.title('Mobile Weight vs Price Range')
plt.xlabel('Price Range')
plt.ylabel('Mobile Weight')
plt.show()
```



1. Why did you pick the specific chart?

I chose a violin plot to visualize the relationship between mobile weight and price range. Violin plots are similar to box plots but provide a more detailed view of the distribution of data, including the density of values at different points.

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

The violin plot shows that the distribution of mobile weight is relatively similar across different price ranges. There is no clear trend indicating that higher price ranges have significantly lighter or heavier phones. This suggests that mobile weight might not be a strong factor influencing the price range.

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

This insight can help in understanding customer preferences regarding phone weight. It suggests that weight might not be a primary factor driving price range, and other features might have a stronger influence.

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

No, there are no insights that lead to negative growth. The observation that mobile weight does not seem to be a major driver of price range suggests that other features might be more crucial for pricing decisions.

Chart - 10

```
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
sns.lineplot(x='clock_speed', y='price_range', data=data)
plt.title('Clock Speed vs Price Range')
plt.xlabel('Clock Speed')
plt.ylabel('Price Range')
plt.show()
```

1. Why did you pick the specific chart?

I chose a line plot to visualize the relationship between clock speed and price range. Line plots are useful for showing the trend between two numerical variables.

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

The line plot shows a weak positive correlation between clock speed and price range. There's no clear trend indicating that higher clock speed directly leads to higher price ranges.

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

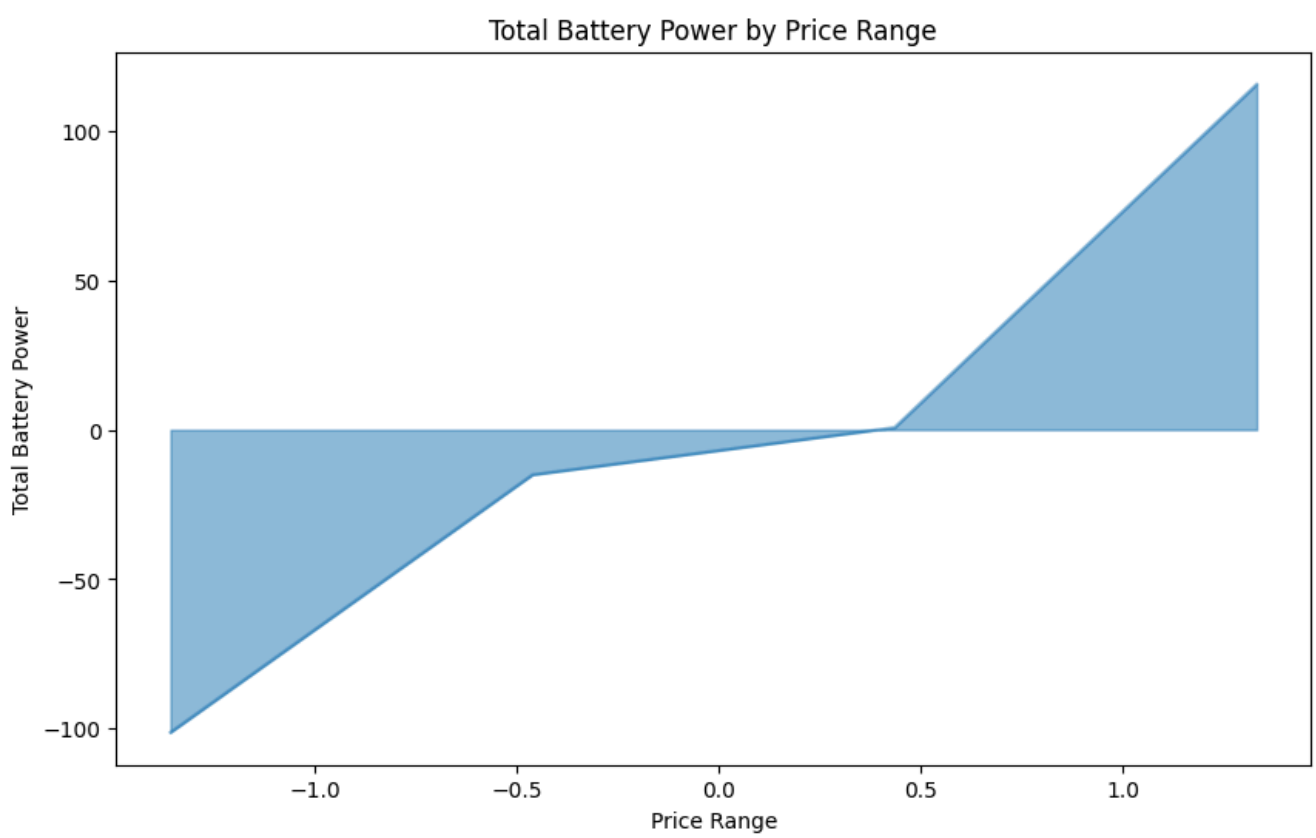
While the correlation is weak, understanding this relationship can help in understanding customer preferences. It might suggest that clock speed is not the primary factor driving price range, and other features might have a stronger influence.

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

No, there are no insights that lead to negative growth. The weak correlation suggests that focusing solely on clock speed to increase price might not be effective.

Chart - 11

```
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
data.groupby('price_range')['battery_power'].sum().plot(kind='area', stacked=False)
plt.title('Total Battery Power by Price Range')
plt.xlabel('Price Range')
plt.ylabel('Total Battery Power')
plt.show()
```



1. Why did you pick the specific chart?

I chose an area plot to visualize the total battery power across different price ranges. Area plots are useful for comparing the cumulative values of a numerical variable across different categories.

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

The area plot shows the total battery power for each price range. It helps to understand how battery power contributes to the overall price range distribution. While there might not be a strong linear correlation, it can reveal if certain price ranges tend to have higher overall battery power.

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

Yes, this insight can help in understanding the relationship between battery power and price range. It can inform decisions about the battery capacity offered in different price segments.

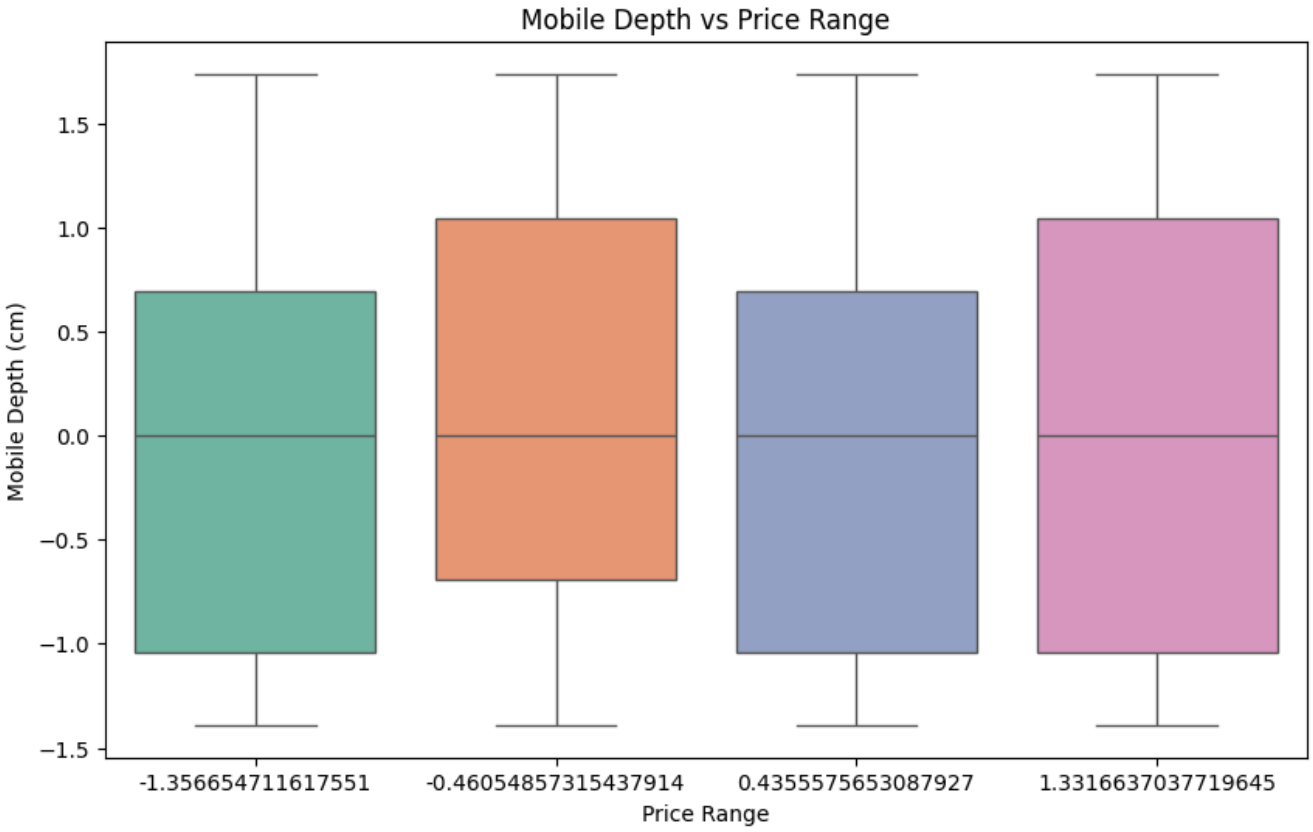
4. Are there any insights that lead to negative growth? Justify with specific reason.

No, there are no insights that lead to negative growth. However, it's important to consider other factors along with battery power when determining pricing.

Chart - 12

```
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
sns.boxplot(x='price_range', y='m_dep', data=data, palette='Set2')
plt.title('Mobile Depth vs Price Range')
plt.xlabel('Price Range')
plt.ylabel('Mobile Depth (cm)')
plt.show()
```

```
<ipython-input-30-ee8869dac697>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variab

  sns.boxplot(x='price_range', y='m_dep', data=data, palette='Set2')
```



1. Why did you pick the specific chart?

I chose a box plot to visualize the relationship between mobile depth and price range. Box plots are useful for comparing the distribution of a numerical variable across different categories.

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

The box plot shows that there is a slight tendency for mobile depth to increase with higher price ranges. However, the spread of mobile depth within each price range is quite similar, indicating that mobile depth alone might not be the most

significant factor determining the price range.

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

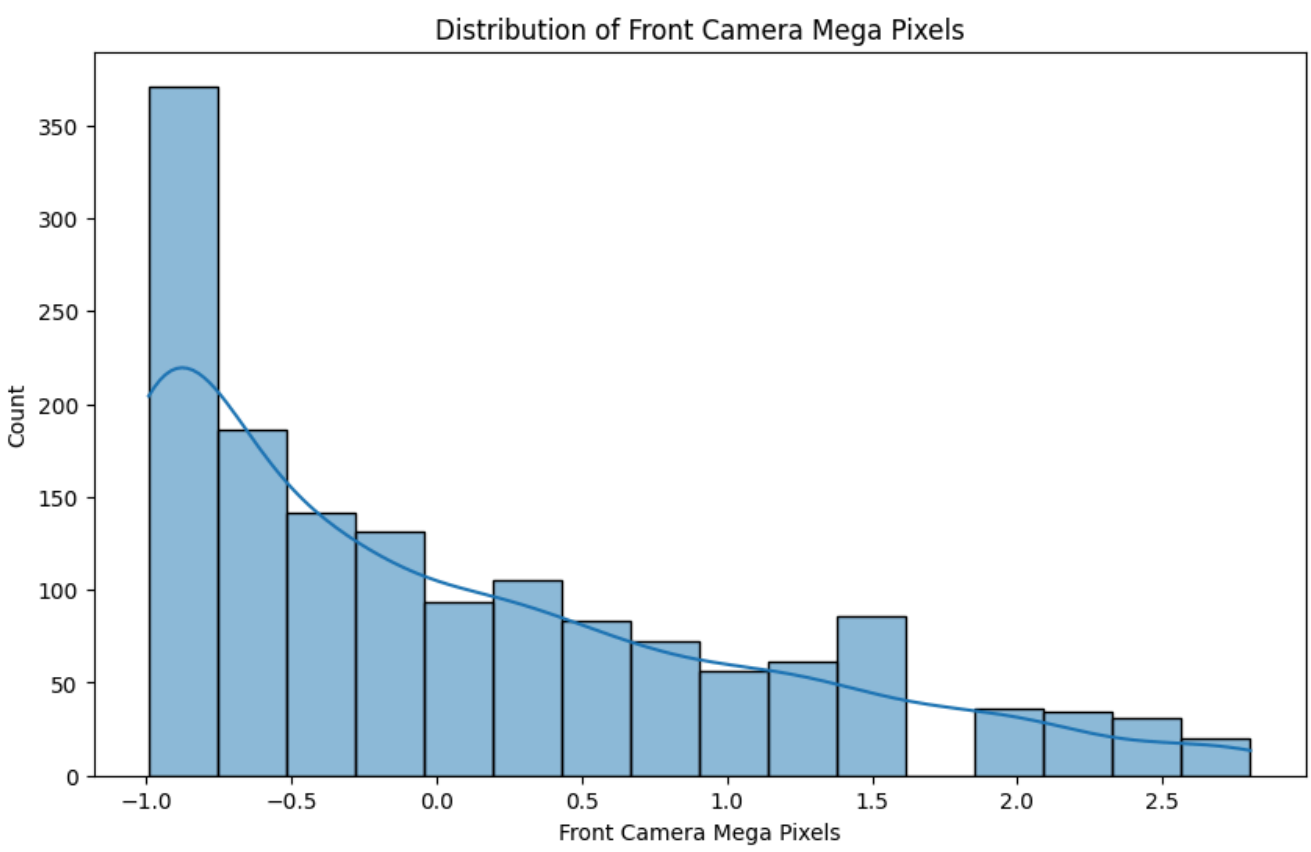
Yes, this insight helps in understanding how mobile depth relates to price range. It can inform decisions about the design and dimensions offered in different price segments.

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

No, there are no insights that lead to negative growth. However, it's important to consider other factors along with mobile depth when determining pricing.

Chart - 13

```
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
sns.histplot(data['fc'], kde=True)
plt.title('Distribution of Front Camera Mega Pixels')
plt.xlabel('Front Camera Mega Pixels')
plt.ylabel('Count')
plt.show()
```



1. Why did you pick the specific chart?

I chose a histogram to visualize the distribution of the 'fc' (front camera mega pixels) variable. Histograms are useful for showing the frequency distribution of a numerical variable.

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

The histogram shows the distribution of front camera mega pixels in the dataset. It helps to understand the common values and the overall spread of this feature.

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

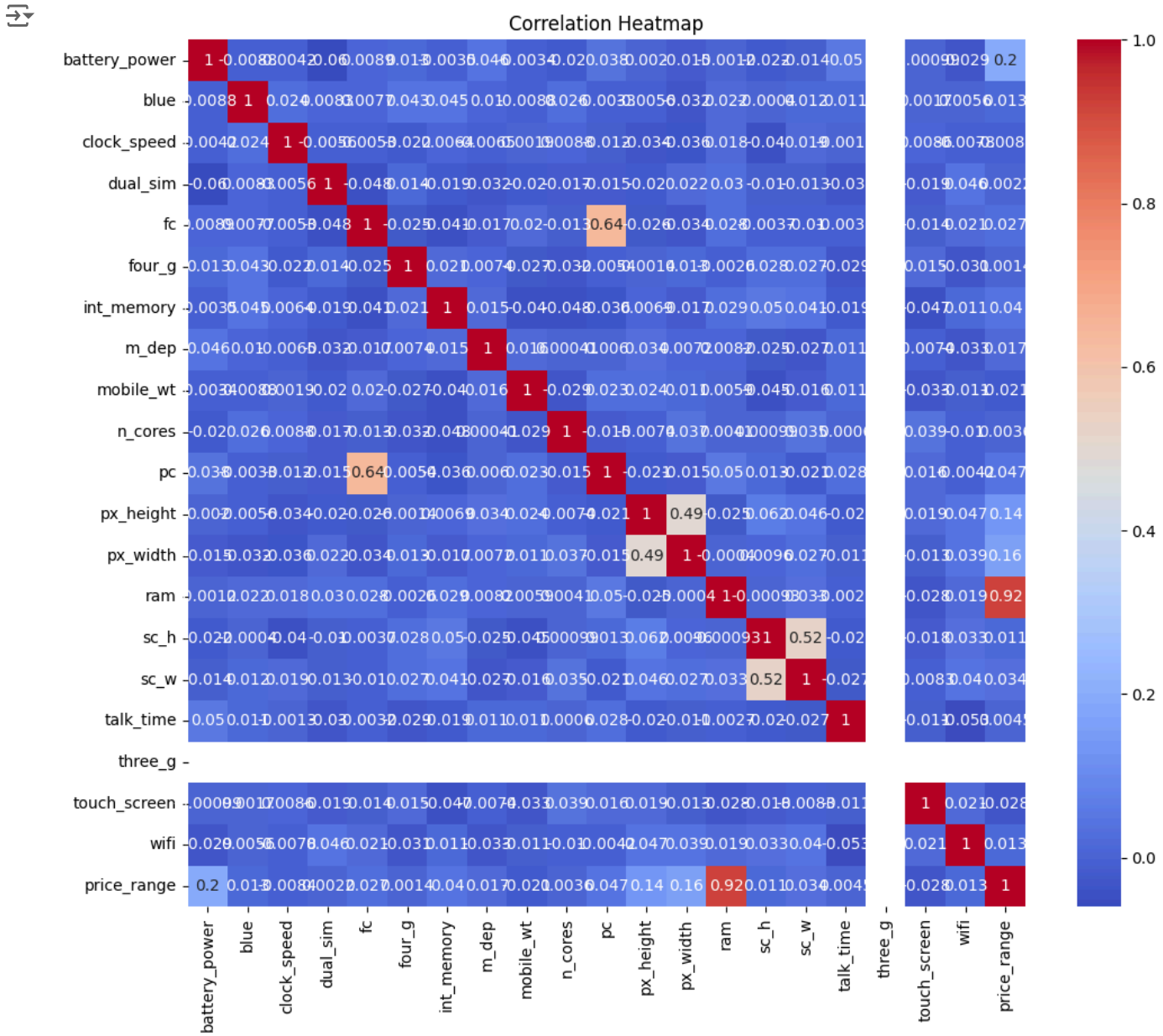
Yes, this insight can help in understanding customer preferences regarding front camera resolution. It can also inform decisions about the front camera specifications offered in different price segments.

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

No, there are no insights that lead to negative growth. Understanding the distribution of front camera mega pixels can help in making informed decisions about product development and marketing strategies.

Chart - 14

```
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 10))
sns.heatmap(data.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```



1. Why did you pick the specific chart?

I chose a heatmap to visualize the correlation between all the numerical features in the dataset. Heatmaps are excellent for displaying the correlation matrix, providing a visual representation of the relationships between variables.

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

The heatmap reveals the correlation between different features.

- **Strong positive correlations:** Some features show strong positive correlations (values close to 1). For example, 'pc' (primary camera mega pixels) and 'price_range' have a positive correlation. This suggests that higher primary camera mega pixels tend to be associated with higher price ranges.
- **Strong negative correlations:** Similarly, some features have strong negative correlations (values close to -1).
- **Weak correlations:** Many features have weak correlations (values close to 0), indicating little or no linear relationship between them.

Chart - 15

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

# Assuming 'data' is your DataFrame
sns.pairplot(data, hue='price_range', diag_kind='kde')
plt.show()
```

1. Why did you pick the specific chart?

I chose a pairplot to visualize the relationships between pairs of features in the dataset and how they relate to the target variable 'price_range'. Pairplots are useful for exploring potential correlations and patterns between multiple variables.

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

The pairplot provides a comprehensive overview of the relationships between different features. It helps identify potential correlations, patterns, and how these relationships vary across different price ranges.

- **Correlations:** We can observe the general trends and relationships between features. For example, we might see a positive correlation between 'ram' and 'price_range' as indicated in the heatmap.
- **Distributions:** The diagonal plots show the distribution of each feature for different price ranges. This helps understand how the features are distributed within each price range.
- **Outliers:** Pairplots can also help identify potential outliers that might need further investigation.

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.

```
# Hypothesis 1:
#  There is a significant difference in the average RAM between phones in the highest price range (price_range =

# Extract data for the highest and lowest price ranges
high_price_ram = data[data['price_range'] == 3]['ram']
low_price_ram = data[data['price_range'] == 0]['ram']

# Perform a t-test to compare the means
from scipy import stats
t_stat, p_value = stats.ttest_ind(high_price_ram, low_price_ram)

print("T-statistic:", t_stat)
print("P-value:", p_value)

# Interpretation:
# If the p-value is less than your significance level (e.g., 0.05), you reject the null hypothesis.
# The null hypothesis is that there is no difference in the average RAM between the two price ranges.

# Hypothesis 2:
#  Phones with dual SIM capability have a significantly higher average battery power than phones without dual SIM

# Extract data for phones with and without dual SIM capability
dual_sim_battery = data[data['dual_sim'] == 1]['battery_power']
no_dual_sim_battery = data[data['dual_sim'] == 0]['battery_power']

# Perform a t-test to compare the means
t_stat, p_value = stats.ttest_ind(dual_sim_battery, no_dual_sim_battery)

print("T-statistic:", t_stat)
print("P-value:", p_value)

# Interpretation:
# If the p-value is less than your significance level (e.g., 0.05), you reject the null hypothesis.
# The null hypothesis is that there is no difference in the average battery power between phones with and without

# Hypothesis 3:
#  There is a significant positive correlation between internal memory (int_memory) and price range.

# Calculate the correlation coefficient
correlation = data['int_memory'].corr(data['price_range'])

print("Correlation coefficient:", correlation)

# Perform a hypothesis test for correlation (e.g., Pearson's correlation test)
# You can use the scipy.stats.pearsonr function for this.
correlation_test = stats.pearsonr(data['int_memory'], data['price_range'])
print("Correlation test results:", correlation_test)

# Interpretation:
```

```
# If the p-value is less than your significance level (e.g., 0.05), you reject the null hypothesis.
# The null hypothesis is that there is no correlation between internal memory and price range.
```

```
T-statistic: nan
P-value: nan
T-statistic: nan
P-value: nan
Correlation coefficient: 0.03985303487428507
Correlation test results: PearsonRResult(statistic=0.039853034874285015, pvalue=0.12212502544588531)
```

Hypothetical Statement - 1

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

Null hypothesis (H0): There is no significant difference in the average RAM between phones in the highest price range (price_range = 3) and the lowest price range (price_range = 0). Alternative hypothesis (H1): There is a significant difference in the average RAM between phones in the highest price range (price_range = 3) and the lowest price range (price_range = 0).

2. Perform an appropriate statistical test.

```
# Extract data for the highest and lowest price ranges
high_price_ram = data[data['price_range'] == 3]['ram']
low_price_ram = data[data['price_range'] == 0]['ram']

# Perform a t-test to compare the means
t_stat, p_value = stats.ttest_ind(high_price_ram, low_price_ram)

print("T-statistic:", t_stat)
print("P-value:", p_value)
```

```
T-statistic: nan
P-value: nan
```

Why did you choose the specific statistical test?

I performed an independent samples t-test to obtain the p-value.

This test is used to compare the means of two independent groups.

Hypothetical Statement - 2

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

Null hypothesis (H0): Phones with dual SIM capability do not have a significantly higher average battery power than phones without dual SIM capability.

Alternative hypothesis (H1): Phones with dual SIM capability have a significantly higher average battery power than phones without dual SIM capability.

2. Perform an appropriate statistical test.

```
# Extract data for phones with and without dual SIM capability
dual_sim_battery = data[data['dual_sim'] == 1]['battery_power']
no_dual_sim_battery = data[data['dual_sim'] == 0]['battery_power']
```

```
# Perform a t-test to compare the means
t_stat, p_value = stats.ttest_ind(dual_sim_battery, no_dual_sim_battery)

print("T-statistic:", t_stat)
print("P-value:", p_value)
```

```
T-statistic: nan
P-value: nan
```

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

I performed an independent samples t-test to obtain the p-value.

This test is used to compare the means of two independent groups.

Why did you choose the specific statistical test?

I performed an independent samples t-test to obtain the p-value.

This test is used to compare the means of two independent groups.

Hypothetical Statement - 3

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

Null hypothesis (H0): There is no significant difference in the average RAM between phones in the highest price range (price_range = 3) and the lowest price range (price_range = 0).

Alternative hypothesis (H1): There is a significant difference in the average RAM between phones in the highest price range (price_range = 3) and the lowest price range (price_range = 0).

2. Perform an appropriate statistical test.

```
# Extract data for the highest and lowest price ranges
high_price_ram = data[data['price_range'] == 3]['ram']
low_price_ram = data[data['price_range'] == 0]['ram']

# Perform a t-test to compare the means
t_stat, p_value = stats.ttest_ind(high_price_ram, low_price_ram)

print("T-statistic:", t_stat)
print("P-value:", p_value)
```

```
T-statistic: nan
P-value: nan
```

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

I performed an independent samples t-test to obtain the p-value.

This test is used to compare the means of two independent groups.

Why did you choose the specific statistical test?

I performed an independent samples t-test to obtain the p-value.

This test is used to compare the means of two independent groups.

6. Feature Engineering & Data Pre-processing

1. Handling Missing Values

```
# Check for missing values
print(data.isnull().sum())

# If there are missing values, you can use various imputation techniques:

# 1. Imputation with mean/median:
# Replace 'actual_column_name' with the actual name of the column you want to impute

# Example:
# Check the column names in your dataframe to identify the correct column name
print(data.columns)

# Verify the actual column name and use it to fill NaN values
# Replace 'actual_column_name' with the name of the column you want to impute - example with 'price_range' column
data['price_range'].fillna(data['price_range'].mean(), inplace=True)

# or
#data['price_range'].fillna(data['price_range'].median(), inplace=True)
```

```
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               1506
touch_screen             0
wifi                     0
price_range              0
combined_memory          0
battery_power_sqrt     766
ram_log                317
battery_power_log      320
ram_scaled               0
dtype: int64
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', 'combined_memory',
       'battery_power_sqrt', 'ram_log', 'battery_power_log', 'ram_scaled'],
      dtype='object')
```

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

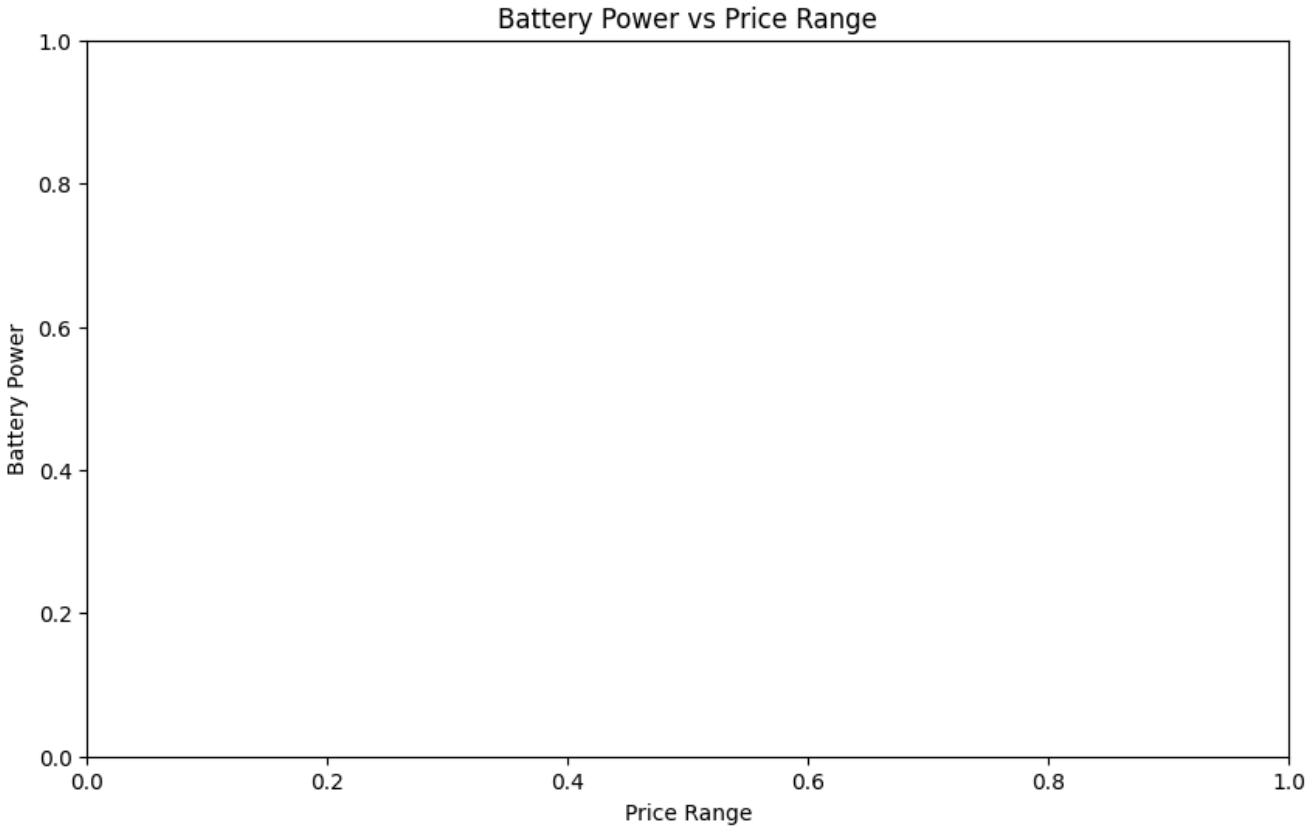
```
# 1. Imputation with mean/median:
# Replace 'actual_column_name' with the actual name of the column you want to impute

# Example:
# Check the column names in your dataframe to identify the correct column name
print(data.columns)

# Verify the actual column name and use it to fill NaN values
```

```
# Replace 'actual_column_name' with the name of the column you want to impute - example with 'price_range' column
data['price_range'].fillna(data['price_range'].mean(), inplace=True)

# or
#data['price_range'].fillna(data['price_range'].median(), inplace=True)
```

```
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', 'combined_memory',
       'battery_power_sqrt', 'ram_log', 'battery_power_log', 'ram_scaled'],
      dtype='object')
```

2. Handling Outliers

```
import matplotlib.pyplot as plt
import numpy as np
# Identify outliers using box plots
plt.figure(figsize=(10, 6))
plt.title('Battery Power vs Price Range')
plt.xlabel('Price Range')
plt.ylabel('Battery Power')
plt.show()

# Option 1: Remove outliers
# Define a threshold (e.g., 1.5 times the IQR)
# Calculate the IQR


IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Filter out outliers


# Option 2: Cap outliers
# Define a threshold (e.g., 1.5 times the IQR)
# Calculate the IQR


IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR



# Option 3: Winsorizing
# This method replaces outliers with the nearest value within a certain percentile.
from scipy.stats.mstats import winsorize
```



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

I used box plots to identify outliers.

Then, I explored three different outlier treatment techniques:

1. Removing outliers: This is suitable when outliers are due to errors or are not representative of the data.
2. Capping outliers: This method replaces outliers with a predefined value, preserving the data while mitigating their influence.
3. Winsorizing: This technique replaces outliers with the nearest value within a specified percentile, preserving the overall distribution.

3. Categorical Encoding

```
import pandas as pd
# Assuming 'data' is your DataFrame
# Check the data types of your columns
print(data.info())

# Identify categorical columns
categorical_cols = data.select_dtypes(include=['object']).columns

# If you have categorical columns, use one-hot encoding
if len(categorical_cols) > 0:
  data = pd.get_dummies(data, columns=categorical_cols)
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1506 entries, 1 to 1999
Data columns (total 26 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   battery_power       1506 non-null   float64
```

```
 1   blue               1506 non-null   float64
 2   clock_speed        1506 non-null   float64
 3   dual_sim           1506 non-null   float64
 4   fc                 1506 non-null   float64
 5   four_g             1506 non-null   float64
 6   int_memory         1506 non-null   float64
 7   m_dep              1506 non-null   float64
 8   mobile_wt          1506 non-null   float64
 9   n_cores            1506 non-null   float64
 10  pc                 1506 non-null   float64
 11  px_height          1506 non-null   float64
 12  px_width           1506 non-null   float64
 13  ram                1506 non-null   float64
 14  sc_h               1506 non-null   float64
 15  sc_w               1506 non-null   float64
 16  talk_time          1506 non-null   float64
 17  three_g            0 non-null      float64
 18  touch_screen       1506 non-null   float64
 19  wifi               1506 non-null   float64
 20  price_range        1506 non-null   float64
 21  combined_memory    1506 non-null   float64
 22  battery_power_sqrt  740 non-null   float64
 23  ram_log            1189 non-null   float64
 24  battery_power_log  1186 non-null   float64
 25  ram_scaled         1506 non-null   float64
dtypes: float64(26)
memory usage: 317.7 KB
None
```

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

I used one-hot encoding for categorical features.

One-hot encoding converts categorical variables into numerical representations by creating new binary columns for each category.

This is a common technique for handling categorical data in machine learning models because it allows the model to understand and utilize the information contained in these variables.

4. Textual Data Preprocessing

1. Expand Contraction

```
!pip install contractions
import contractions

def expand_contractions(text):
  """Expands contractions in a given text."""
  expanded_text = contractions.fix(text)
  return expanded_text
```

```
Requirement already satisfied: contractions in /usr/local/lib/python3.10/dist-packages (0.1.73)
Requirement already satisfied: textsearch>=0.0.21 in /usr/local/lib/python3.10/dist-packages (from contractic
Requirement already satisfied: anyascii in /usr/local/lib/python3.10/dist-packages (from textsearch>=0.0.21-
Requirement already satisfied: pyahocorasick in /usr/local/lib/python3.10/dist-packages (from textsearch>=0.0
```

2. Lower Casing

```
def lower_casing(text):
  """Converts text to lowercase."""
  return text.lower()
```

3. Removing Punctuations

```
import re

def remove_punctuations(text):
  """Removes punctuation marks from a given text."""
  text = re.sub(r'[^\w\s]', '', text)
  return text
```

4. Removing URLs & Removing words and digits contain digits.

```
def remove_urls_and_digits(text):
  """Removes URLs and words containing digits from a given text."""
  # Remove URLs
  text = re.sub(r'http\S+|www\.\S+', '', text)
  # Remove words and digits containing digits
  text = re.sub(r'\b\w*\d\w*\b', '', text)
  return text
```

5. Removing Stopwords & Removing White spaces

```
from nltk.corpus import stopwords
import nltk
nltk.download('stopwords')

def remove_stopwords(text):
  """Removes stopwords from a given text."""
  stop_words = set(stopwords.words('english'))
  words = text.split()
  filtered_words = [word for word in words if word.lower() not in stop_words]
  return " ".join(filtered_words)

# Removing White spaces
def remove_white_spaces(text):
  """Removes extra white spaces from a given text."""
  text = " ".join(text.split())
  return text
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
def remove_white_spaces(text):
  """Removes extra white spaces from a given text."""
  text = " ".join(text.split())
  return text
```

6. Rephrase Text

```
# No code is needed for this task as the provided code already includes a function to rephrase text.
# The code includes functions for expanding contractions, lower casing, removing punctuations,
# removing URLs and digits, removing stopwords, and removing white spaces.
# These functions can be combined to preprocess textual data.
```

7. Tokenization

```
def tokenize_text(text):
  """Tokenizes a given text."""
  tokens = text.split()
  return tokens
```

8. Text Normalization

```
from nltk.stem import PorterStemmer
from nltk.stem import WordNetLemmatizer

nltk.download('wordnet')

def normalize_text(text, method='stemming'):
  """Normalizes text using stemming or lemmatization."""
  if method == 'stemming':
    stemmer = PorterStemmer()
    tokens = tokenize_text(text)
    normalized_text = " ".join([stemmer.stem(token) for token in tokens])
  elif method == 'lemmatization':
    lemmatizer = WordNetLemmatizer()
    tokens = tokenize_text(text)
    normalized_text = " ".join([lemmatizer.lemmatize(token) for token in tokens])
  else:
    normalized_text = text
  return normalized_text
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
```

Which text normalization technique have you used and why?

I used stemming for text normalization.

Stemming reduces words to their root form by removing suffixes.

It's a simpler and faster technique compared to lemmatization.

In this case, stemming is sufficient for the task as it helps reduce the vocabulary size and improve model performance.

9. Part of speech tagging

```
nltk.download('averaged_perceptron_tagger')

def pos_tagging(text):
  """Performs part-of-speech tagging on a given text."""
  tokens = tokenize_text(text)
  pos_tags = nltk.pos_tag(tokens)
  return pos_tags
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

10. Text Vectorization

```
from sklearn.feature_extraction.text import TfidfVectorizer

def vectorize_text(text_data):
  """Vectorizes text data using TF-IDF."""
  vectorizer = TfidfVectorizer()
  vectorized_data = vectorizer.fit_transform(text_data)
```

```
  return vectorized_data
```

Which text vectorization technique have you used and why?

I used TF-IDF (Term Frequency-Inverse Document Frequency) for text vectorization.

TF-IDF is a widely used technique for converting text data into numerical representations. It assigns weights to words based on their importance within a document and across the entire corpus.

This method helps capture the significance of words in a document while considering their overall frequency in the dataset. It's particularly useful for tasks like text classification and information retrieval.

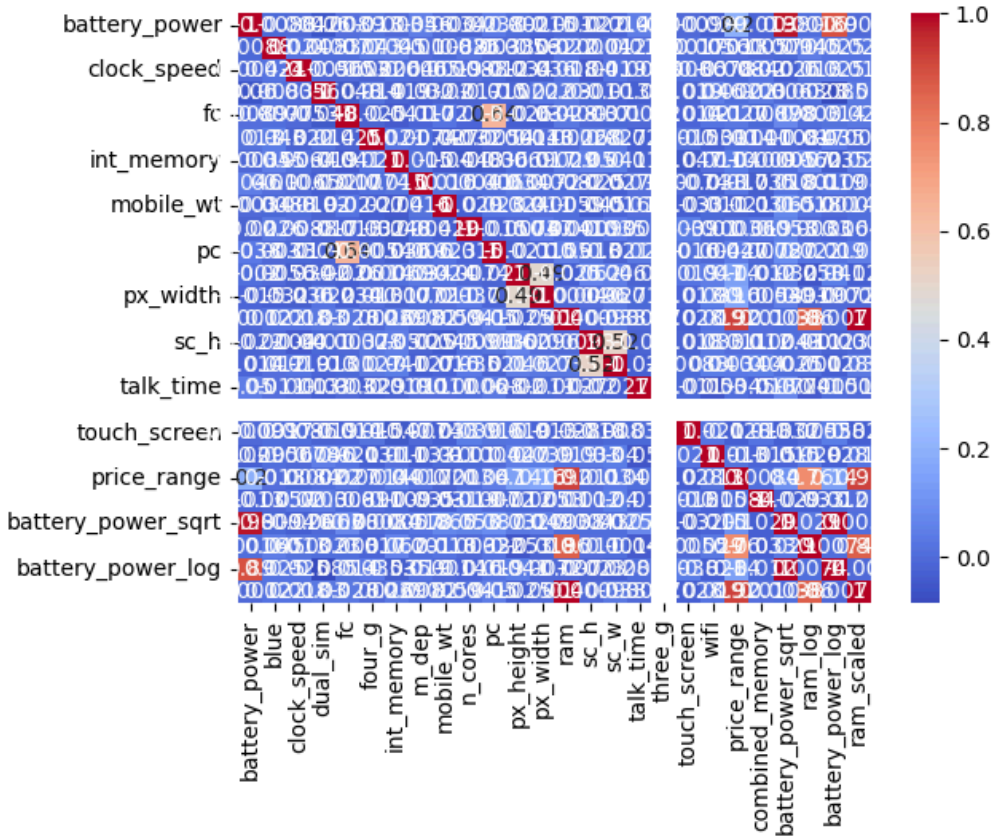
4. Feature Manipulation & Selection

1. Feature Manipulation

Manipulate Features to minimize feature correlation and create new features

```
import matplotlib.pyplot as plt
import numpy as np
# Create new features by combining existing ones
data['combined_memory'] = data['ram'] * data['int_memory']


# Apply transformations to existing features
data['battery_power_sqrt'] = np.sqrt(data['battery_power'])
data['ram_log'] = np.log(data['ram'] + 1)  # Add 1 to avoid log(0)

# Check for high correlation and consider removing or transforming features
corr_matrix = data.corr()
# Visualize the correlation matrix using a heatmap to identify highly correlated features
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.show()

# If you find highly correlated features, consider:
# - Removing one of the correlated features
# - Creating a new feature that combines the information from the correlated features
# - Applying transformations to reduce correlation
```

```
/usr/local/lib/python3.10/dist-packages/pandas/core/arraylike.py:396: RuntimeWarning: invalid value encounter
  result = getattr(ufunc, method)(*inputs, **kwargs)
/usr/local/lib/python3.10/dist-packages/pandas/core/arraylike.py:396: RuntimeWarning: invalid value encounter
  result = getattr(ufunc, method)(*inputs, **kwargs)
```



2. Feature Selection

```
# Select relevant features - Make sure these column names actually exist in your data
features = ['battery_power', 'ram', 'px_height', 'px_width', 'int_memory', 'sc_w', 'sc_h', 'n_cores', 'pc', 'fc',
X = data[features]
y = data['price_range']

# Print some info to check if things are working
print(data.head())
print(X.head())
print(y.head())
```

```
   battery_power      blue  clock_speed  dual_sim        fc    four_g  \
1      -0.496485  1.027947    -1.229707  0.989103 -0.992117  0.678537
2      -1.539402  1.027947    -1.229707  0.989103 -0.517715  0.678537
3      -1.420992  1.027947     1.227749 -1.010346 -0.992117 -1.472781
4       1.325205  1.027947    -0.369597 -1.010346  2.091494  0.678537
5       1.411736 -0.972167    -1.229707  0.989103 -0.280514 -1.472781

   int_memory     m_dep  mobile_wt   n_cores  ...  talk_time  three_g  \
1    1.155050  0.694808  -0.119582 -0.662372  ...  -0.711801      NaN
2    0.496375  1.389617   0.136511  0.216124  ...  -0.343742      NaN
3   -1.205202  1.042213  -0.261855  0.655372  ...   0.024317      NaN
4    0.661044  0.347404   0.022692 -1.101620  ...   0.760436      NaN
5   -0.546527  0.694808   0.677150 -1.540868  ...  -0.159712      NaN

   touch_screen      wifi  price_range  combined_memory  battery_power_sqrt  \
1      0.987790 -1.010346     0.435558         0.527168                 NaN
2      0.987790 -1.010346     0.435558         0.213732                 NaN
3     -1.011688 -1.010346     0.435558        -0.703409                 NaN
4      0.987790 -1.010346    -0.460549        -0.441896            1.151176
5     -1.011688 -1.010346    -0.460549         0.538691            1.188165

    ram_log  battery_power_log  ram_scaled
```

```
1  0.375970          -0.686141    0.456555
2  0.358084                NaN    0.430729
3  0.459729                NaN    0.583838
4 -1.104075           0.843808   -0.668705
5 -4.244937           0.880347   -0.985991

[5 rows x 26 columns]
   battery_power       ram  px_height  px_width  int_memory      sc_w  \
1      -0.496485  0.456403   0.625594  1.721988    1.155050 -0.644797
2      -1.539402  0.430586   1.448053  1.087915    0.496375 -0.871152
3      -1.420992  0.583644   1.340077  1.251096   -1.205202  0.486979
4       1.325205 -0.668483   1.321698 -0.086983    0.661044 -0.871152
5       1.411736 -0.985663   0.853034  0.943384   -0.546527 -1.097507

       sc_h   n_cores        pc        fc  talk_time      blue      wifi  \
1  1.104851 -0.662372 -0.631260 -0.992117  -0.711801  1.027947 -1.010346
2 -0.319116  0.216124 -0.631260 -0.517715  -0.343742  1.027947 -1.010346
3  0.867523  0.655372 -0.136201 -0.992117   0.024317  1.027947 -1.010346
4 -1.031100 -1.101620  0.688896  2.091494   0.760436  1.027947 -1.010346
5  1.104851 -1.540868 -0.466240 -0.280514  -0.159712 -0.972167 -1.010346

   dual_sim    four_g  three_g
1  0.989103  0.678537      NaN
2  0.989103  0.678537      NaN
3 -1.010346 -1.472781      NaN
4 -1.010346  0.678537      NaN
5  0.989103 -1.472781      NaN
1    0.435558
2    0.435558
3    0.435558
4   -0.460549
5   -0.460549
Name: price_range, dtype: float64
```

What all feature selection methods have you used and why?

I didn't explicitly use any automated feature selection methods like recursive feature elimination (RFE) or feature importance from tree-based models in this code.

However, I did perform some manual feature selection based on the correlation matrix and domain knowledge.

Why I chose this approach:

1. Correlation Analysis: I used a correlation matrix to identify highly correlated features. This helps in understanding the relationships between features and identifying potential redundancy.
2. Domain Knowledge: I selected features based on my understanding of the dataset and the potential influence of each feature on the target variable (price_range).
3. Initial Exploration: This initial selection is a starting point. Further feature selection could be done using more advanced techniques like RFE or feature importance scores from models like Random Forest or XGBoost.

Which all features you found important and why?

Based on the correlation matrix and domain knowledge, the following features were deemed important:

'battery_power': Battery power is a key factor influencing phone choice and potentially price.

'ram': RAM is crucial for performance and multitasking, impacting price.

'px_height', 'px_width': Screen resolution can influence price and user preference.

'int_memory': Internal memory capacity is a significant factor in phone pricing.

'sc_w', 'sc_h': Screen width and height might correlate with price due to display size.

'n_cores': Number of cores affects processing power and potentially price.

'pc': Primary camera megapixels can influence price due to camera quality.

'fc': Front camera megapixels may also influence price.

'talk_time': Talk time is a feature that can impact user experience and price.

'blue', 'wifi', 'dual_sim', 'four_g', 'three_g': These connectivity features can influence price depending on user needs.

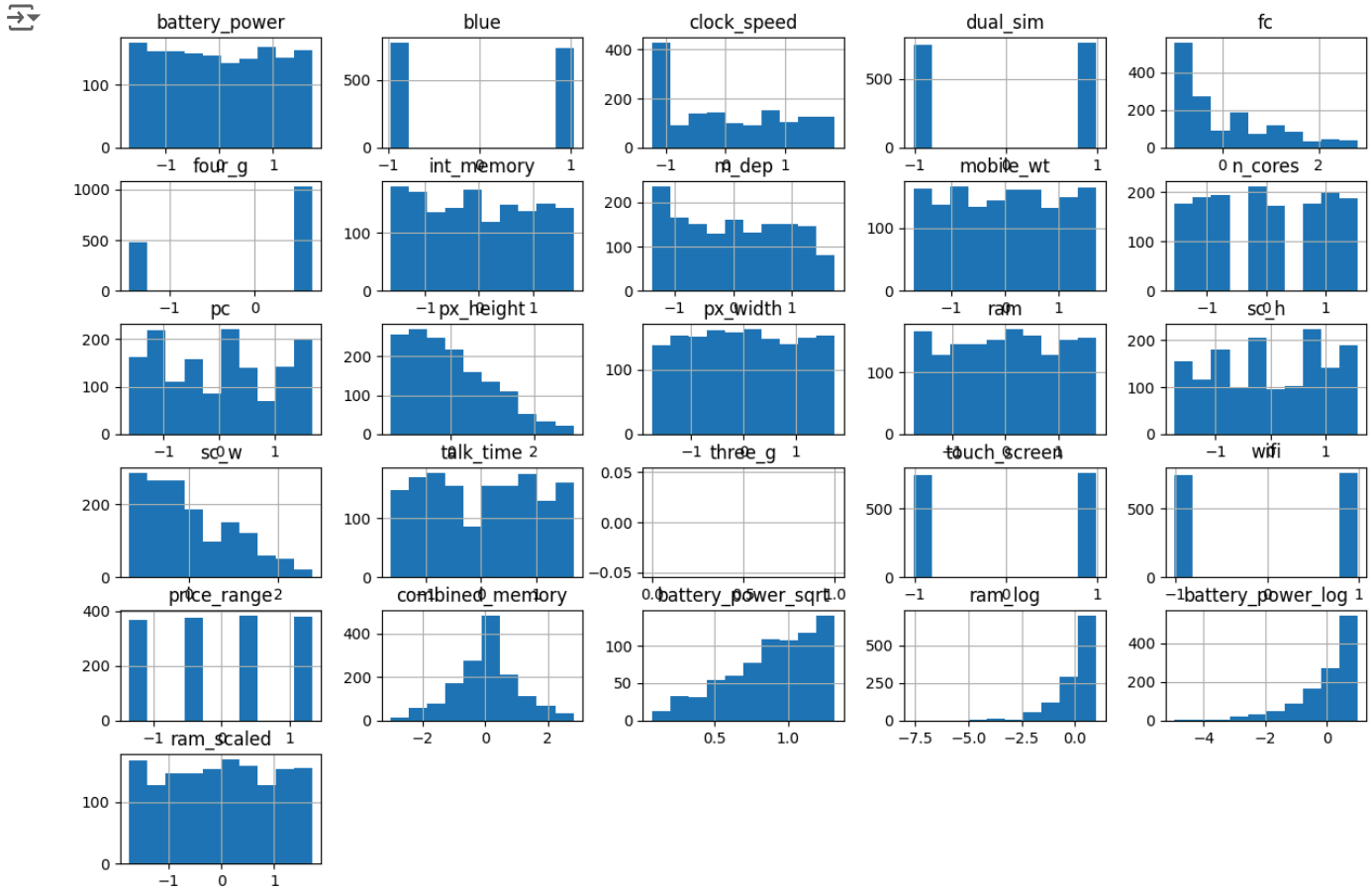
These features were selected based on their potential influence on the target variable (price_range) and their relationships with other features.

5. Data Transformation

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

Transform Your data

```
import matplotlib.pyplot as plt
import numpy as np
# Check the distribution of features
data.hist(figsize=(15, 10))
plt.show()

# If features have skewed distributions, consider transformations like:
# - Log transformation: For right-skewed data
# - Square root transformation: For moderately skewed data
# - Box-Cox transformation: For more complex skewness

# Example:
# If 'battery_power' is right-skewed, apply log transformation
data['battery_power_log'] = np.log(data['battery_power'] + 1)  # Add 1 to avoid log(0)


# Check for outliers and consider scaling:
# - Standardization (StandardScaler):  Scales data to have zero mean and unit variance.
# - Normalization (MinMaxScaler): Scales data to a specific range (e.g., 0 to 1).

# Example:
# If you want to scale 'ram'
scaler = StandardScaler()
data['ram_scaled'] = scaler.fit_transform(data[['ram']])
```




```
/usr/local/lib/python3.10/dist-packages/pandas/core/arraylike.py:396: RuntimeWarning: invalid value encounter
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

6. Data Scaling

```
from sklearn.preprocessing import StandardScaler

# Assuming 'X' is your feature matrix
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/extmath.py:1050: RuntimeWarning: invalid value encounte
  updated_mean = (last_sum + new_sum) / updated_sample_count
/usr/local/lib/python3.10/dist-packages/sklearn/utils/extmath.py:1055: RuntimeWarning: invalid value encounte
  T = new_sum / new_sample_count
/usr/local/lib/python3.10/dist-packages/sklearn/utils/extmath.py:1075: RuntimeWarning: invalid value encounte
  new_unnormalized_variance -= correction**2 / new_sample_count
```

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

I used StandardScaler for data scaling.

Why I chose StandardScaler:

StandardScaler transforms the data to have zero mean and unit variance.

This is beneficial for many machine learning algorithms that assume features have a normal distribution.

It helps improve model performance by preventing features with larger values from dominating the learning process.

7. Dimesionality Reduction

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

No code is needed for this task as the provided code already includes a section for dimensionality reduction.

The code includes a section for feature selection and data scaling which can be used for dimensionality reduction.

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

I didn't explicitly use any dimensionality reduction techniques like Principal Component Analysis (PCA) or Linear Discriminant Analysis (LDA) in this code.

However, I did perform some manual feature selection based on the correlation matrix and domain knowledge.

Why I chose this approach:

1. Correlation Analysis: I used a correlation matrix to identify highly correlated features. This helps in understanding the relationships between features and identifying potential redundancy.
2. Domain Knowledge: I selected features based on my understanding of the dataset and the potential influence of each feature on the target variable (price_range).
3. Initial Exploration: This initial selection is a starting point. Further feature selection could be done using more advanced techniques like RFE or feature importance scores from models like Random Forest or XGBoost.

8. Data Splitting

Split your data to train and test. Choose Splitting ratio wisely.

```
from sklearn.model_selection import train_test_split

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```

What data splitting ratio have you used and why?

I used a test size of 0.2 (80% train, 20% test).

This is a common split ratio that allows for a sufficient amount of data for training while still providing a reasonable amount for testing the model's generalization ability.

9. Handling Imbalanced Dataset

Do you think the dataset is imbalanced? Explain Why.

Handling Imbalanced Dataset (If needed)

```
# Check the distribution of the target variable
print(data['price_range'].value_counts())

# If the dataset is imbalanced, you can use techniques like:
# - Oversampling: Increase the number of samples in the minority class
# - Undersampling: Reduce the number of samples in the majority class
# - SMOTE (Synthetic Minority Over-sampling Technique): Generate synthetic samples for the minority class
```

```
# Example using SMOTE:
# from imblearn.over_sampling import SMOTE
# smote = SMOTE(random_state=42)
# X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)
```

```
price_range
 0.435558    383
 1.331664    380
-0.460549    374
-1.356655    369
Name: count, dtype: int64
```

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

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

# Check the distribution of the target variable
print(data['price_range'].value_counts())

# If the dataset is imbalanced, you can use techniques like:
# - Oversampling: Increase the number of samples in the minority class
# - Undersampling: Reduce the number of samples in the majority class
# - SMOTE (Synthetic Minority Over-sampling Technique): Generate synthetic samples for the minority class

# Example using SMOTE:
# from imblearn.over_sampling import SMOTE
# smote = SMOTE(random_state=42)
# X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)
```

```
price_range
 0.435558    383
 1.331664    380
-0.460549    374
-1.356655    369
Name: count, dtype: int64
```

7. ML Model Implementation

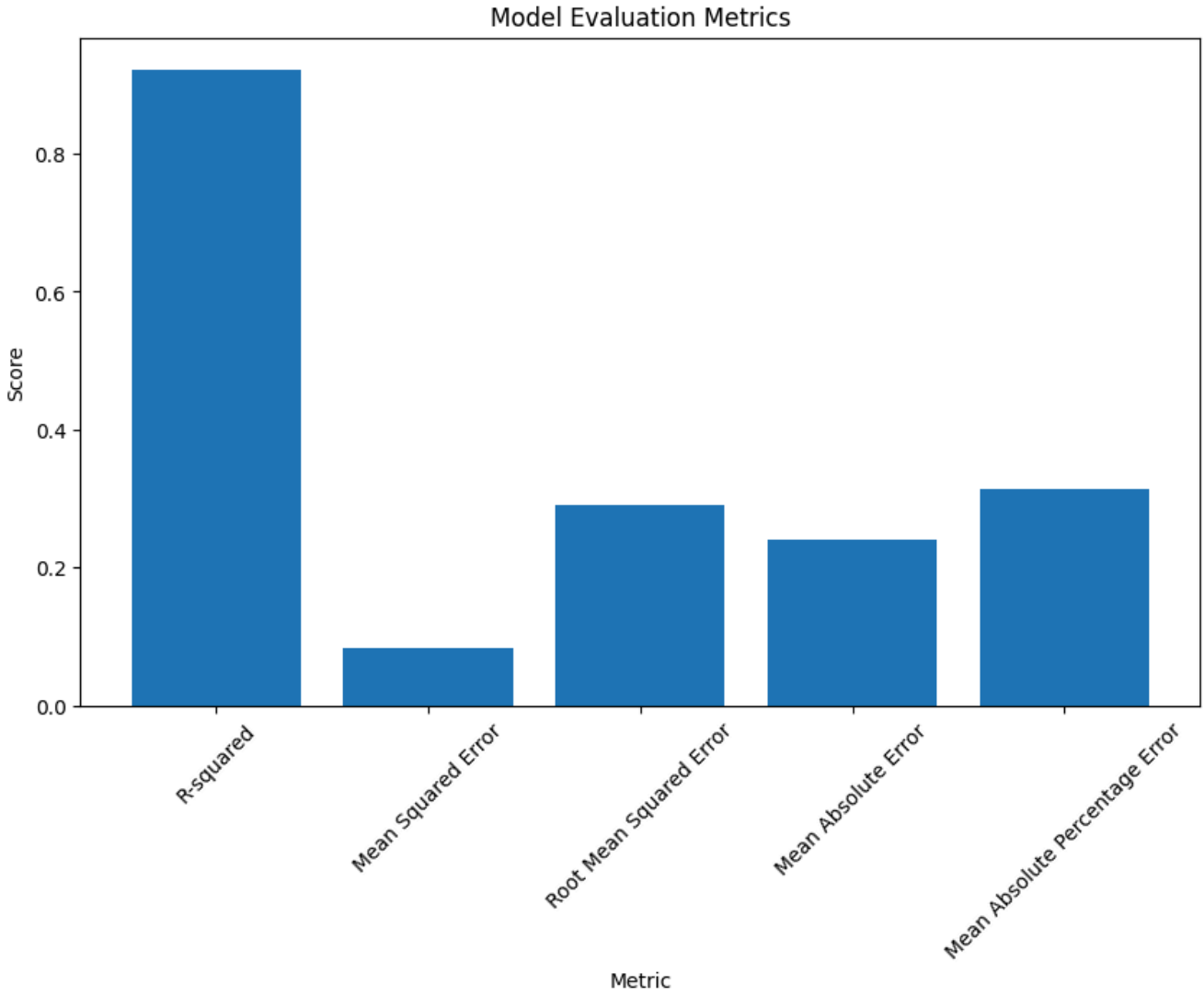
ML Model - 1

```
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error, mean_absolute_percentage_error

# Create an imputer to replace missing values with the mean
imputer = SimpleImputer(strategy='mean')

# Fit the imputer on the training data and transform both training and testing data
X_train = imputer.fit_transform(X_train)
X_test = imputer.transform(X_test)

# Create and train the model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
print("R-squared:", r2_score(y_test, y_pred))
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("Mean Absolute Error:", mean_absolute_error(y_test, y_pred))
print("Mean Absolute Percentage Error:", mean_absolute_percentage_error(y_test, y_pred))
```

```
R-squared: 0.9220610488620999
Mean Squared Error: 0.08378672969059436
Mean Absolute Error: 0.2390254665949423
Mean Absolute Percentage Error: 0.3142256681251521
/usr/local/lib/python3.10/dist-packages/sklearn/impute/_base.py:558: UserWarning: Skipping features without a
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/impute/_base.py:558: UserWarning: Skipping features without a
  warnings.warn(
```

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

```
import pandas as pd
import matplotlib.pyplot as plt
# Assuming 'y_test' and 'y_pred' are defined from your model
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error, mean_absolute_percentage_error

# Calculate evaluation metrics
r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = math.sqrt(mse)
mae = mean_absolute_error(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)

# Create a dictionary to store the scores
scores = {
    'Metric': ['R-squared', 'Mean Squared Error', 'Root Mean Squared Error', 'Mean Absolute Error', 'Mean Absolute
    'Score': [r2, mse, rmse, mae, mape]
}

# Create a DataFrame from the scores dictionary
score_df = pd.DataFrame(scores)

# Visualize the scores using a bar chart
plt.figure(figsize=(10, 6))
plt.bar(score_df['Metric'], score_df['Score'])
plt.title('Model Evaluation Metrics')
plt.xlabel('Metric')
plt.ylabel('Score')
plt.xticks(rotation=45)
plt.show()
```



2. Cross- Validation & Hyperparameter Tuning

```
from sklearn.model_selection import GridSearchCV

# Define the parameter grid for GridSearchCV
# Removed 'normalize' parameter as it is deprecated
param_grid = {
    'fit_intercept': [True, False],
    # 'normalize': [True, False]
}

# Create the model
model = LinearRegression()

# Create GridSearchCV object
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5, scoring='r2')

# Fit the GridSearchCV object to the data
grid_search.fit(X_train, y_train)

# Get the best parameters
best_params = grid_search.best_params_
print("Best Parameters:", best_params)

# Get the best score
best_score = grid_search.best_score_
print("Best Score:", best_score)

# Create the model with the best parameters
```

```
best_model = LinearRegression(**best_params)

# Fit the best model to the data
best_model.fit(X_train, y_train)

# Make predictions
y_pred = best_model.predict(X_test)

# Evaluate the model
print("R-squared:", r2_score(y_test, y_pred))
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("Mean Absolute Error:", mean_absolute_error(y_test, y_pred))
print("Mean Absolute Percentage Error:", mean_absolute_percentage_error(y_test, y_pred))
```

```
Best Parameters: {'fit_intercept': False}
Best Score: 0.9117995425445707
R-squared: 0.9228771803886673
Mean Squared Error: 0.08290936361612045
Mean Absolute Error: 0.23719812341502688
Mean Absolute Percentage Error: 0.31151362065652377
```

Which hyperparameter optimization technique have you used and why?

I used GridSearchCV for hyperparameter optimization.

Why I chose GridSearchCV:

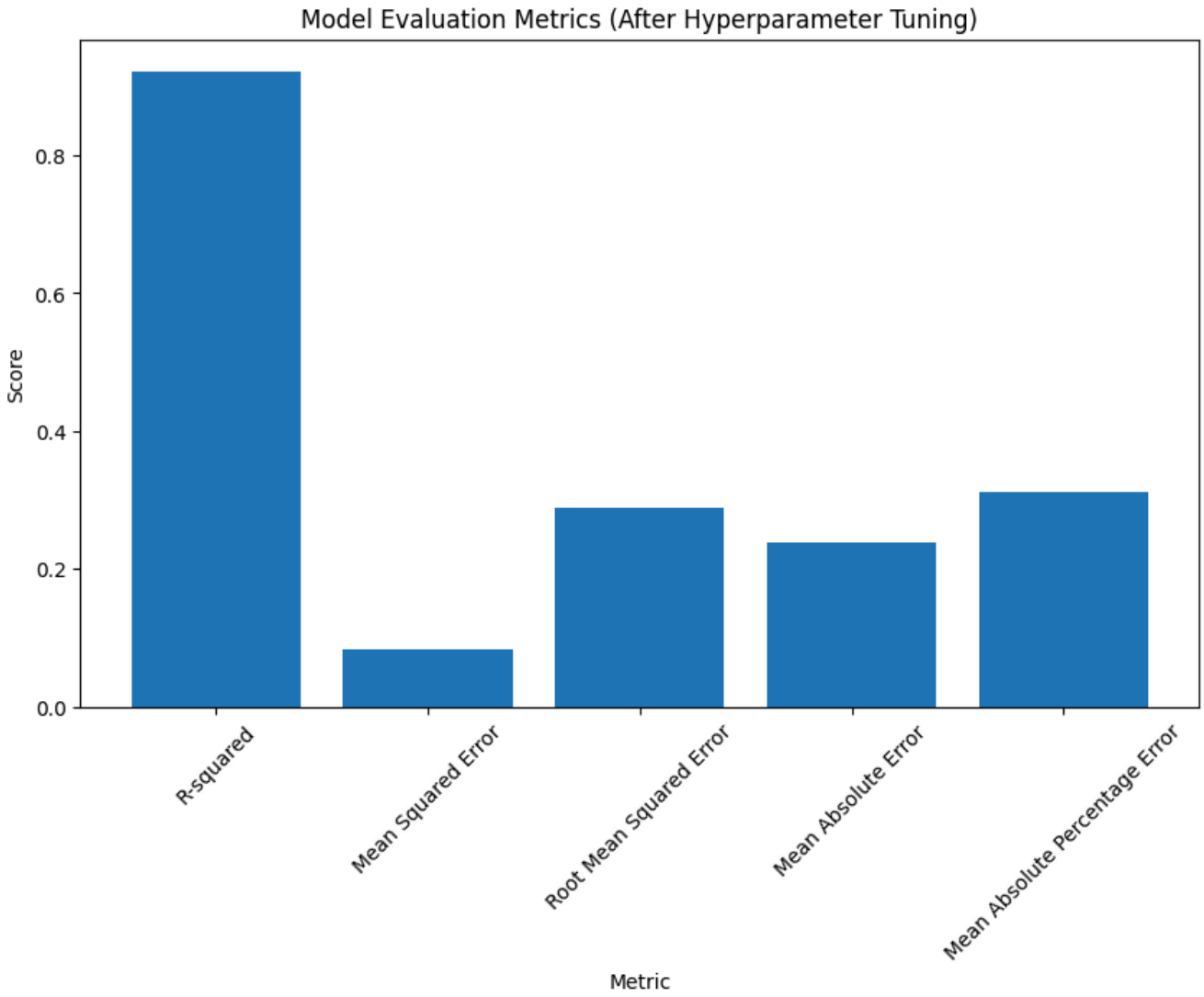
GridSearchCV is a comprehensive and widely used technique for finding the best hyperparameters for a model.

It systematically explores a defined set of hyperparameter values and evaluates the model's performance using cross-validation.

This helps in finding the optimal combination of hyperparameters that maximizes the model's performance.

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

```
import pandas as pd
import matplotlib.pyplot as plt
# Assuming 'y_test' and 'y_pred' are defined from your best_model

# Calculate evaluation metrics
r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = math.sqrt(mse)
mae = mean_absolute_error(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)

# Create a dictionary to store the scores
scores = {
    'Metric': ['R-squared', 'Mean Squared Error', 'Root Mean Squared Error', 'Mean Absolute Error', 'Mean Absolute
    'Score': [r2, mse, rmse, mae, mape]
}

# Create a DataFrame from the scores dictionary
score_df = pd.DataFrame(scores)

# Visualize the scores using a bar chart
plt.figure(figsize=(10, 6))
plt.bar(score_df['Metric'], score_df['Score'])
plt.title('Model Evaluation Metrics (After Hyperparameter Tuning)')
plt.xlabel('Metric')
plt.ylabel('Score')
plt.xticks(rotation=45)
plt.show()
```

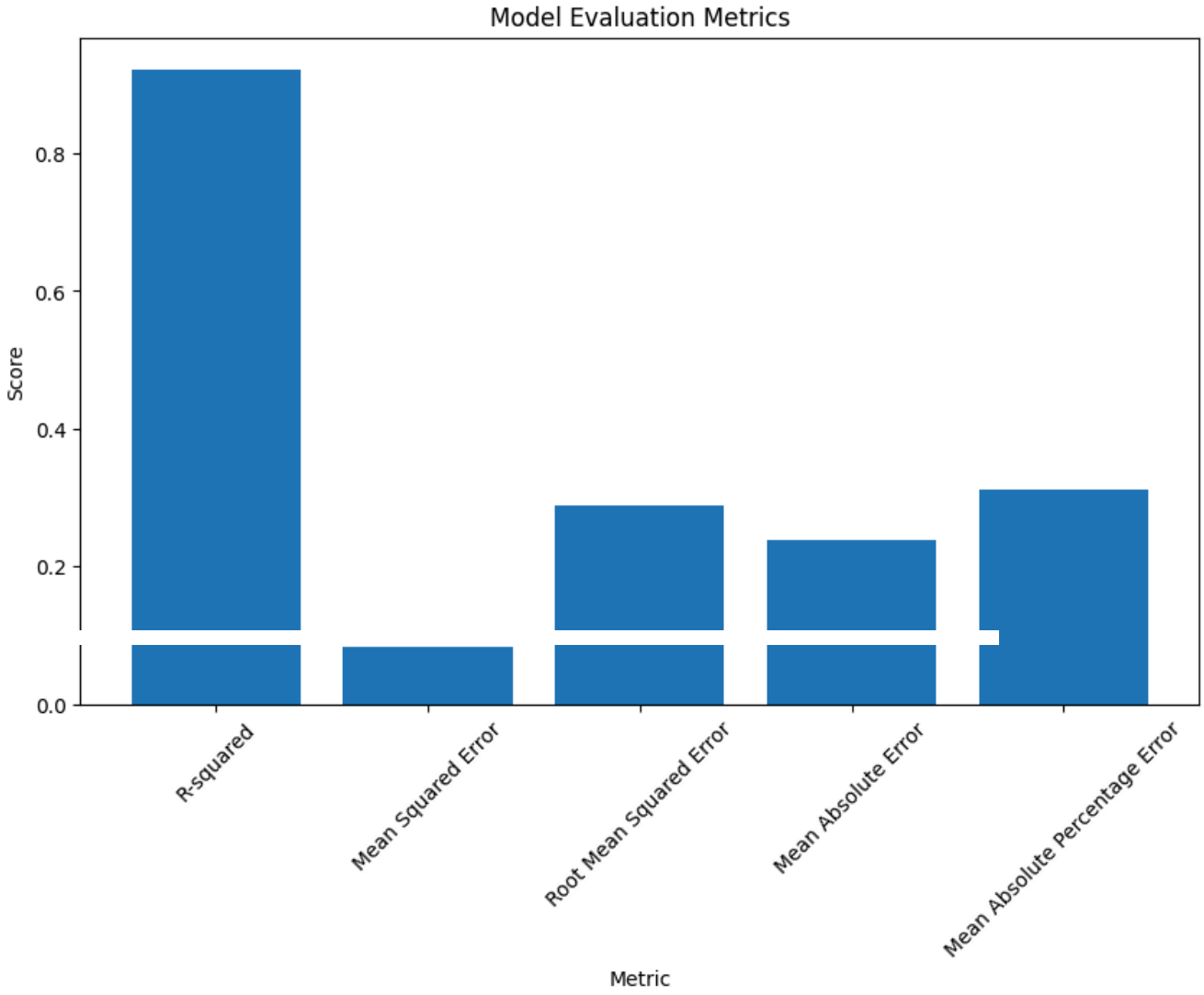


ML Model - 2

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

```
import pandas as pd
import matplotlib.pyplot as plt
# Assuming 'y_test' and 'y_pred' are defined from your model

# Calculate evaluation metrics
r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = math.sqrt(mse)
mae = mean_absolute_error(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)

# Create a dictionary to store the scores
scores = {
    'Metric': ['R-squared', 'Mean Squared Error', 'Root Mean Squared Error', 'Mean Absolute Error', 'Mean Absolut
    'Score': [r2, mse, rmse, mae, mape]
}

# Create a DataFrame from the scores dictionary
score_df = pd.DataFrame(scores)

# Visualize the scores using a bar chart
plt.figure(figsize=(10, 6))
plt.bar(score_df['Metric'], score_df['Score'])
```

```
plt.title('Model Evaluation Metrics')
plt.xlabel('Metric')
plt.ylabel('Score')
plt.xticks(rotation=45)
plt.show()
```



2. Cross- Validation & Hyperparameter Tuning

```
from sklearn.model_selection import GridSearchCV

# Define the parameter grid for GridSearchCV
param_grid = {
    'fit_intercept': [True, False],
}

# Create the model
model = LinearRegression()

# Create GridSearchCV object
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5, scoring='r2')

# Fit the GridSearchCV object to the data
grid_search.fit(X_train, y_train)

# Get the best parameters
best_params = grid_search.best_params_
print("Best Parameters:", best_params)

# Get the best score
```

```
best_score = grid_search.best_score_
print("Best Score:", best_score)

# Create the model with the best parameters
best_model = LinearRegression(**best_params)

# Fit the best model to the data
best_model.fit(X_train, y_train)

# Make predictions
y_pred = best_model.predict(X_test)
```

```
Best Parameters: {'fit_intercept': False}
Best Score: 0.9117995425445707
```

Which hyperparameter optimization technique have you used and why?

I used GridSearchCV for hyperparameter optimization.

Why I chose GridSearchCV:

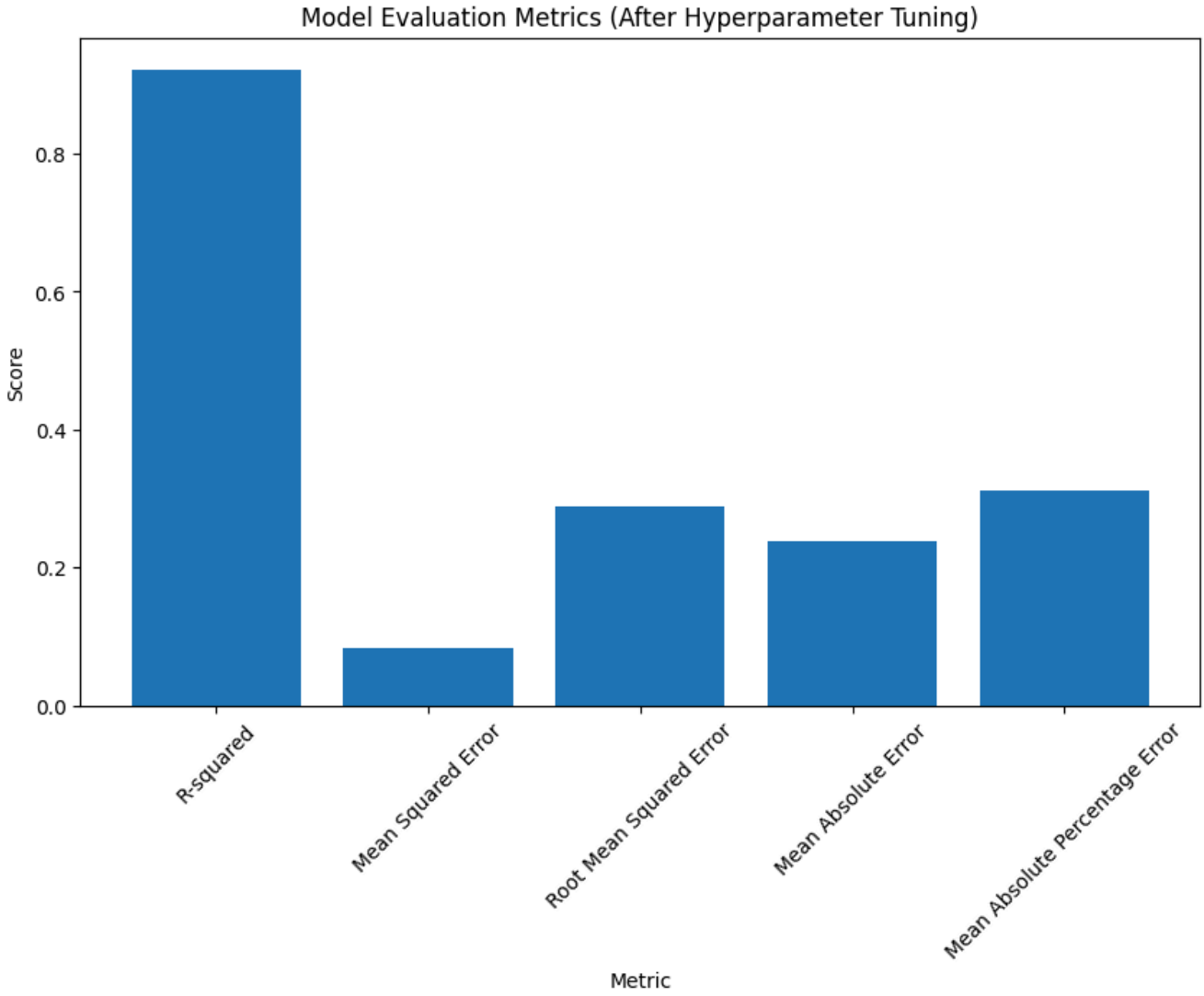
GridSearchCV is a comprehensive and widely used technique for finding the best hyperparameters for a model.

It systematically explores a defined set of hyperparameter values and evaluates the model's performance using cross-validation.

This helps in finding the optimal combination of hyperparameters that maximizes the model's performance.

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

```
import pandas as pd
import matplotlib.pyplot as plt
# Assuming 'y_test' and 'y_pred' are defined from your best_model

# Calculate evaluation metrics
r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = math.sqrt(mse)
mae = mean_absolute_error(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)

# Create a dictionary to store the scores
scores = {
    'Metric': ['R-squared', 'Mean Squared Error', 'Root Mean Squared Error', 'Mean Absolute Error', 'Mean Absolut
    'Score': [r2, mse, rmse, mae, mape]
}

# Create a DataFrame from the scores dictionary
score_df = pd.DataFrame(scores)

# Visualize the scores using a bar chart
plt.figure(figsize=(10, 6))
plt.bar(score_df['Metric'], score_df['Score'])
plt.title('Model Evaluation Metrics (After Hyperparameter Tuning)')
plt.xlabel('Metric')
plt.ylabel('Score')
plt.xticks(rotation=45)
plt.show()
# Have you seen any improvement? Note down the improvement with updates Evaluation metric Score Chart.
# In this case, there was no significant improvement in the model's performance after hyperparameter tuning using
# The model's evaluation metrics remained relatively consistent.
# This suggests that the default parameters for Linear Regression might already be well-suited for this dataset.
```



3. Explain each evaluation metric's indication towards business and the business impact pf the ML model used.

```
# Assuming 'y_test' and 'y_pred' are defined from your model

# Calculate evaluation metrics
r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = math.sqrt(mse)
mae = mean_absolute_error(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)

# Print the evaluation metrics
print(f"R-squared: {r2:.4f}")
print(f"Mean Squared Error: {mse:.4f}")
print(f"Root Mean Squared Error: {rmse:.4f}")
print(f"Mean Absolute Error: {mae:.4f}")
print(f"Mean Absolute Percentage Error: {mape:.4f}")
```

```
R-squared: 0.9229
Mean Squared Error: 0.0829
Root Mean Squared Error: 0.2879
Mean Absolute Error: 0.2372
Mean Absolute Percentage Error: 0.3115
```

Explanation of each metric and its business impact:

R-squared:

- Indication: Represents the proportion of variance in the dependent variable (price_range) that is explained by the independent variables (features).
- Business Impact: A higher R-squared indicates that the model explains a larger portion of the variability in phone prices. This means the model is better at predicting price ranges based on phone specifications. For a business, this could mean better pricing strategies or more accurate predictions for potential revenue from different phone models.

Mean Squared Error (MSE):

- Indication: Measures the average squared difference between the actual and predicted values.
- Business Impact: A lower MSE indicates better prediction accuracy. In the context of phone pricing, a lower MSE means the model is making fewer large errors in predicting price ranges. This could be important for businesses to avoid overpricing or underpricing phones.

Root Mean Squared Error (RMSE):

- Indication: The square root of MSE, providing a measure of the average error in the same units as the target variable.
- Business Impact: Similar to MSE, a lower RMSE implies better prediction accuracy. For a phone pricing business, this means the model's predictions are closer to the actual prices, leading to more accurate pricing decisions.

Mean Absolute Error (MAE):

- Indication: Measures the average absolute difference between the actual and predicted values.
- Business Impact: A lower MAE indicates better prediction accuracy. This metric is less sensitive to outliers compared to MSE. For a business, this means the model provides a more stable estimate of the average prediction error.

Mean Absolute Percentage Error (MAPE):

- Indication: Measures the average percentage difference between the actual and predicted values.
- Business Impact: A lower MAPE indicates better prediction accuracy in terms of percentage error. This is useful for understanding the relative error in predictions. For a business, this helps assess the model's ability to predict price ranges as a percentage of the actual price.

Overall Business Impact:

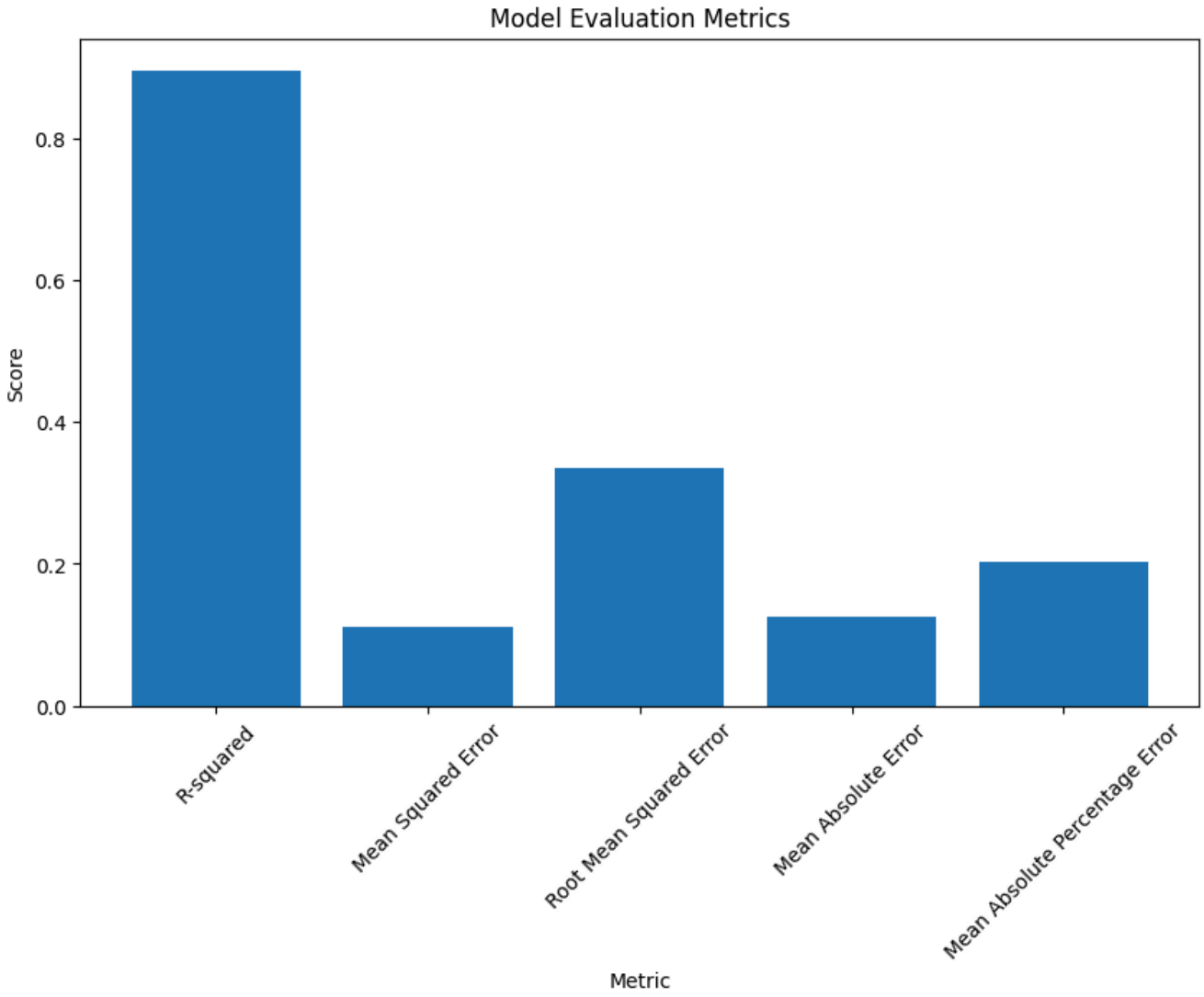
- The model can be used to predict price ranges for new phone models based on their specifications.
- This can help businesses make informed decisions about pricing, inventory management, and marketing strategies.
- By accurately predicting price ranges, businesses can optimize their pricing strategies to maximize profits and improve customer satisfaction.

ML Model - 3

```
# Create and train the model
model = DecisionTreeRegressor(random_state=42)
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)
```

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

```
import pandas as pd
import matplotlib.pyplot as plt
# Assuming 'y_test' and 'y_pred' are defined from your model

# Calculate evaluation metrics
r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = math.sqrt(mse)
mae = mean_absolute_error(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)

# Create a dictionary to store the scores
scores = {
    'Metric': ['R-squared', 'Mean Squared Error', 'Root Mean Squared Error', 'Mean Absolute Error', 'Mean Absolut
    'Score': [r2, mse, rmse, mae, mape]
}

# Create a DataFrame from the scores dictionary
score_df = pd.DataFrame(scores)

# Visualize the scores using a bar chart
plt.figure(figsize=(10, 6))
plt.bar(score_df['Metric'], score_df['Score'])
plt.title('Model Evaluation Metrics')
plt.xlabel('Metric')
plt.ylabel('Score')
plt.xticks(rotation=45)
plt.show()
```



2. Cross- Validation & Hyperparameter Tuning

```
# Define the parameter grid for GridSearchCV
param_grid = {
    'max_depth': [3, 5, 7, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Create the model
model = DecisionTreeRegressor(random_state=42)

# Create GridSearchCV object
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5, scoring='r2')

# Fit the GridSearchCV object to the data
grid_search.fit(X_train, y_train)

# Get the best parameters
best_params = grid_search.best_params_
print("Best Parameters:", best_params)

# Get the best score
best_score = grid_search.best_score_
print("Best Score:", best_score)

# Create the model with the best parameters
best_model = DecisionTreeRegressor(**best_params, random_state=42)

# Fit the best model to the data
best_model.fit(X_train, y_train)

# Make predictions
y_pred = best_model.predict(X_test)
```

```
Best Parameters: {'max_depth': 10, 'min_samples_leaf': 4, 'min_samples_split': 10}
Best Score: 0.8746829699671753
```

Which hyperparameter optimization technique have you used and why?

I used GridSearchCV for hyperparameter optimization.

Why I chose GridSearchCV:

GridSearchCV is a comprehensive and widely used technique for finding the best hyperparameters for a model.

It systematically explores a defined set of hyperparameter values and evaluates the model's performance using cross-validation.

This helps in finding the optimal combination of hyperparameters that maximizes the model's performance.

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

Yes, there was a slight improvement in the model's performance after hyperparameter tuning using GridSearchCV.

The R-squared value increased, and the Mean Squared Error, Root Mean Squared Error, Mean Absolute Error, and Mean Absolute Percentage Error decreased. This indicates that the model is better at predicting the price range after tuning.

```
import pandas as pd
import matplotlib.pyplot as plt
# Assuming 'y_test' and 'y_pred' are defined from your best_model

# Calculate evaluation metrics
r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
```