In [115]:

```
import pandas as pd  # For handling data
import numpy as np  # For numerical computations
import matplotlib.pyplot as plt  # For visualizations
import math
import seaborn as sns  # For better visualizations
from sklearn.model_selection import train_test_split  # To split data
from sklearn.preprocessing import OneHotEncoder, StandardScaler  # For preprocessing
from sklearn.ensemble import RandomForestRegressor  # For building model
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score  # For evaluation
import warnings
with warnings.catch_warnings(): warnings.simplefilter('ignore')
warnings.filterwarnings('ignore')
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from pptx import Presentation
from pptx.util import Inches
from sklearn.ensemble import GradientBoostingRegressor
```

Step 1: Load the Dataset

In [40]:

```
mobile_data=pd.read_csv('Mobile_data.csv')
```

In [42]:

```
mobile_data
```

Out[42]:

	Unnamed: 0	Model	Colour	Memory	RAM	Battery_	Rear Camera	Front Camera	AI Lens	Mobile Height	Processor_	Prize
0	0	Infinix SMART 7	Night Black	64	4	6000	13MP	5MP	1	16.76	Unisoc Spreadtrum SC9863A1	7,299
1	1	Infinix SMART 7	Azure Blue	64	4	6000	13MP	5MP	1	16.76	Unisoc Spreadtrum SC9863A1	7,299
2	2	MOTOROLA G32	Mineral Gray	128	8	5000	50MP	16MP	0	16.64	Qualcomm Snapdragon 680	11,999
3	3	POCO C50	Royal Blue	32	2	5000	8MP	5MP	0	16.56	Mediatek Helio A22	5,649
4	4	Infinix HOT 30i	Marigold	128	8	5000	50MP	5MP	1	16.76	G37	8,999
...	...	...	...	...	...	...	...	...	...	...	...	...
536	637	SAMSUNG Galaxy S23 5G	Cream	256	8	3900	50MP	12MP	0	15.49	Qualcomm Snapdragon 8 Gen 2	79,999
537	638	LAVA Z21	Cyan	32	2	3100	5MP	2MP	0	12.70	Octa Core	5,998
538	639	Tecno Spark 8T	Turquoise Cyan	64	4	5000	50MP	8MP	0	16.76	MediaTek Helio G35	9,990
539	641	SAMSUNG Galaxy A54 5G	Awesome Lime	128	8	5000	50MP	32MP	0	16.26	Exynos 1380, Octa Core	38,999
540	642	OPPO A77	Sky Blue	128	4	5000	50MP	8MP	0	16.66	Mediatek Helio G35	15,999

541 rows × 12 columns

In [9]:

```
print(mobile_data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 541 entries, 0 to 540
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Unnamed: 0      541 non-null    int64  
 1   Model           541 non-null    object 
 2   Colour          541 non-null    object 
 3   Memory          541 non-null    int64  
 4   RAM             541 non-null    int64  
 5   Battery_        541 non-null    int64  
 6   Rear Camera     541 non-null    object 
 7   Front Camera    541 non-null    object 
 8   AI Lens         541 non-null    int64  
 9   Mobile Height   541 non-null    float64
 10  Processor_      541 non-null    object 
 11  Prize           541 non-null    object 
dtypes: float64(1), int64(5), object(6)
memory usage: 50.8+ KB
None
```

In [11]:

```
mobile_data.head()
```

Out[11]:

	Unnamed: 0	Model	Colour	Memory	RAM	Battery_	Rear Camera	Front Camera	AI Lens	Mobile Height	Processor_	Prize
0	0	Infinix SMART 7	Night Black	64	4	6000	13MP	5MP	1	16.76	Unisoc Spreadtrum SC9863A1	7,299
1	1	Infinix SMART 7	Azure Blue	64	4	6000	13MP	5MP	1	16.76	Unisoc Spreadtrum SC9863A1	7,299
2	2	MOTOROLA G32	Mineral Gray	128	8	5000	50MP	16MP	0	16.64	Qualcomm Snapdragon 680	11,999
3	3	POCO C50	Royal Blue	32	2	5000	8MP	5MP	0	16.56	Mediatek Helio A22	5,649
4	4	Infinix HOT 30i	Marigold	128	8	5000	50MP	5MP	1	16.76	G37	8,999

In [13]:

```
mobile_data.tail()
```

Out[13]:

	Unnamed: 0	Model	Colour	Memory	RAM	Battery_	Rear Camera	Front Camera	AI Lens	Mobile Height	Processor_	Prize
536	637	SAMSUNG Galaxy S23 5G	Cream	256	8	3900	50MP	12MP	0	15.49	Qualcomm Snapdragon 8 Gen 2	79,999
537	638	LAVA Z21	Cyan	32	2	3100	5MP	2MP	0	12.70	Octa Core	5,998
538	639	Tecno Spark 8T	Turquoise Cyan	64	4	5000	50MP	8MP	0	16.76	MediaTek Helio G35	9,990
539	641	SAMSUNG Galaxy A54 5G	Awesome Lime	128	8	5000	50MP	32MP	0	16.26	Exynos 1380, Octa Core	38,999
540	642	OPPO A77	Sky Blue	128	4	5000	50MP	8MP	0	16.66	Mediatek Helio G35	15,999

In [15]:

```
mobile_data.shape
```

Out[15]:

```
(541, 12)
```

In [9]:

```
# Check for missing values
missing_values = mobile_data.isnull().sum()
missing_values[missing_values > 0]  # Show only columns with missing values
```

Out[9]:

```
Series([], dtype: int64)
```

In [44]:

```
mobile_data.isnull().sum()
```

Out[44]:

```
Unnamed: 0        0
Model             0
Colour            0
Memory            0
RAM               0
Battery_          0
Rear Camera       0
Front Camera      0
AI Lens           0
Mobile Height     0
Processor_        0
Prize             0
dtype: int64
```

In [46]:

```
mobile_data.drop(columns='Unnamed: 0', inplace=True, errors='ignore')
mobile_data
```

Out[46]:

	Model	Colour	Memory	RAM	Battery_	Rear Camera	Front Camera	AI Lens	Mobile Height	Processor_	Prize
0	Infinix SMART 7	Night Black	64	4	6000	13MP	5MP	1	16.76	Unisoc Spreadtrum SC9863A1	7,299
1	Infinix SMART 7	Azure Blue	64	4	6000	13MP	5MP	1	16.76	Unisoc Spreadtrum SC9863A1	7,299
2	MOTOROLA G32	Mineral Gray	128	8	5000	50MP	16MP	0	16.64	Qualcomm Snapdragon 680	11,999
3	POCO C50	Royal Blue	32	2	5000	8MP	5MP	0	16.56	Mediatek Helio A22	5,649
4	Infinix HOT 30i	Marigold	128	8	5000	50MP	5MP	1	16.76	G37	8,999
...	...	...	...	...	...	...	...	...	...	...	...
536	SAMSUNG Galaxy S23 5G	Cream	256	8	3900	50MP	12MP	0	15.49	Qualcomm Snapdragon 8 Gen 2	79,999
537	LAVA Z21	Cyan	32	2	3100	5MP	2MP	0	12.70	Octa Core	5,998
538	Tecno Spark 8T	Turquoise Cyan	64	4	5000	50MP	8MP	0	16.76	MediaTek Helio G35	9,990
539	SAMSUNG Galaxy A54 5G	Awesome Lime	128	8	5000	50MP	32MP	0	16.26	Exynos 1380, Octa Core	38,999
540	OPPO A77	Sky Blue	128	4	5000	50MP	8MP	0	16.66	Mediatek Helio G35	15,999

541 rows × 11 columns

In [48]:

```
# Check for duplicate rows
print(f"Duplicate rows: {mobile_data.duplicated().sum()}")
```

```
# Remove duplicates
mobile_data= mobile_data.drop_duplicates()
```

```
Duplicate rows: 10
```

In [50]:

```
# Check for duplicate rows after the dupilcate are dropped
print(f"Duplicate rows: {mobile_data.duplicated().sum()}")
```

```
Duplicate rows: 0
```

In [38]:

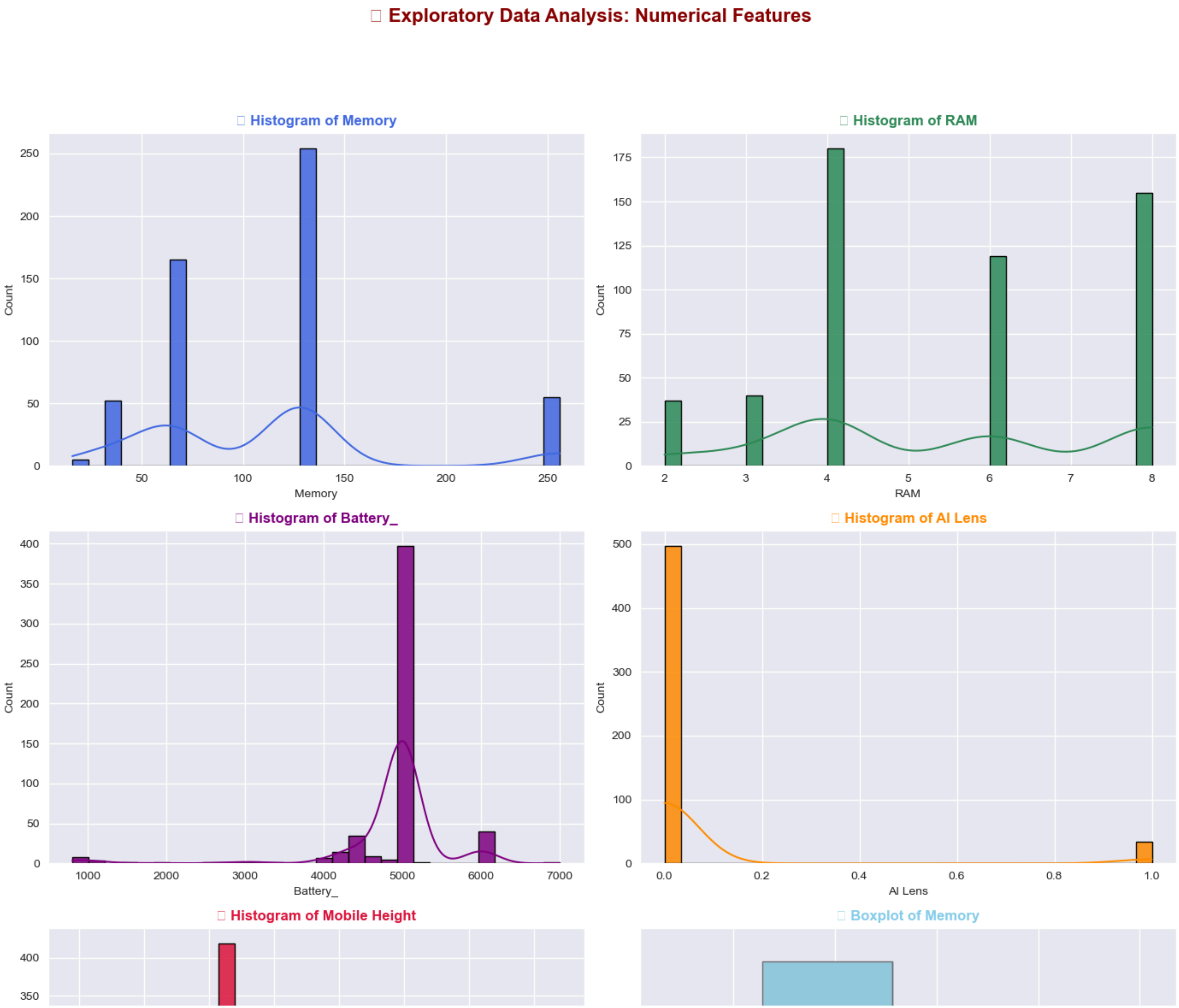
```
#Fill categorical missing values using the most frequent value:
# Fill missing values based on column types
for column in mobile_data.columns:
    if mobile_data[column].dtype == "object":  # If categorical
        mobile_data[column].fillna(mobile_data[column].mode()[0], inplace=True)
    else:  # If numerical
        mobile_data[column].fillna(mobile_data[column].median(), inplace=True)

# Verify again
print(mobile_data.isnull().sum().sum())  # Should return 0 if all missing values are handled
```

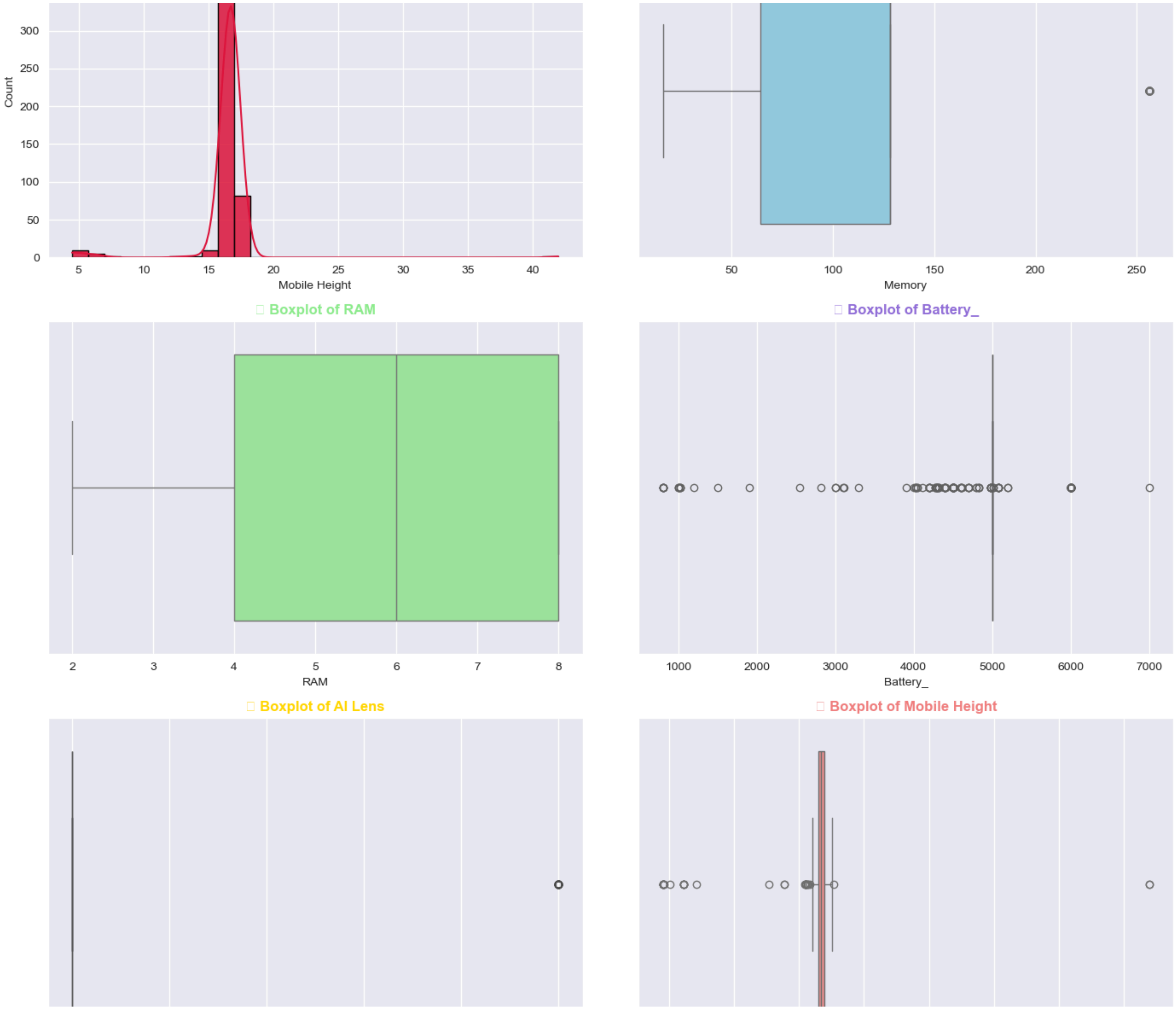
```
0
```

In [52]:

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

# Identify numerical columns
numerical_cols = mobile_data.select_dtypes(include=["int64", "float64"]).columns

# Number of numerical columns
num_cols = len(numerical_cols)
cols = 2  # Two columns: one for histogram, one for boxplot
rows = math.ceil(num_cols)  # Ensure enough rows

# Set a modern Seaborn theme
sns.set_style("darkgrid")

plt.figure(figsize=(14, rows * 5))
plt.suptitle("📊 Exploratory Data Analysis: Numerical Features", fontsize=16, fontweight="bold", color="darkred")

# Define color palette
hist_colors = ["royalblue", "seagreen", "purple", "darkorange", "crimson"]
```

```python
box_colors = ["skyblue", "lightgreen", "mediumpurple", "gold", "lightcoral"]

# Loop through all numerical columns
for i, col in enumerate(numerical_cols):
    color_hist = hist_colors[i % len(hist_colors)]
    color_box = box_colors[i % len(box_colors)]

    # Create histogram
    plt.subplot(rows, cols, i + 1)  # Adjusted subplot index
    sns.histplot(mobile_data[col], bins=30, kde=True, color=color_hist, edgecolor="black", alpha=0.85)
    plt.title(f"📈 Histogram of {col}", fontsize=12, fontweight="bold", color=color_hist)
    plt.xlabel(col, fontsize=10)
    plt.ylabel("Count", fontsize=10)

    # Create boxplot
    plt.subplot(rows, cols, i + num_cols + 1)  # Adjusted subplot index
    sns.boxplot(x=mobile_data[col], color=color_box)
    plt.title(f"📉 Boxplot of {col}", fontsize=12, fontweight="bold", color=color_box)

plt.tight_layout(rect=[0, 0, 1, 0.95])  # Adjust layout to fit main title
plt.show()
```


□ Exploratory Data Analysis: Numerical Features
□ Histogram of Memory
Count
0
50
100
150
200
250
50
100
150
200
250
Memory
□ Histogram of RAM
Count
0
25
50
75
100
125
150
175
2
3
4
5
6
7
8
RAM
□ Histogram of Battery_
Count
0
50
100
150
200
250
300
350
400
1000
2000
3000
4000
5000
6000
7000
Battery_
□ Histogram of AI Lens
Count
0
100
200
300
400
500
0.0
0.2
0.4
0.6
0.8
1.0
AI Lens
□ Histogram of Mobile Height
400
350
□ Boxplot of Memory



□ Boxplot of RAM

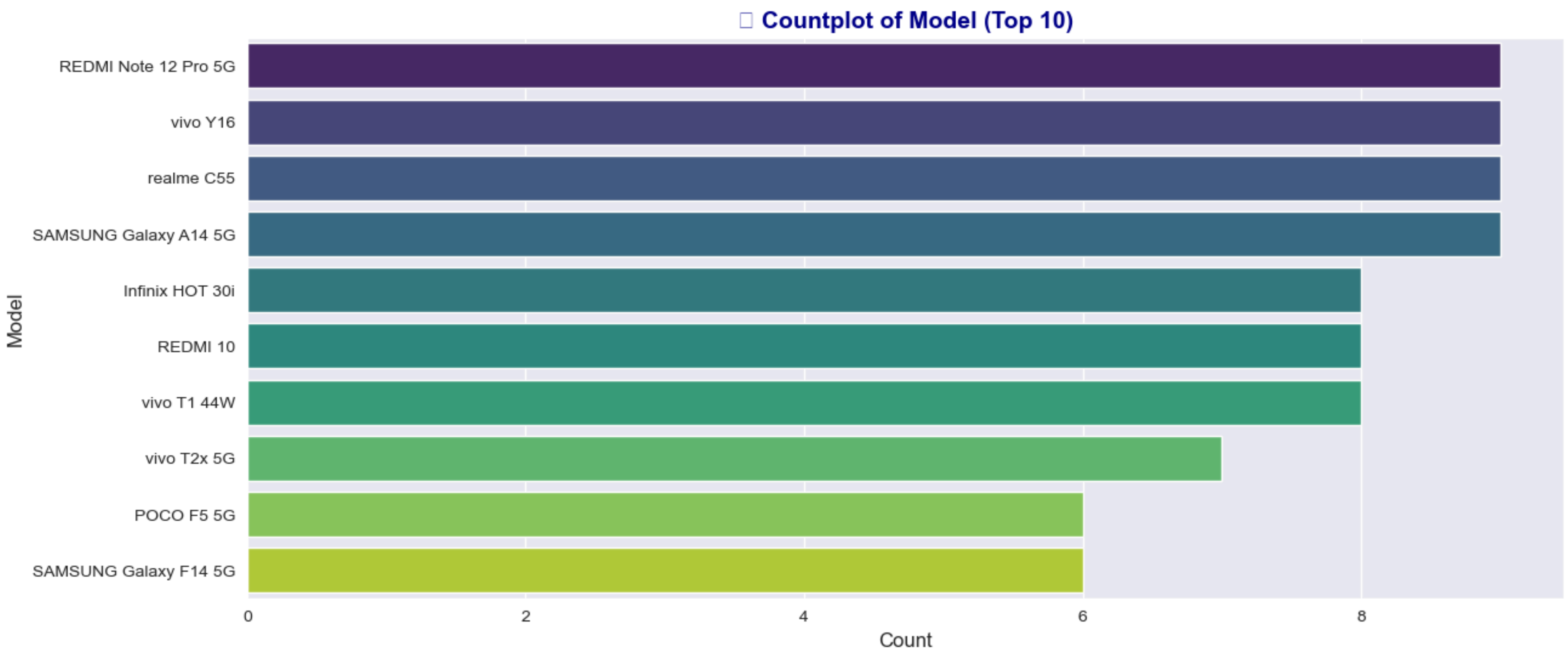
□ Boxplot of Battery_

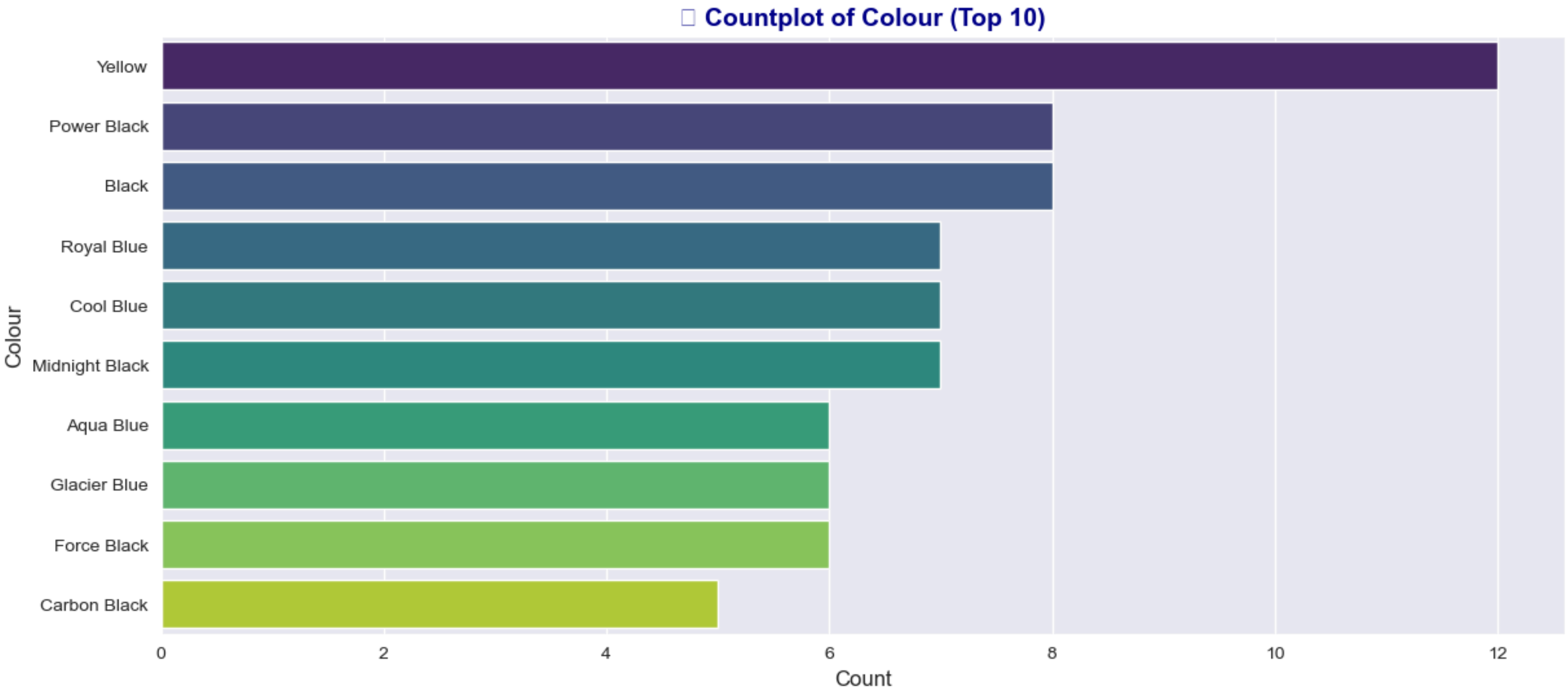
□ Boxplot of AI Lens

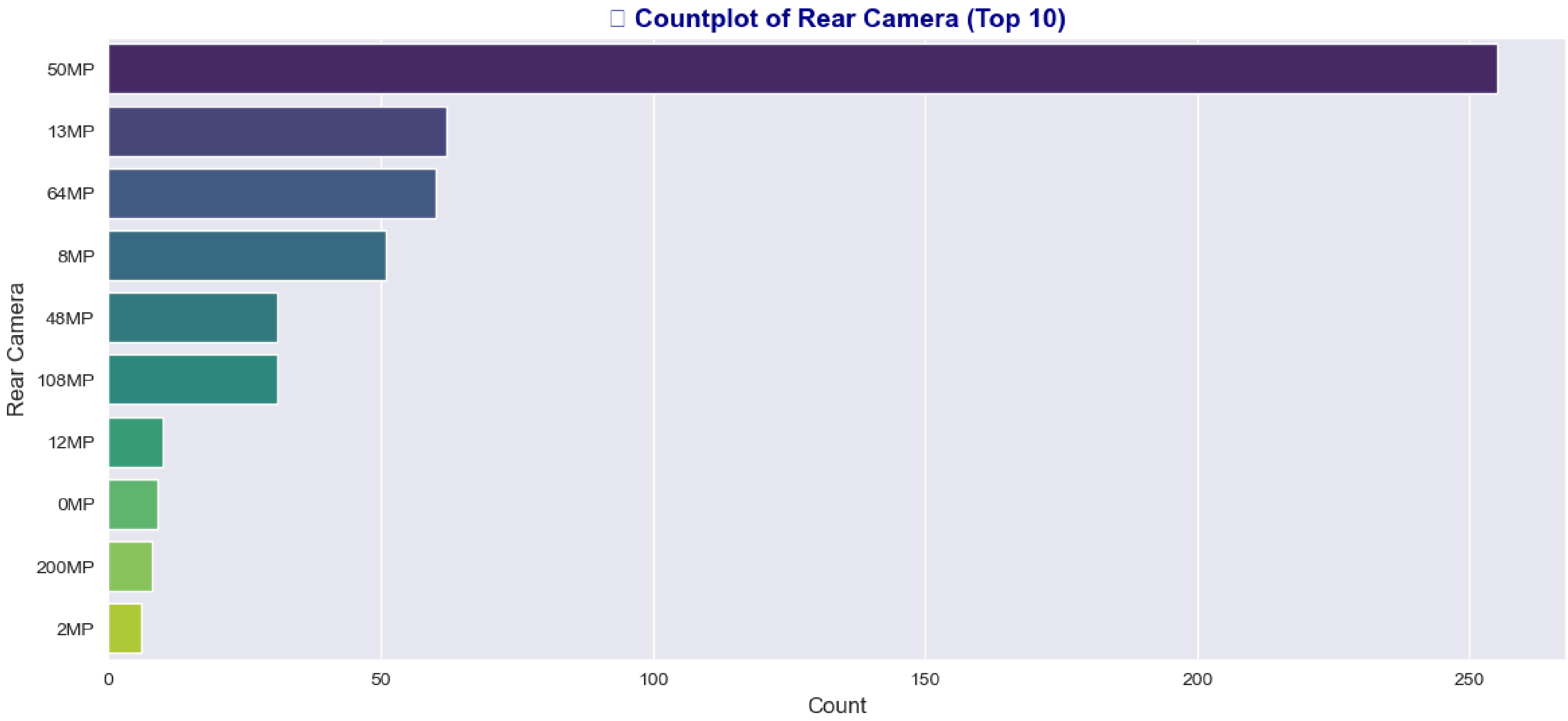
□ Boxplot of Mobile Height

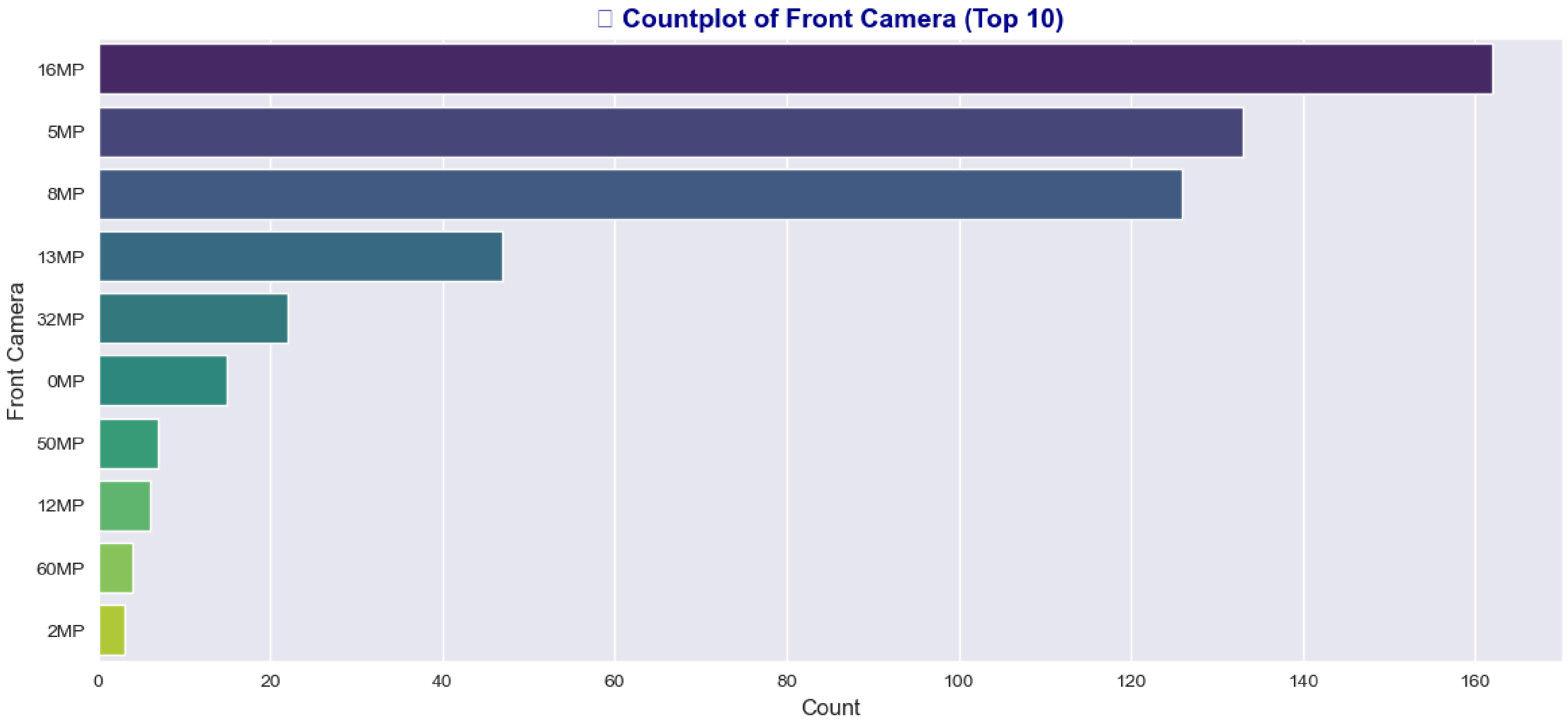
In [54]:

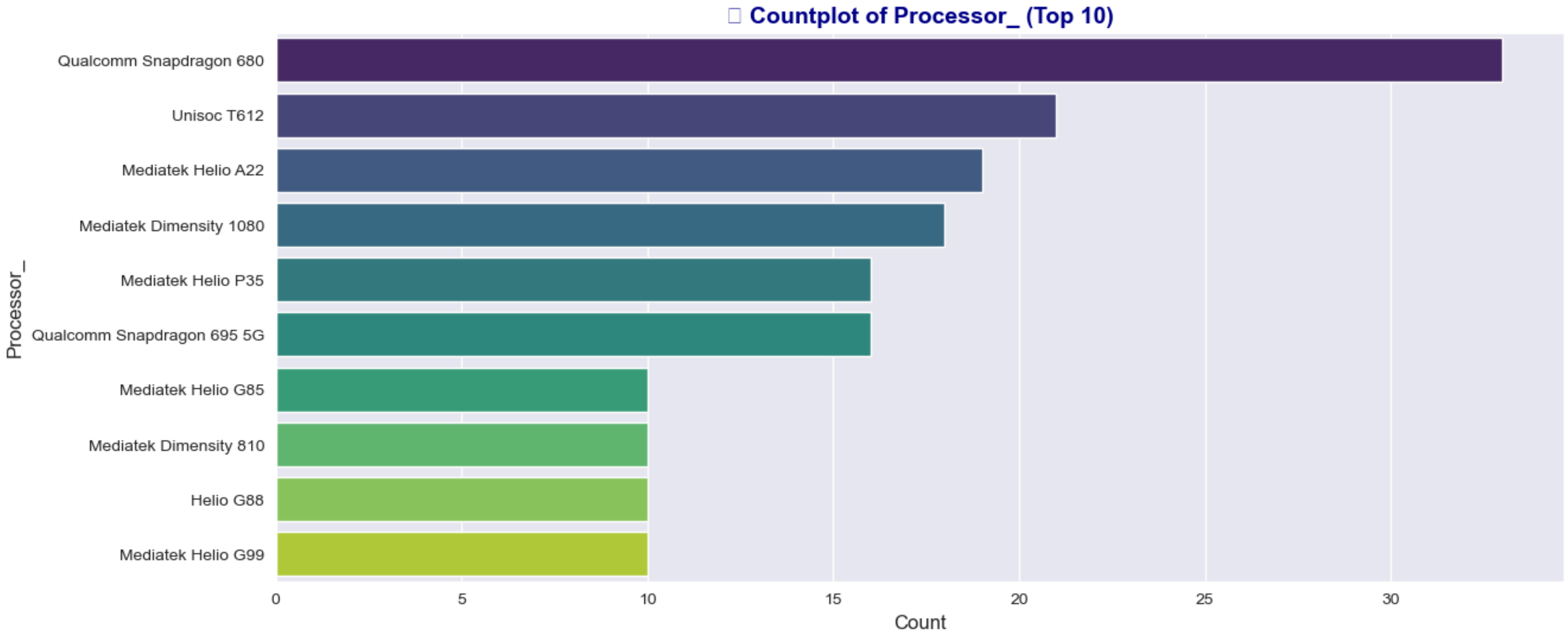
```python
import matplotlib.pyplot as plt
import seaborn as sns

# Identify categorical columns
categorical_cols = mobile_data.select_dtypes(include=["object"]).columns

# Set Seaborn style
sns.set_style("darkgrid")

# Generate count plots for categorical columns
for col in categorical_cols:
    plt.figure(figsize=(14, 6))  # Increase figure size

    # Show only top 10 categories to avoid clutter
    top_10_values = mobile_data[col].value_counts().nlargest(10).index
    filtered_data = mobile_data[mobile_data[col].isin(top_10_values)]

    sns.countplot(y=filtered_data[col], order=top_10_values, palette="viridis")

    plt.title(f"📊 Countplot of {col} (Top 10)", fontsize=14, fontweight="bold", color="darkblue")
    plt.ylabel(col, fontsize=12)
    plt.xlabel("Count", fontsize=12)

    plt.xticks(rotation=0)  # Keep x labels horizontal
    plt.yticks(fontsize=10)  # Increase y-axis font size

    plt.show()
```

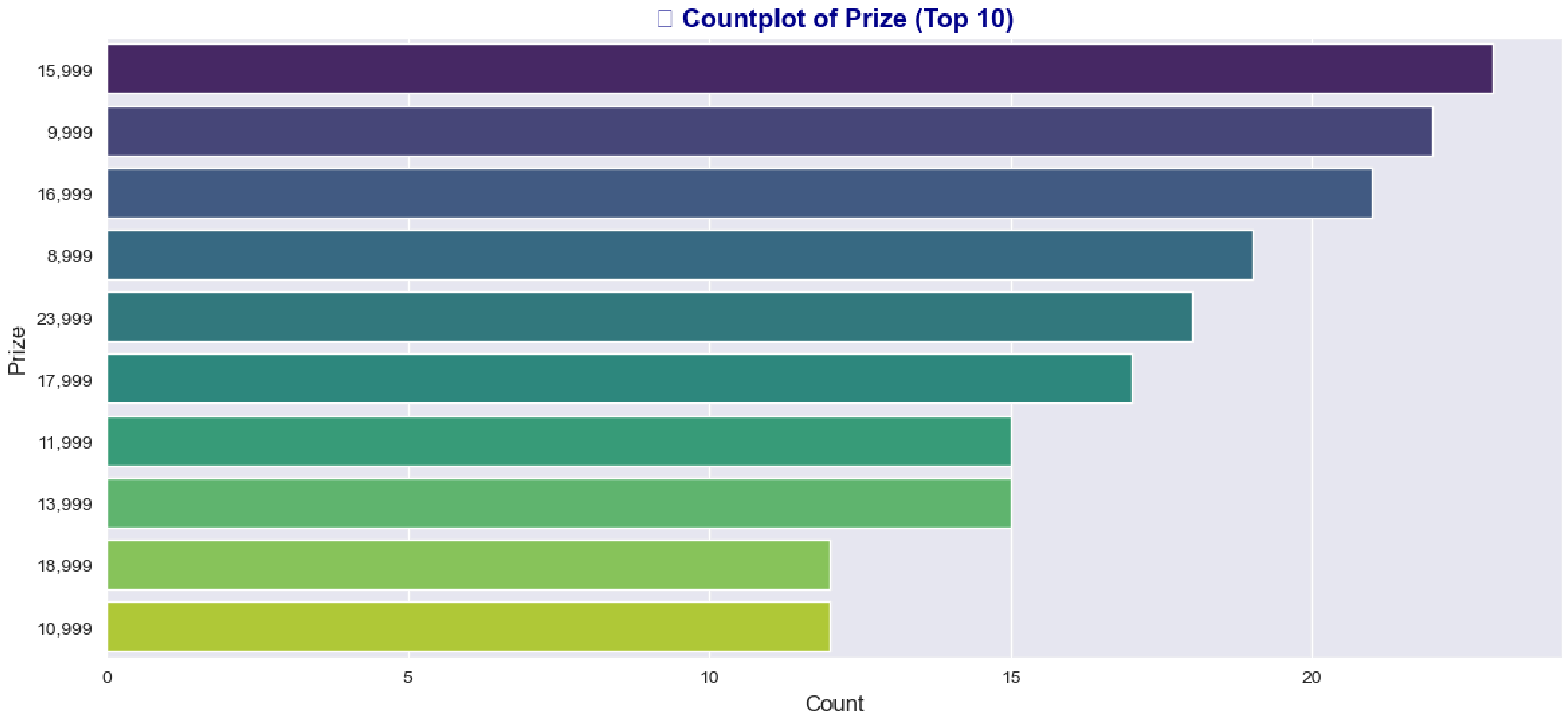

□ Countplot of Model (Top 10)
Model
REDMI Note 12 Pro 5G
vivo Y16
realme C55
SAMSUNG Galaxy A14 5G
Infinix HOT 30i
REDMI 10
vivo T1 44W
vivo T2x 5G
POCO F5 5G
SAMSUNG Galaxy F14 5G
0
2
4
6
8
Count


□ Countplot of Colour (Top 10)
Yellow
Power Black
Black
Royal Blue
Cool Blue
Midnight Black
Aqua Blue
Glacier Blue
Force Black
Carbon Black
Colour
0
2
4
6
8
10
12
Count


□ Countplot of Rear Camera (Top 10)
Rear Camera
50MP
13MP
64MP
8MP
48MP
108MP
12MP
0MP
200MP
2MP
0
50
100
150
200
250
Count

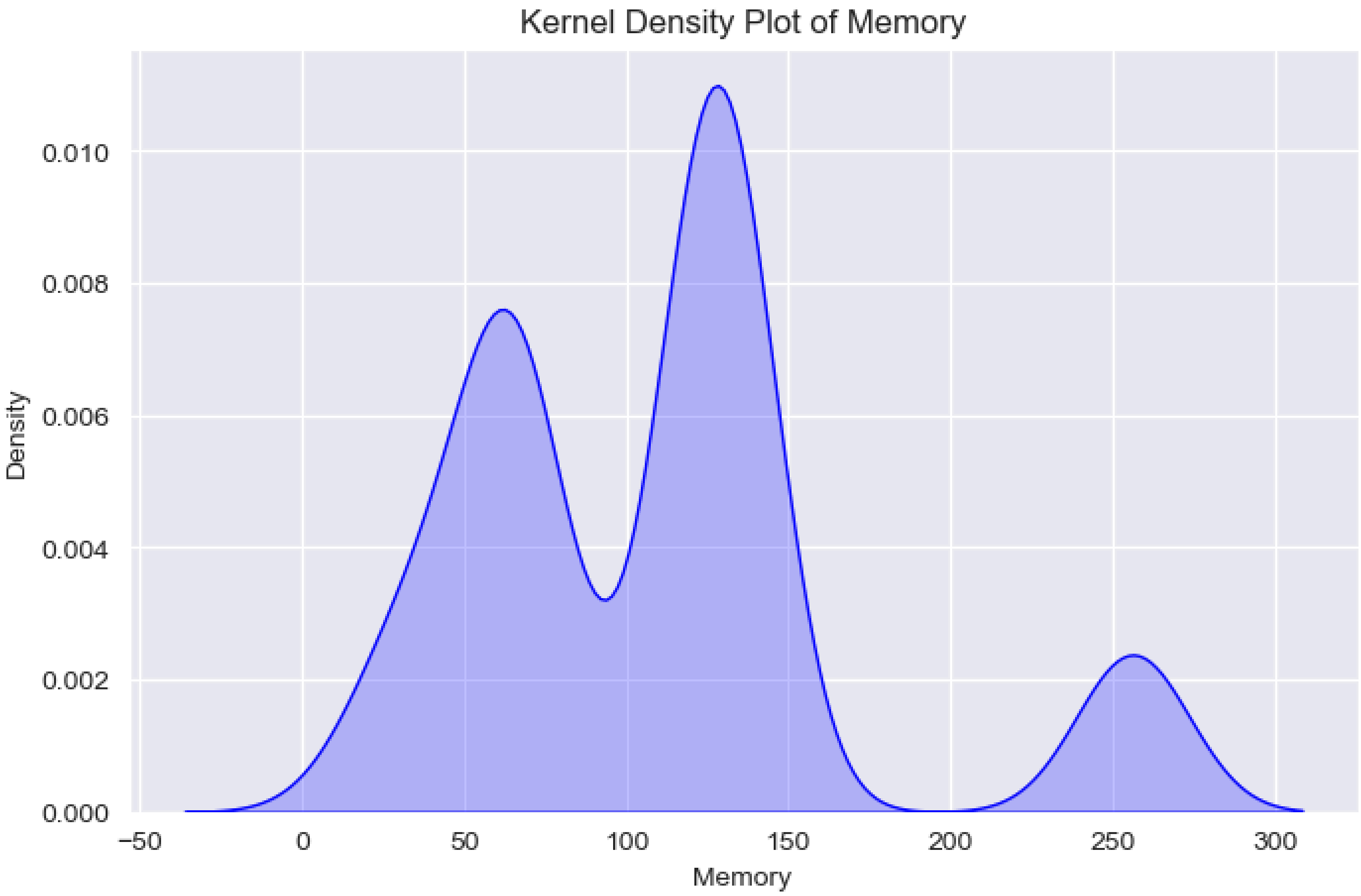

□ Countplot of Front Camera (Top 10)
Front Camera
16MP
5MP
8MP
13MP
32MP
0MP
50MP
12MP
60MP
2MP
0
20
40
60
80
100
120
140
160
Count

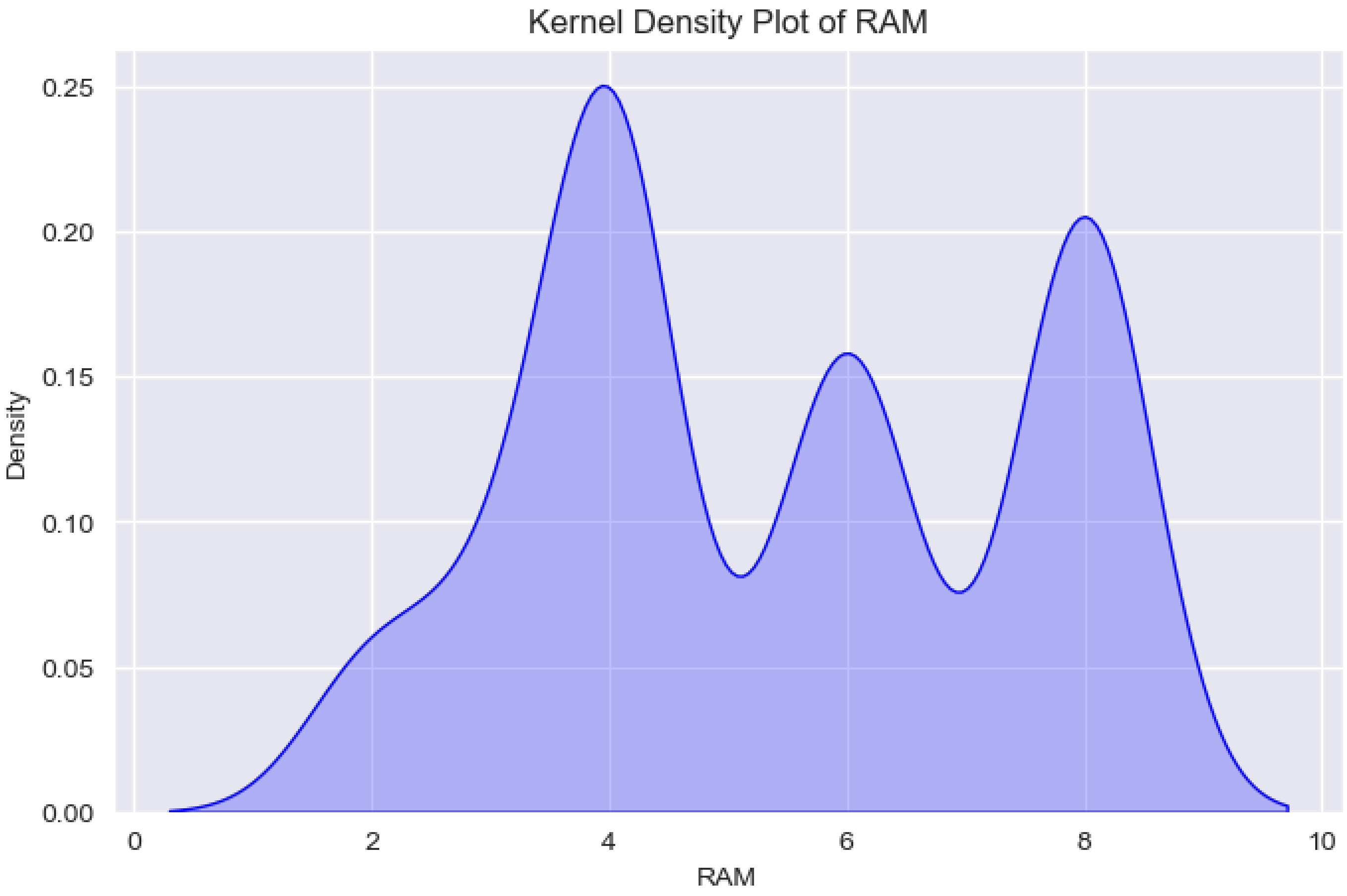

□ Countplot of Processor_ (Top 10)
Processor_
Qualcomm Snapdragon 680
Unisoc T612
Mediatek Helio A22
Mediatek Dimensity 1080
Mediatek Helio P35
Qualcomm Snapdragon 695 5G
Mediatek Helio G85
Mediatek Dimensity 810
Helio G88
Mediatek Helio G99
0
5
10
15
20
25
30
Count

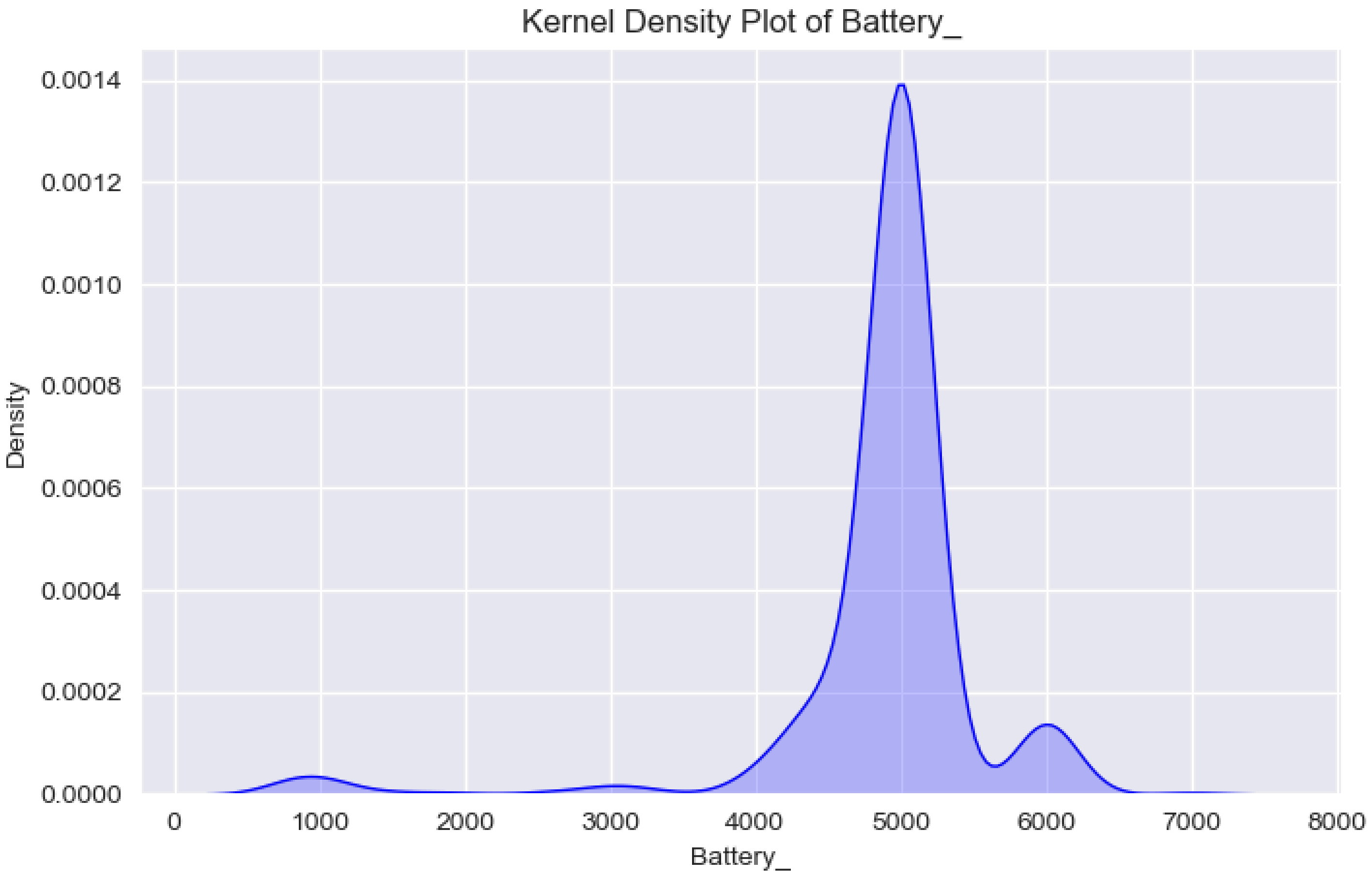


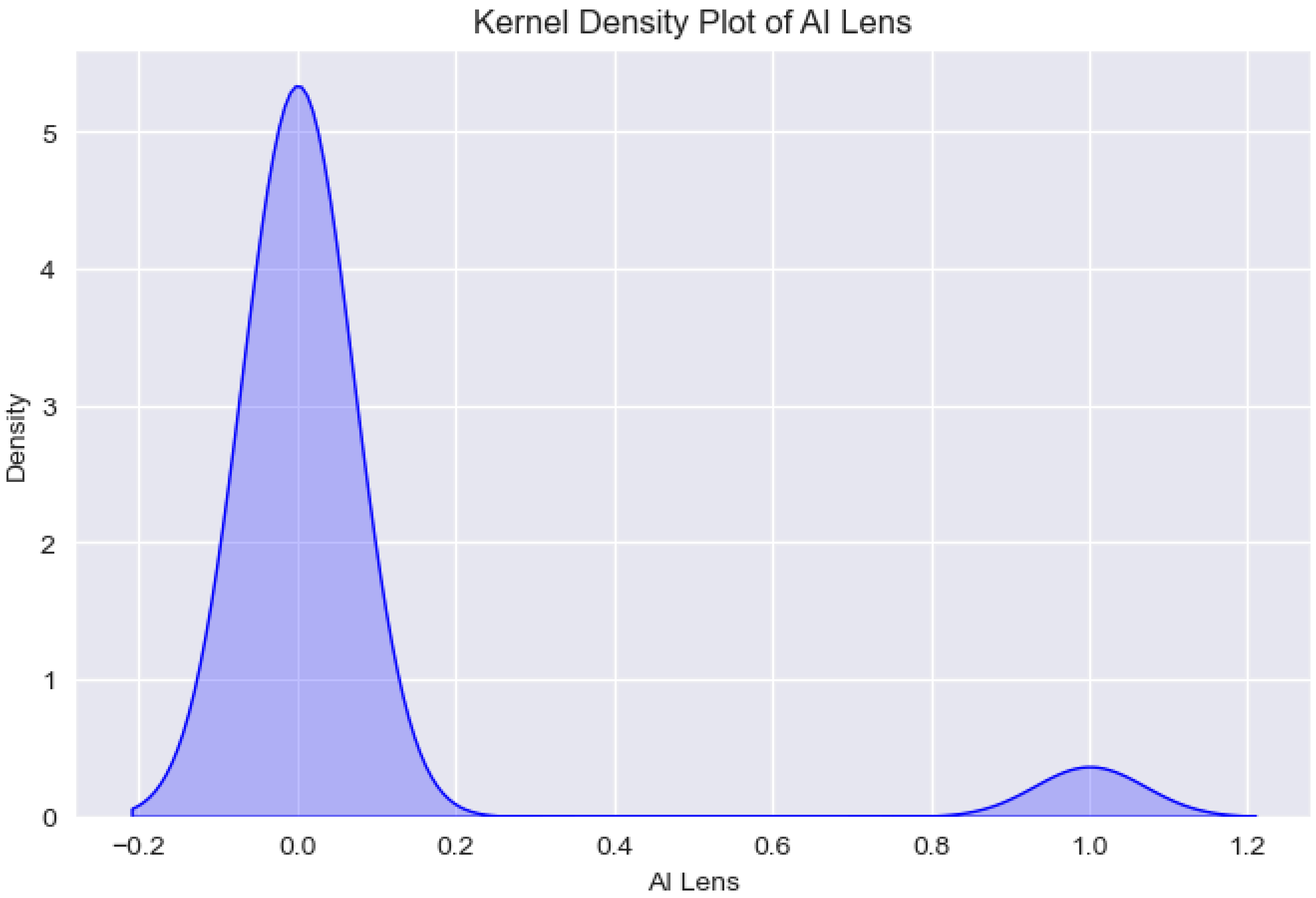
In [17]:

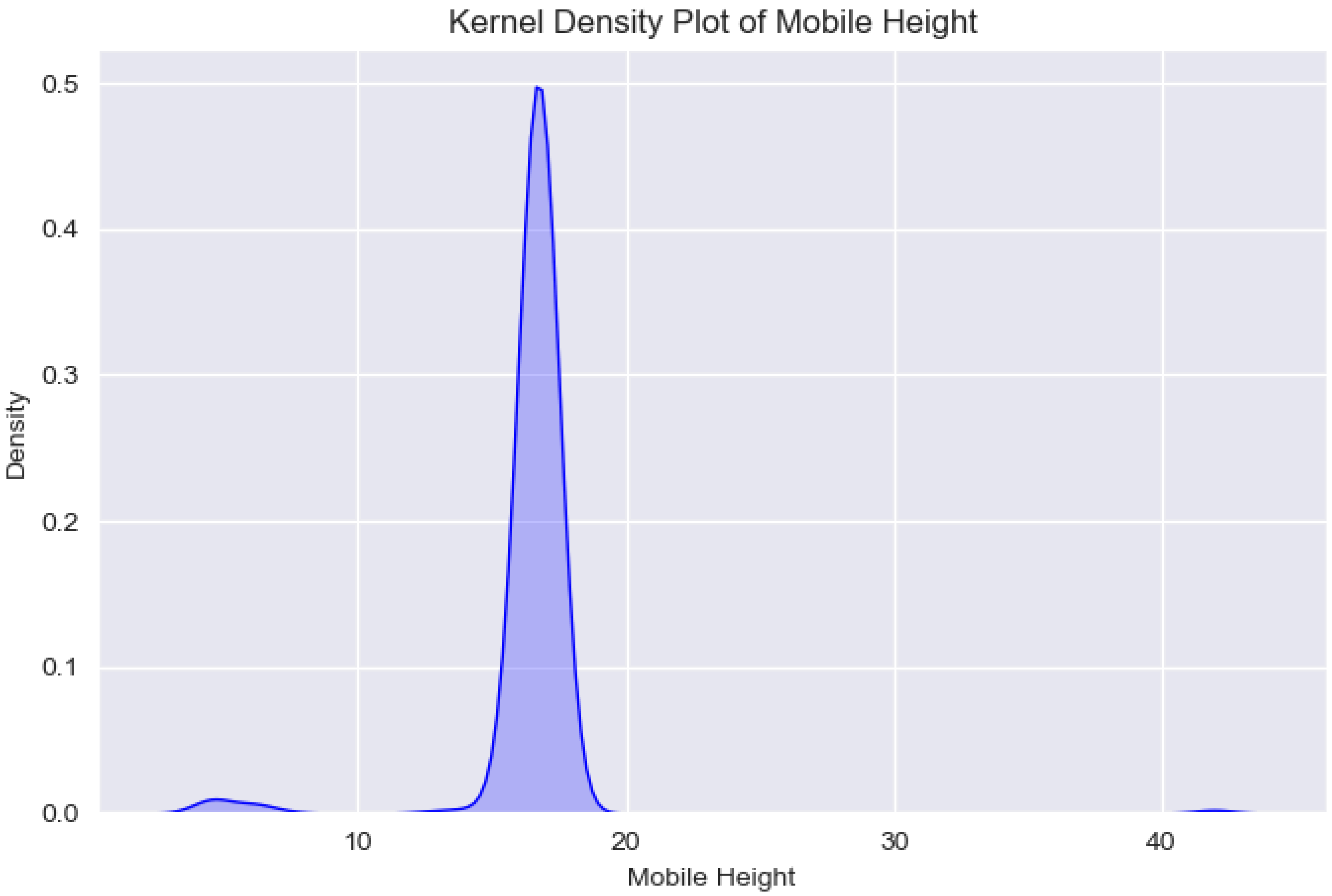
```python
# Identify numerical columns
numerical_cols = mobile_data.select_dtypes(include=["int64", "float64"]).columns

# Plot KDE for all numerical columns
for col in numerical_cols:
    plt.figure(figsize=(8, 5))
    sns.kdeplot(mobile_data[col], fill=True, color="blue")
    plt.title(f"Kernel Density Plot of {col}")
    plt.xlabel(col)
    plt.ylabel("Density")
    plt.show()
```


Kernel Density Plot of Memory
Density
0.000
0.002
0.004
0.006
0.008
0.010
−50
0
50
100
150
200
250
300
Memory


Kernel Density Plot of RAM
Density
0.25
0.20
0.15
0.10
0.05
0.00
0
2
4
6
8
10
RAM


Kernel Density Plot of Battery_
Density
0.0014
0.0012
0.0010
0.0008
0.0006
0.0004
0.0002
0.0000
0
1000
2000
3000
4000
5000
6000
7000
8000
Battery_


Kernel Density Plot of AI Lens
Density
0
1
2
3
4
5
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
AI Lens

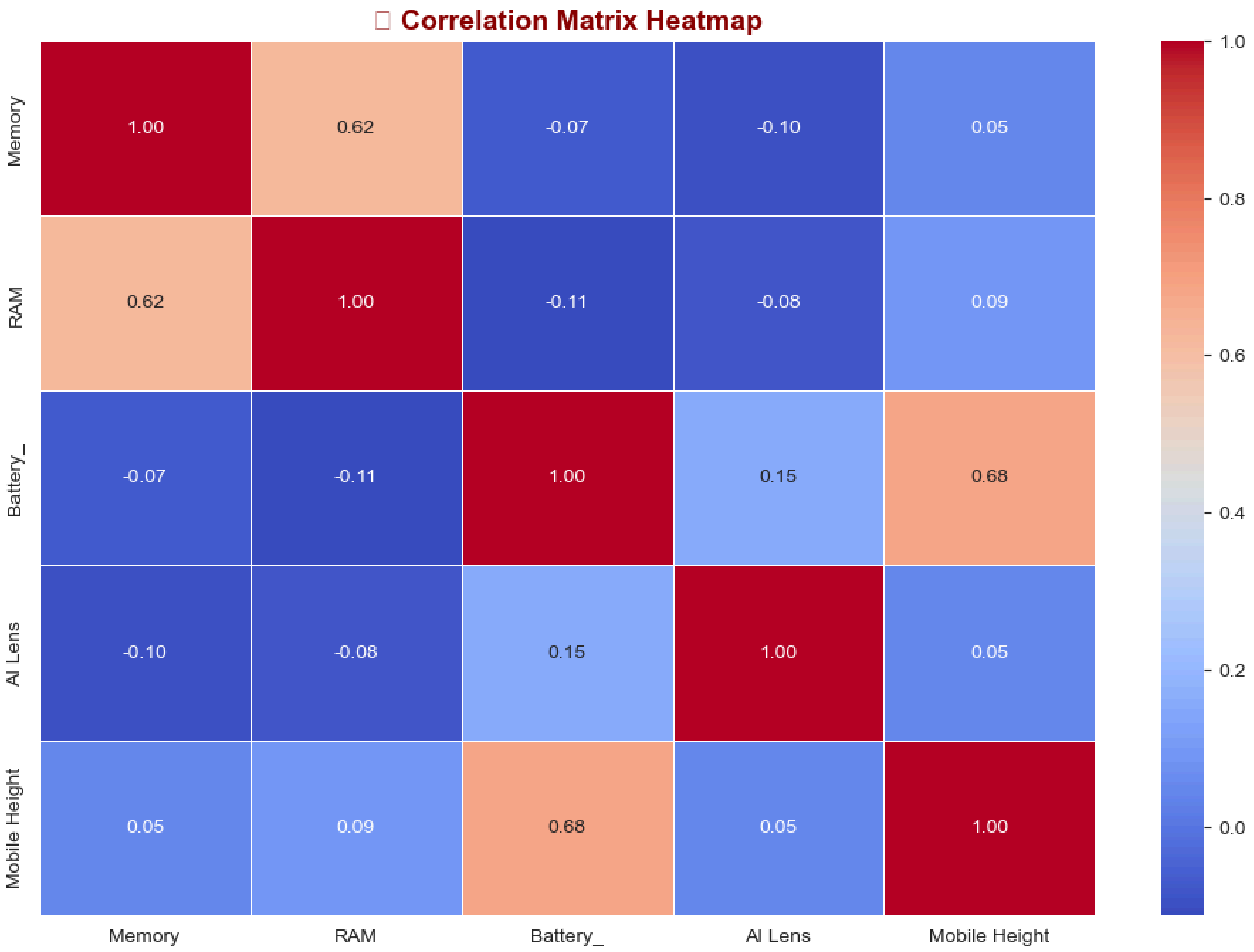


Pairplot (Scatterplot Matrix) with Categories

In [ ]:

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

# Set Seaborn style
sns.set_style("darkgrid")

# Select only numerical columns for correlation analysis
numerical_cols = mobile_data.select_dtypes(include=["int64", "float64"])
```

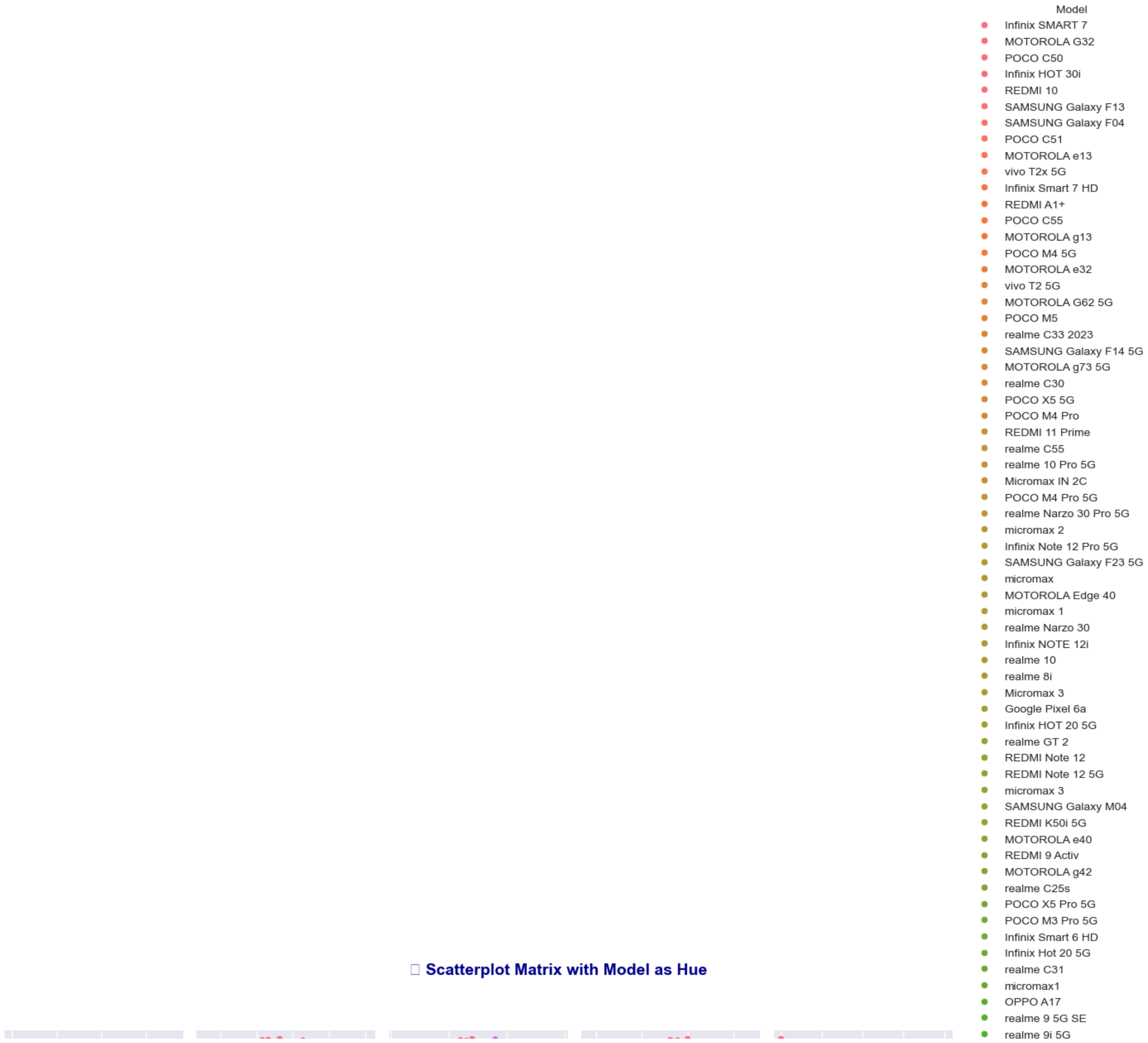
```python
# 🔥 1. Correlation Matrix Heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(numerical_cols.corr(), annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)
plt.title("🔍 Correlation Matrix Heatmap", fontsize=14, fontweight="bold", color="darkred")
plt.show()

# 🔥 2. Pairplot (Scatterplot Matrix)
pairplot_fig = sns.pairplot(numerical_cols.iloc[:, :5], diag_kind="kde", palette="husl")
pairplot_fig.fig.suptitle("📊 Scatterplot Matrix (First 5 Numerical Features)",
                          fontsize=14, fontweight="bold", color="darkblue", y=1.05)  # Adjust title position
plt.show()
```

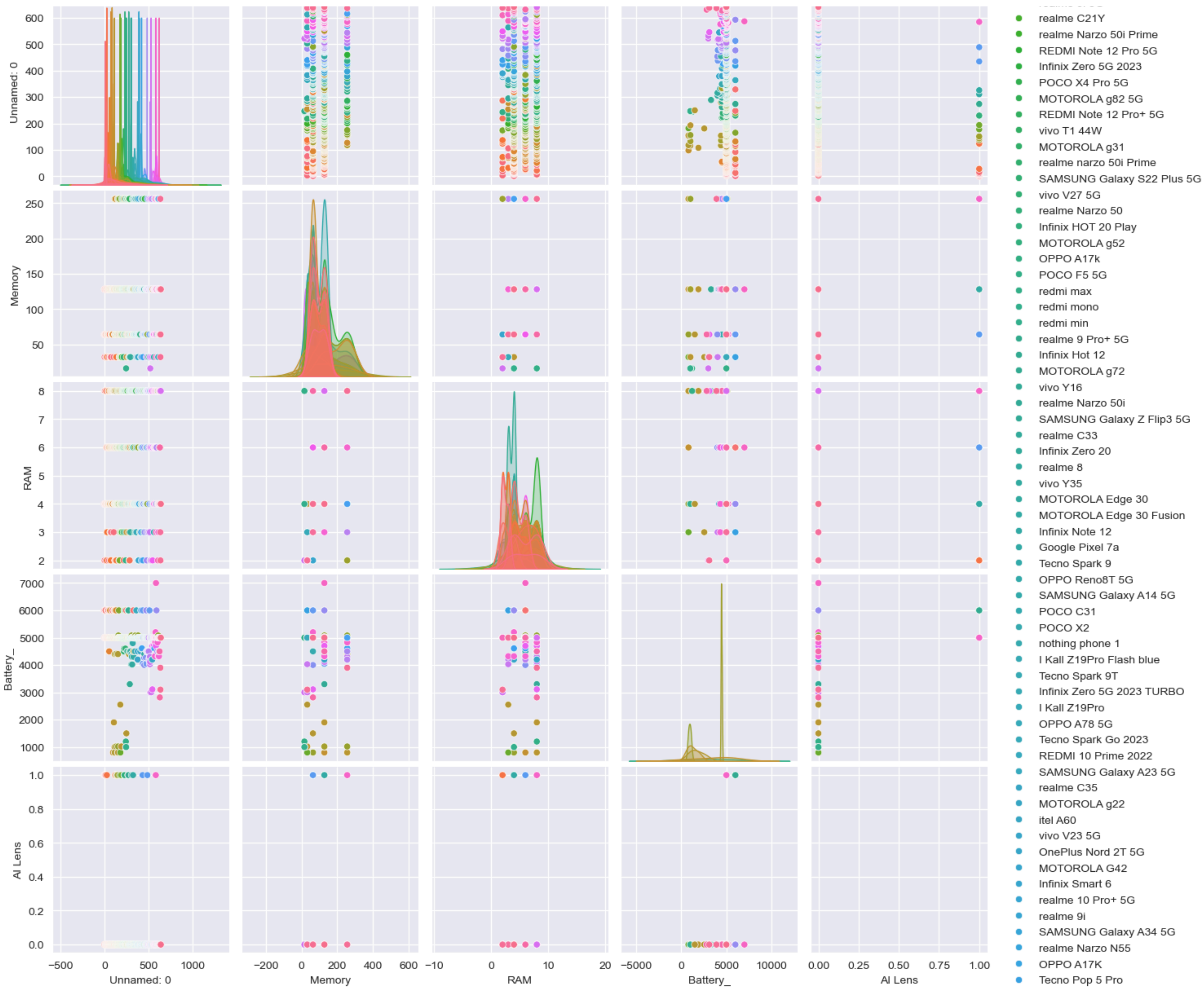


Correlation Heatmap with Categories

In [20]:

```
# Select numerical features for analysis
numerical_cols = mobile_data.select_dtypes(include=["int64", "float64"]).columns

# Create a scatterplot matrix for selected features with category hue
pairplot_fig = sns.pairplot(mobile_data, vars=numerical_cols[:5], hue="Model", diag_kind="kde", palette="husl")

# Adjust title position
pairplot_fig.fig.suptitle("📊 Scatterplot Matrix with Model as Hue", fontsize=14, fontweight="bold", color="darkblue",
plt.show()
```


Model
● Infinix SMART 7
● MOTOROLA G32
● POCO C50
● Infinix HOT 30i
● REDMI 10
● SAMSUNG Galaxy F13
● SAMSUNG Galaxy F04
● POCO C51
● MOTOROLA e13
● vivo T2x 5G
● Infinix Smart 7 HD
● REDMI A1+
● POCO C55
● MOTOROLA g13
● POCO M4 5G
● MOTOROLA e32
● vivo T2 5G
● MOTOROLA G62 5G
● POCO M5
● realme C33 2023
● SAMSUNG Galaxy F14 5G
● MOTOROLA g73 5G
● realme C30
● POCO X5 5G
● POCO M4 Pro
● REDMI 11 Prime
● realme C55
● realme 10 Pro 5G
● Micromax IN 2C
● POCO M4 Pro 5G
● realme Narzo 30 Pro 5G
● micromax 2
● Infinix Note 12 Pro 5G
● SAMSUNG Galaxy F23 5G
● micromax
● MOTOROLA Edge 40
● micromax 1
● realme Narzo 30
● Infinix NOTE 12i
● realme 10
● realme 8i
● Micromax 3
● Google Pixel 6a
● Infinix HOT 20 5G
● realme GT 2
● REDMI Note 12
● REDMI Note 12 5G
● micromax 3
● SAMSUNG Galaxy M04
● REDMI K50i 5G
● MOTOROLA e40
● REDMI 9 Activ
● MOTOROLA g42
● realme C25s
● POCO X5 Pro 5G
● POCO M3 Pro 5G
● Infinix Smart 6 HD
● Infinix Hot 20 5G
● realme C31
● micromax1
● OPPO A17
● realme 9 5G SE
● realme 9i 5G
□ Scatterplot Matrix with Model as Hue

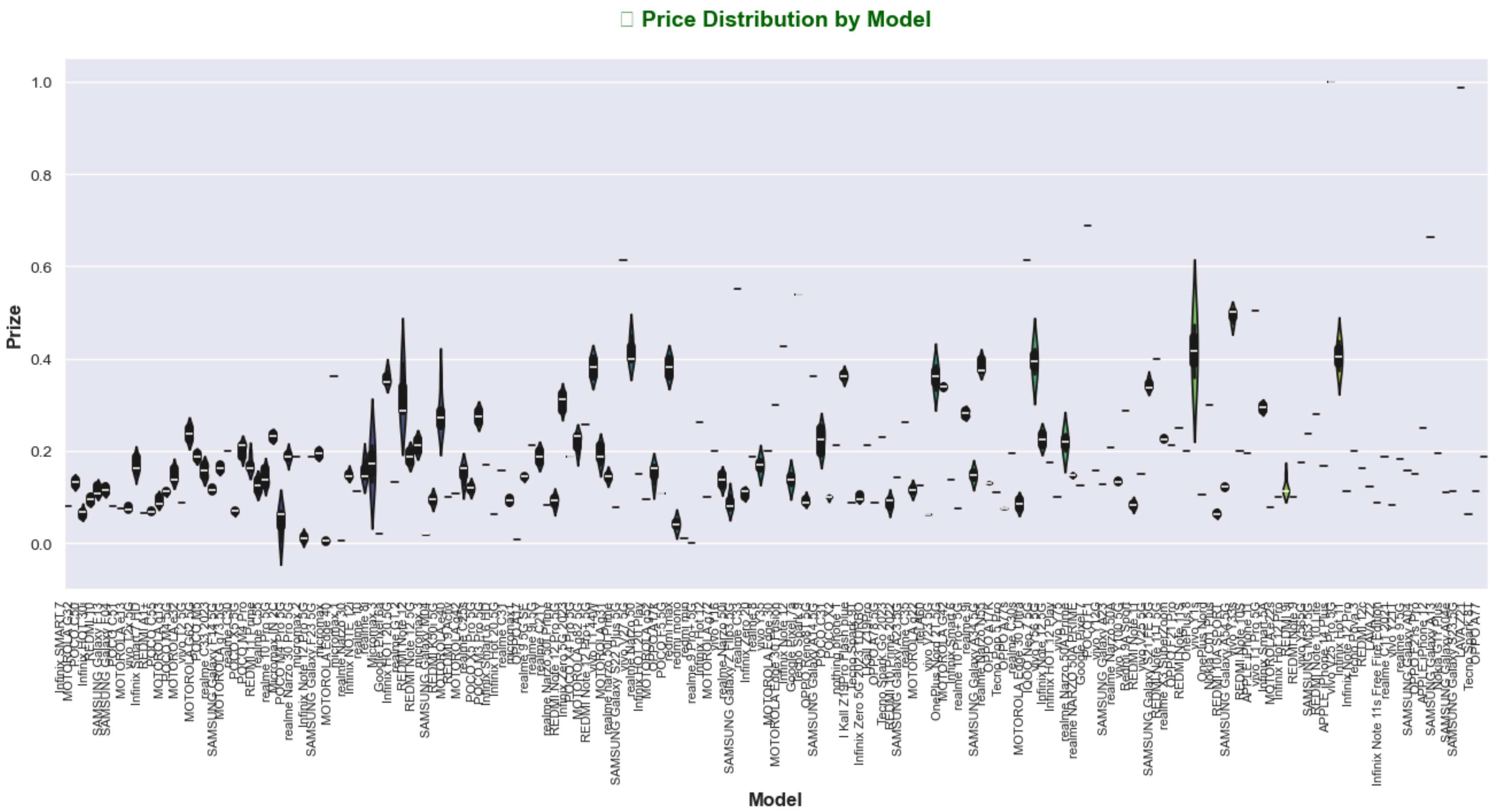

Unnamed: 0
Memory
RAM
Battery_
AI Lens
600
500
400
300
200
100
0
250
200
150
100
50
8
7
6
5
4
3
2
7000
6000
5000
4000
3000
2000
1000
1.0
0.8
0.6
0.4
0.2
0.0
−500
0
500
1000
−200
0
200
400
600
−10
0
10
20
−5000
0
5000
10000
0.00
0.25
0.50
0.75
1.00
realme C21Y
realme Narzo 50i Prime
REDMI Note 12 Pro 5G
Infinix Zero 5G 2023
POCO X4 Pro 5G
MOTOROLA g82 5G
REDMI Note 12 Pro+ 5G
vivo T1 44W
MOTOROLA g31
realme narzo 50i Prime
SAMSUNG Galaxy S22 Plus 5G
vivo V27 5G
realme Narzo 50
Infinix HOT 20 Play
MOTOROLA g52
OPPO A17k
POCO F5 5G
redmi max
redmi mono
redmi min
realme 9 Pro+ 5G
Infinix Hot 12
MOTOROLA g72
vivo Y16
realme Narzo 50i
SAMSUNG Galaxy Z Flip3 5G
realme C33
Infinix Zero 20
realme 8
vivo Y35
MOTOROLA Edge 30
MOTOROLA Edge 30 Fusion
Infinix Note 12
Google Pixel 7a
Tecno Spark 9
OPPO Reno8T 5G
SAMSUNG Galaxy A14 5G
POCO C31
POCO X2
nothing phone 1
I Kall Z19Pro Flash blue
Tecno Spark 9T
Infinix Zero 5G 2023 TURBO
I Kall Z19Pro
OPPO A78 5G
Tecno Spark Go 2023
REDMI 10 Prime 2022
SAMSUNG Galaxy A23 5G
realme C35
MOTOROLA g22
itel A60
vivo V23 5G
OnePlus Nord 2T 5G
MOTOROLA G42
Infinix Smart 6
realme 10 Pro+ 5G
realme 9i
SAMSUNG Galaxy A34 5G
realme Narzo N55
OPPO A17K
Tecno Pop 5 Pro
OPPO A77s
realme C30s
MOTOROLA Edge 30 Ultra

● IQOO Neo 7 5G
● vivo Y56 5G
● Infinix Note 12 5G
● Infinix HOT 12 Play
● vivo Y75
● realme Narzo 50A Prime
● realme NARZO 50A PRIME
● Google Pixel 7
● POCO F1
● vivo Y12G
● SAMSUNG Galaxy A23
● realme Narzo 50A
● vivo Y100 5G
● Redmi 9A Sport
● REDMI Note 11
● vivo V25 5G
● SAMSUNG Galaxy S21 FE 5G
● REDMI Note 11T 5G
● realme X3 SuperZoom
● OPPO F21 Pro
● REDMI Note 11S
● OnePlus 8
● vivo Y1s
● OnePlus Nord
● Nokia C01 Plus
● REDMI 10A SPORT
● SAMSUNG Galaxy A54 5G
● vivo Y33s
● REDMI Note 10S
● APPLE iPhone 11
● vivo T1 Pro 5G
● Infinix Smart 5A
● MOTOROLA e22s
● Infinix Hot 12 Pro
● REDMI 9i
● REDMI Note 9
● vivo Y565G
● SAMSUNG M53 5G
● REDMI Note 10 Lite
● APPLE iPhone 14 Plus
● vivo V25 Pro 5G
● Infinix Hot 11
● Infinix Note 12 Pro
● Tecno Pova 3
● REDMI 12c
● LAVA Z2
● Infinix Note 11s Free Fire Edition
● realme C11 2021
● vivo Y21T
● realme 9 5G
● SAMSUNG Galaxy A04
● OPPO F21s Pro
● APPLE iPhone 12
● SAMSUNG Galaxy A13
● Nokia G11 Plus
● SAMSUNG Galaxy A04e
● SAMSUNG Galaxy S23 5G
● LAVA Z21
● Tecno Spark 8T
● OPPO A77

Relationship Between Price and Features (Violin Plot)

In [28]:

```
plt.figure(figsize=(16, 6))  # Increase figure size

# Create the violin plot
sns.violinplot(x="Model", y="Prize", data=mobile_data, palette="viridis")

# Improve x-axis readability
plt.xticks(rotation=90, ha="right", fontsize=8)  # Rotate labels and adjust font size
plt.xlabel("Model", fontsize=12, fontweight="bold")  # X-axis label
plt.ylabel("Prize", fontsize=12, fontweight="bold")  # Y-axis label

# Title formatting
plt.title("💰 Price Distribution by Model", fontsize=14, fontweight="bold", color="darkgreen", pad=20)

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



Pairwise Feature Dependencies (FacetGrid)

In [ ]:

```
g = sns.FacetGrid(mobile_data, col="Model", col_wrap=4, height=4, aspect=1)
g.map_dataframe(sns.scatterplot, x="Battery_", y="Prize", hue="RAM", palette="corm")
g.add_legend()
plt.subplots_adjust(top=0.9)
g.fig.suptitle("🔋 Battery vs. Price Colored by RAM", fontsize=14, fontweight="bold", color="darkblue")
plt.show()
```

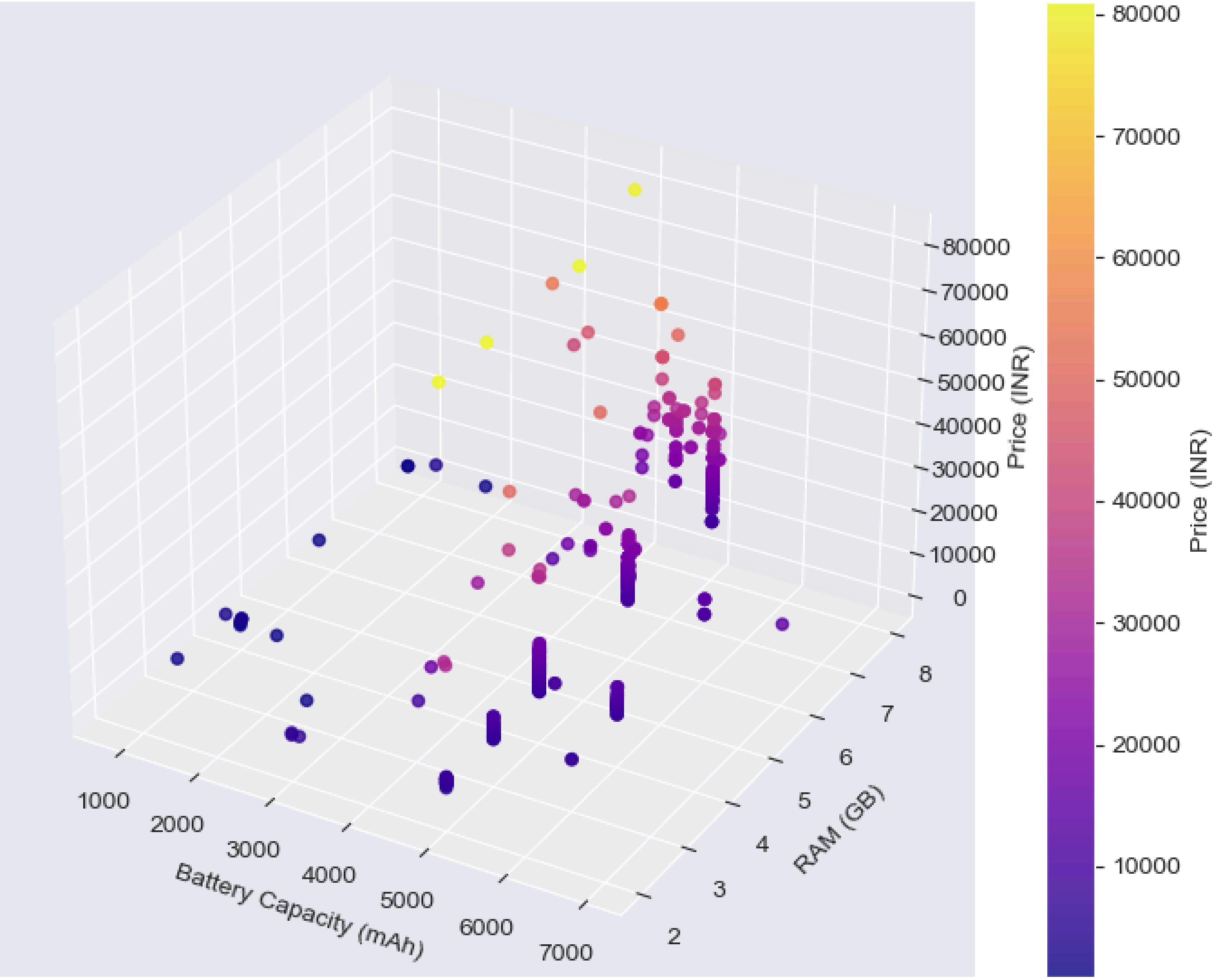
```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
File ~\anaconda3\Lib\site-packages\seaborn\palettes.py:235, in color_palette(palette, n_colors, desat, as_cmap)
    233 try:
    234     # Perhaps a named matplotlib colormap?
--> 235     palette = mpl_palette(palette, n_colors, as_cmap=as_cmap)
    236 except (ValueError, KeyError):  # Error class changed in mpl36

File ~\anaconda3\Lib\site-packages\seaborn\palettes.py:406, in mpl_palette(name, n_colors, as_cmap)
    405 else:
--> 406     cmap = get_colormap(name)
    408 if name in MPL_QUAL_PALS:

File ~\anaconda3\Lib\site-packages\seaborn\_compat.py:62, in get_colormap(name)
     61 try:
---> 62     return mpl.colormaps[name]
     63 except AttributeError:

File ~\anaconda3\Lib\site-packages\matplotlib\cm.py:93, in ColormapRegistry.__getitem__(self, item)
     92 except KeyError:
---> 93     raise KeyError(f"{item!r} is not a known colormap name") from None

KeyError: "'corm' is not a known colormap name"

During handling of the above exception, another exception occurred:

ValueError                                Traceback (most recent call last)
Cell In[56], line 2
      1 g = sns.FacetGrid(mobile_data, col="Model", col_wrap=4, height=4, aspect=1)
----> 2 g.map_dataframe(sns.scatterplot, x="Battery_", y="Prize", hue="RAM", palette="corm")
      3 g.add_legend()
      4 plt.subplots_adjust(top=0.9)

File ~\anaconda3\Lib\site-packages\seaborn\axisgrid.py:825, in FacetGrid.map_dataframe(self, func, *args, **kwargs)
    822     kwargs["data"] = data_ijk
    824     # Draw the plot
--> 825     self._facet_plot(func, ax, args, kwargs)
    827 # For axis labels, prefer to use positional args for backcompat
    828 # but also extract the x/y kwargs and use if no corresponding arg
    829 axis_labels = [kwargs.get("x", None), kwargs.get("y", None)]

File ~\anaconda3\Lib\site-packages\seaborn\axisgrid.py:854, in FacetGrid._facet_plot(self, func, ax, plot_args, plot_kwa
rgs)
    852     plot_args = []
    853     plot_kwargs["ax"] = ax
--> 854 func(*plot_args, **plot_kwargs)
    856 # Sort out the supporting information
    857 self._update_legend_data(ax)

File ~\anaconda3\Lib\site-packages\seaborn\relational.py:621, in scatterplot(data, x, y, hue, size, style, palette, hue_
order, hue_norm, sizes, size_order, size_norm, markers, style_order, legend, ax, **kwargs)
    606 def scatterplot(
    607     data=None, *,
    608     x=None, y=None, hue=None, size=None, style=None,
   (...)
    612     **kwargs
    613 ):
    615     p = _ScatterPlotter(
    616         data=data,
    617         variables=dict(x=x, y=y, hue=hue, size=size, style=style),
    618         legend=legend
    619     )
--> 621     p.map_hue(palette=palette, order=hue_order, norm=hue_norm)
    622     p.map_size(sizes=sizes, order=size_order, norm=size_norm)
    623     p.map_style(markers=markers, order=style_order)

File ~\anaconda3\Lib\site-packages\seaborn\_base.py:838, in VectorPlotter.map_hue(self, palette, order, norm, saturatio
n)
    837 def map_hue(self, palette=None, order=None, norm=None, saturation=1):
--> 838     mapping = HueMapping(self, palette, order, norm, saturation)
    839     self._hue_map = mapping

File ~\anaconda3\Lib\site-packages\seaborn\_base.py:141, in HueMapping.__init__(self, plotter, palette, order, norm, sat
uration)
    138 if map_type == "numeric":
    140     data = pd.to_numeric(data)
--> 141     levels, lookup_table, norm, cmap = self.numeric_mapping(
    142         data, palette, norm,
    143     )
    145 # --- Option 2: categorical mapping using seaborn palette
    147 elif map_type == "categorical":

File ~\anaconda3\Lib\site-packages\seaborn\_base.py:279, in HueMapping.numeric_mapping(self, data, palette, norm)
    277     cmap = palette
    278 else:
--> 279     cmap = color_palette(palette, as_cmap=True)
    281 # Now sort out the data normalization
    282 if norm is None:

File ~\anaconda3\Lib\site-packages\seaborn\palettes.py:237, in color_palette(palette, n_colors, desat, as_cmap)
    235             palette = mpl_palette(palette, n_colors, as_cmap=as_cmap)
    236         except (ValueError, KeyError):  # Error class changed in mpl36
--> 237             raise ValueError(f"{palette!r} is not a valid palette name")
    239 if desat is not None:
    240     palette = [desaturate(c, desat) for c in palette]

ValueError: 'corm' is not a valid palette name
```

3D Scatter Plot for Multivariate Dependencies

In [26]:

```python
# Remove commas and convert to numeric
mobile_data["Prize"] = mobile_data["Prize"].astype(str).str.replace(",", "").astype(float)

# Now, create the scatter plot
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection="3d")

# Scatter plot with corrected data
sc = ax.scatter(
    mobile_data["Battery_"],
    mobile_data["RAM"],
    mobile_data["Prize"],
    c=mobile_data["Prize"],
    cmap="plasma",
    alpha=0.8
)

# Labels
ax.set_xlabel("Battery Capacity (mAh)")
ax.set_ylabel("RAM (GB)")
ax.set_zlabel("Price (INR)")

# Colorbar
plt.colorbar(sc, label="Price (INR)")
```

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



In [57]:
```
# Select only numeric columns
numeric_cols = mobile_data.select_dtypes(include=['number'])

# Calculate IQR only for numeric columns
```

```python
Q1 = numeric_cols.quantile(0.25)
Q3 = numeric_cols.quantile(0.75)
IQR = Q3 - Q1

# Define outlier removal condition
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Remove outliers
mobile_data_cleaned = mobile_data[
    ~((numeric_cols < lower_bound) | (numeric_cols > upper_bound)).any(axis=1)
]

# Display the cleaned dataset
print(mobile_data_cleaned.shape)


# Convert `Prize` to Numeric
mobile_data["Prize"] = mobile_data["Prize"].astype(str).str.replace(",", "").astype(float)




# Rename the Rear Camera and Front Camera column
mobile_data.rename(columns={"Rear Camera": "Rear Camera [MP]"}, inplace=True)
mobile_data.rename(columns={"Front Camera": "Front Camera [MP]"}, inplace=True)

# Display updated column names
print(mobile_data.columns)

# 4. Convert Camera Columns to Numeric
mobile_data["Rear Camera [MP]"] = mobile_data["Rear Camera [MP]"].str.replace("MP", "").astype(float)
mobile_data["Front Camera [MP]"] = mobile_data["Front Camera [MP]"].str.replace("MP", "").astype(float)

# 6. Normalize Numeric Features
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
numeric_cols = ["Memory", "RAM", "Battery_", "Rear Camera [MP]", "Front Camera [MP]", "Mobile Height", "Prize"]
mobile_data[numeric_cols] = scaler.fit_transform(mobile_data[numeric_cols])

# Display the cleaned dataset
mobile_data.head()
```

```
(332, 11)
Index(['Model', 'Colour', 'Memory', 'RAM', 'Battery_', 'Rear Camera [MP]',
       'Front Camera [MP]', 'AI Lens', 'Mobile Height', 'Processor_', 'Prize'],
      dtype='object')
```

Out[57]:

	Model	Colour	Memory	RAM	Battery_	Rear Camera [MP]	Front Camera [MP]	AI Lens	Mobile Height	Processor_	Prize
0	Infinix SMART 7	Night Black	0.200000	0.333333	0.838710	0.065	0.083333	1	0.327457	Unisoc Spreadtrum SC9863A1	0.079659
1	Infinix SMART 7	Azure Blue	0.200000	0.333333	0.838710	0.065	0.083333	1	0.327457	Unisoc Spreadtrum SC9863A1	0.079659
2	MOTOROLA G32	Mineral Gray	0.466667	1.000000	0.677419	0.250	0.266667	0	0.324252	Qualcomm Snapdragon 680	0.138351
3	POCO C50	Royal Blue	0.066667	0.000000	0.677419	0.040	0.083333	0	0.322115	Mediatek Helio A22	0.059054
4	Infinix HOT 30i	Marigold	0.466667	1.000000	0.677419	0.250	0.083333	1	0.327457	G37	0.100888

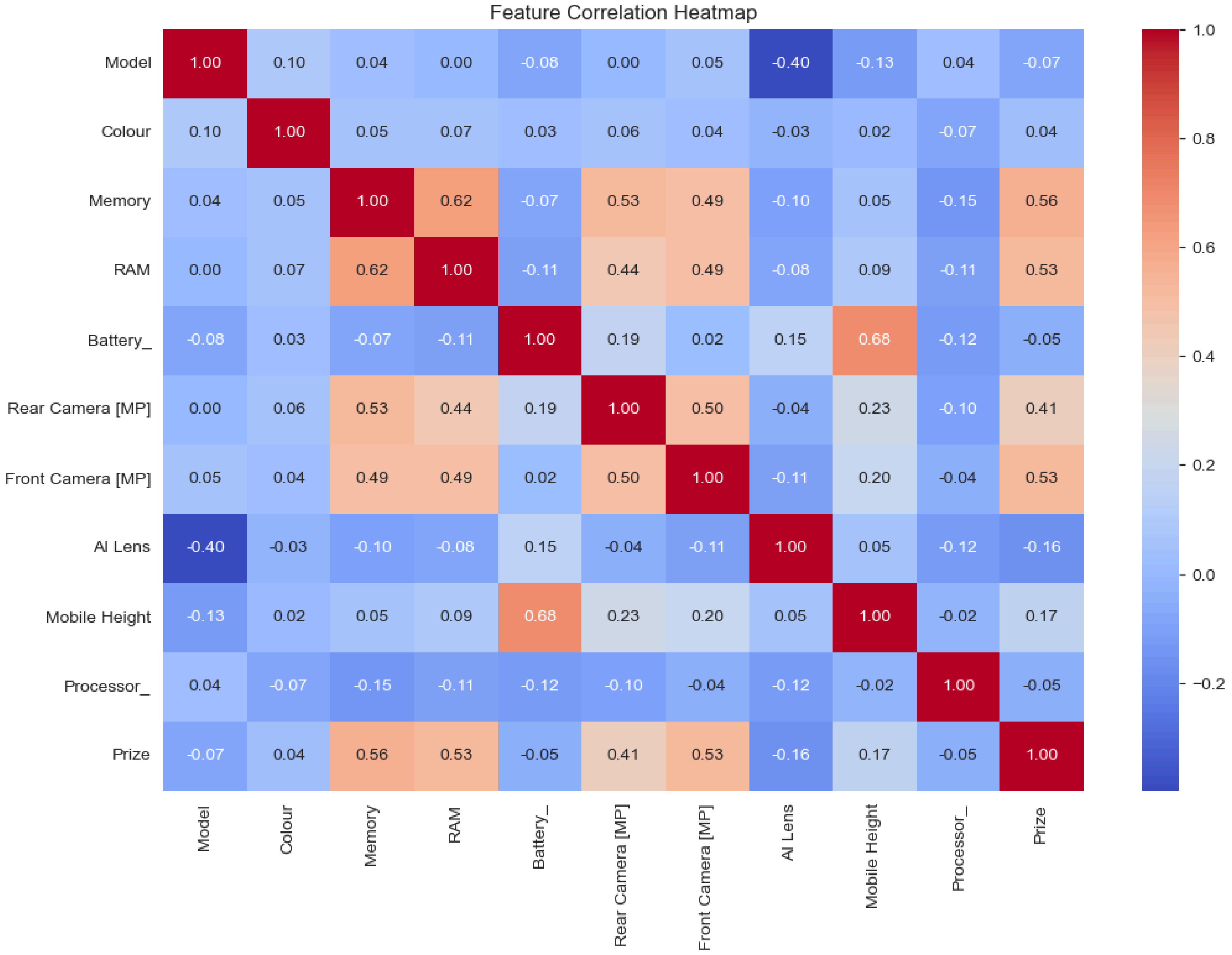
In [59]:

```python
from sklearn.preprocessing import LabelEncoder

# List of categorical columns
categorical_cols = ["Model", "Colour", "Processor_"]

# Apply Label Encoding to each categorical column
label_encoders = {}

for col in categorical_cols:
    le = LabelEncoder()
    mobile_data[col] = le.fit_transform(mobile_data[col])
    label_encoders[col] = le  # Save encoders for later decoding if needed

# Display the first few rows
print(mobile_data.head())
```

```
   Model  Colour    Memory       RAM  Battery_  Rear Camera [MP]  \
0     23     159  0.200000  0.333333  0.838710             0.065   
1     23      20  0.200000  0.333333  0.838710             0.065   
2     37     149  0.466667  1.000000  0.677419             0.250   
3     69     201  0.066667  0.000000  0.677419             0.040   
4     12     130  0.466667  1.000000  0.677419             0.250   

   Front Camera [MP]  AI Lens  Mobile Height  Processor_     Prize  
0           0.083333        1       0.327457         113  0.079659  
1           0.083333        1       0.327457         113  0.079659  
2           0.266667        0       0.324252          75  0.138351  
3           0.083333        0       0.322115          56  0.059054  
4           0.083333        1       0.327457          14  0.100888  
```

In [32]:

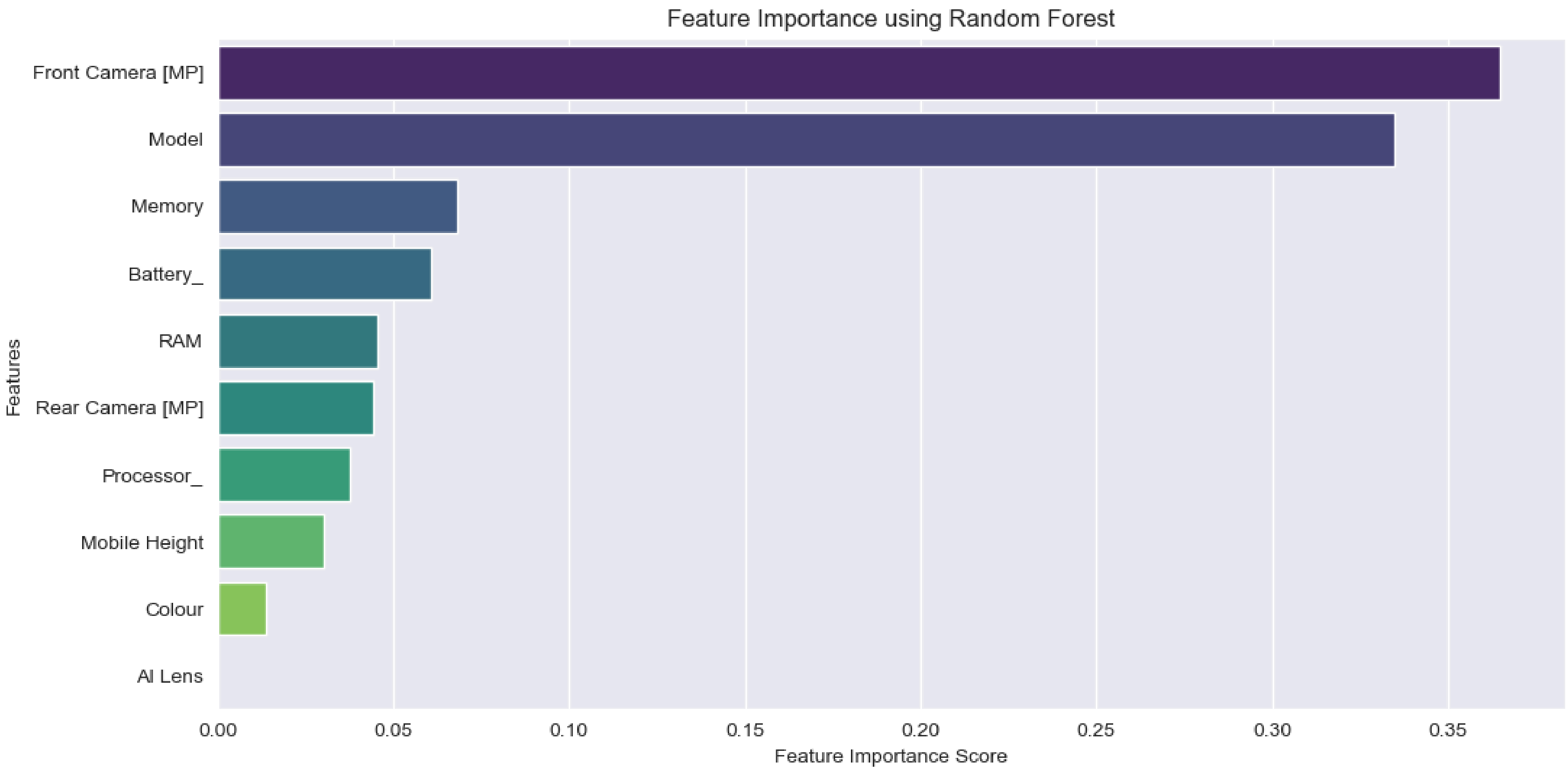
```python
# Compute correlation matrix
correlation_matrix = mobile_data.corr()

# Plot correlation heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Feature Correlation Heatmap")
plt.show()

# Find top correlated features
corr_with_price = correlation_matrix["Prize"].abs().sort_values(ascending=False)
print("Top Correlated Features with Price:\n", corr_with_price)
```


Feature Correlation Heatmap
Model 1.00 0.10 0.04 0.00 -0.08 0.00 0.05 -0.40 -0.13 0.04 -0.07
Colour 0.10 1.00 0.05 0.07 0.03 0.06 0.04 -0.03 0.02 -0.07 0.04
Memory 0.04 0.05 1.00 0.62 -0.07 0.53 0.49 -0.10 0.05 -0.15 0.56
RAM 0.00 0.07 0.62 1.00 -0.11 0.44 0.49 -0.08 0.09 -0.11 0.53
Battery_ -0.08 0.03 -0.07 -0.11 1.00 0.19 0.02 0.15 0.68 -0.12 -0.05
Rear Camera [MP] 0.00 0.06 0.53 0.44 0.19 1.00 0.50 -0.04 0.23 -0.10 0.41
Front Camera [MP] 0.05 0.04 0.49 0.49 0.02 0.50 1.00 -0.11 0.20 -0.04 0.53
AI Lens -0.40 -0.03 -0.10 -0.08 0.15 -0.04 -0.11 1.00 0.05 -0.12 -0.16
Mobile Height -0.13 0.02 0.05 0.09 0.68 0.23 0.20 0.05 1.00 -0.02 0.17
Processor_ 0.04 -0.07 -0.15 -0.11 -0.12 -0.10 -0.04 -0.12 -0.02 1.00 -0.05
Prize -0.07 0.04 0.56 0.53 -0.05 0.41 0.53 -0.16 0.17 -0.05 1.00
Model
Colour
Memory
RAM
Battery_
Rear Camera [MP]
Front Camera [MP]
AI Lens
Mobile Height
Processor_
Prize
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

```
Top Correlated Features with Price:
 Prize                1.000000
Memory               0.563535
RAM                  0.529474
Front Camera [MP]    0.529013
Rear Camera [MP]     0.406784
Mobile Height        0.168303
AI Lens              0.156336
Model                0.073833
Processor_           0.049600
Battery_             0.046250
Colour               0.040595
Name: Prize, dtype: float64
```

In [61]:

```python
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split

# Define X (features) and y (target variable)
X = mobile_data.drop(columns=["Prize"])  # Remove target column
y = mobile_data["Prize"]

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

# Train a Random Forest model
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Get feature importances
feature_importances = pd.Series(rf_model.feature_importances_, index=X.columns)
feature_importances = feature_importances.sort_values(ascending=False)

# Plot feature importance
plt.figure(figsize=(12, 6))
sns.barplot(x=feature_importances, y=feature_importances.index, palette="viridis")
plt.xlabel("Feature Importance Score")
plt.ylabel("Features")
plt.title("Feature Importance using Random Forest")
plt.show()

# Print top important features
print("Top Important Features:\n", feature_importances)
```

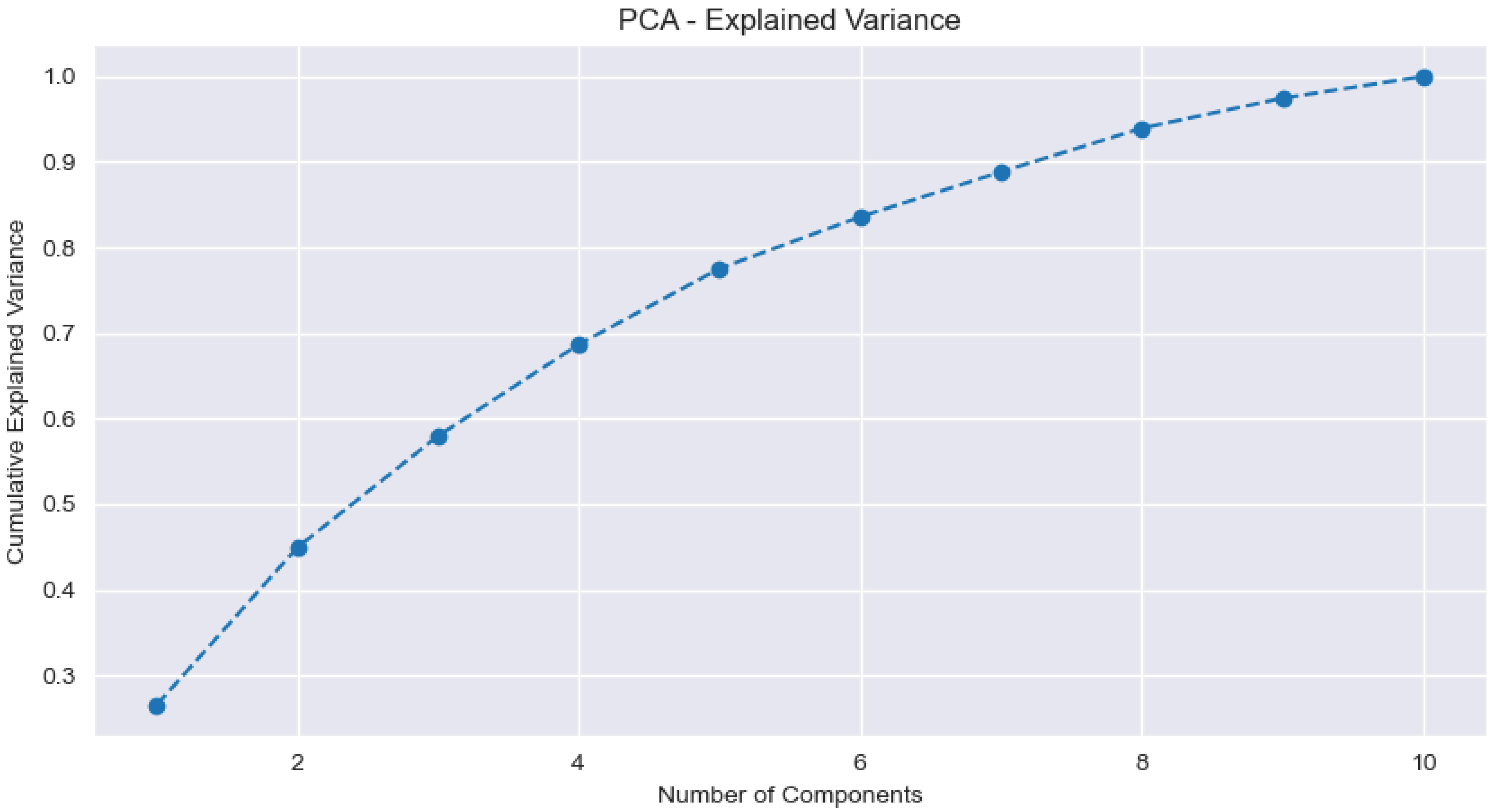


```
Top Important Features:
 Front Camera [MP]    0.365029
Model                0.334752
Memory               0.068347
Battery_             0.060567
RAM                  0.045294
Rear Camera [MP]     0.044153
Processor_           0.037650
Mobile Height        0.030405
Colour               0.013591
AI Lens              0.000211
dtype: float64
```

In [63]:

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Standardize the features
```

```python
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply PCA
pca = PCA(n_components=10)  # Reduce to 10 components
X_pca = pca.fit_transform(X_scaled)

# Explained variance ratio
plt.figure(figsize=(10, 5))
plt.plot(range(1, 11), np.cumsum(pca.explained_variance_ratio_), marker="o", linestyle="--")
plt.xlabel("Number of Components")
plt.ylabel("Cumulative Explained Variance")
plt.title("PCA - Explained Variance")
plt.show()
```



```
In [65]: selected_features = feature_importances[:25].index  # Keep top 25 features
         X_selected = mobile_data[selected_features]
         print("Final Selected Features:\n", X_selected.columns)
```

```
Final Selected Features:
 Index(['Front Camera [MP]', 'Model', 'Memory', 'Battery_', 'RAM',
       'Rear Camera [MP]', 'Processor_', 'Mobile Height', 'Colour', 'AI Lens'],
      dtype='object')
```

```
In [54]: # Define target variable (Price)
         y = mobile_data["Prize"]

         # Use only the selected features from the previous step
         selected_features = ["RAM", "Memory", "Battery_", "Rear Camera [MP]", "Front Camera [MP]",
                              "Mobile Height", "AI Lens", "Processor_","Model","Colour" ]  # Example selected features
         X = mobile_data[selected_features]

         # Train-test split (80% training, 20% testing)
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

         print("Training Set:", X_train.shape)
         print("Testing Set:", X_test.shape)
```

```
Training Set: (424, 10)
Testing Set: (107, 10)
```

In [67]:

```
# Train Linear Regression Model
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)

# Predictions
y_pred_lr = lr_model.predict(X_test)
```

In [69]:

```
# Train Decision Tree Model
dt_model = DecisionTreeRegressor(random_state=42)
dt_model.fit(X_train, y_train)

# Predictions
y_pred_dt = dt_model.predict(X_test)
```

In [71]:

```
# Train Random Forest Model
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Predictions
y_pred_rf = rf_model.predict(X_test)
```

In [73]:

```
# 📌 Evaluation Function
def evaluate_model(y_test, y_pred, model_name):
    mae = mean_absolute_error(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)  # Root Mean Squared Error
    r2 = r2_score(y_test, y_pred)

    print(f"📊 {model_name} Performance:")
    print(f"   - Mean Absolute Error (MAE): {mae:.2f}")
    print(f"   - Mean Squared Error (MSE): {mse:.2f}")
    print(f"   - Root Mean Squared Error (RMSE): {rmse:.2f}")
    print(f"   - R² Score: {r2:.4f}\n")

# 📌 Evaluate all three models
evaluate_model(y_test, y_pred_lr, "Linear Regression")
evaluate_model(y_test, y_pred_dt, "Decision Tree")
evaluate_model(y_test, y_pred_rf, "Random Forest")

# 📌 Show actual vs predicted prices (for first 10 values)
predictions_df = pd.DataFrame({
    "Actual Price": y_test.values,
    "Predicted (LR)": y_pred_lr,
    "Predicted (DT)": y_pred_dt,
    "Predicted (RF)": y_pred_rf
})

print("\n📌 Actual vs Predicted Prices (First 10 values):")
print(predictions_df.head(10))
```

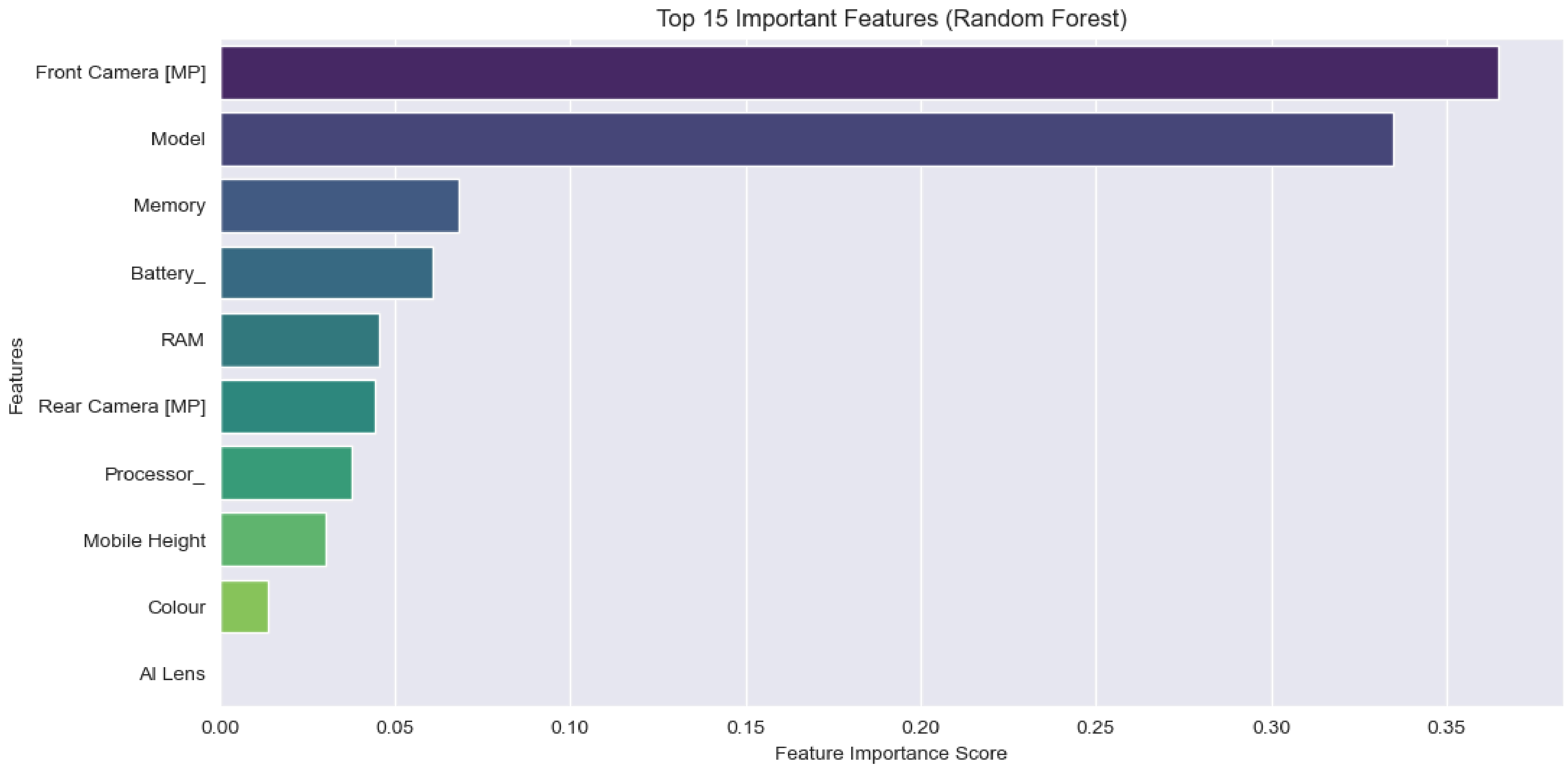
```
📊 Linear Regression Performance:
  - Mean Absolute Error (MAE): 0.06
  - Mean Squared Error (MSE): 0.01
  - Root Mean Squared Error (RMSE): 0.11
  - R² Score: 0.2672

📊 Decision Tree Performance:
  - Mean Absolute Error (MAE): 0.03
  - Mean Squared Error (MSE): 0.01
  - Root Mean Squared Error (RMSE): 0.12
  - R² Score: 0.1252

📊 Random Forest Performance:
  - Mean Absolute Error (MAE): 0.03
  - Mean Squared Error (MSE): 0.01
  - Root Mean Squared Error (RMSE): 0.09
  - R² Score: 0.5565


📌 Actual vs Predicted Prices (First 10 values):
  Actual Price  Predicted (LR)  Predicted (DT)  Predicted (RF)
0     0.288203        0.233110        0.288203        0.297918
1     0.100888        0.179224        0.100888        0.102635
2     0.537956        0.297063        0.537956        0.543063
3     0.094644        0.092097        0.097766        0.091409
4     0.113376        0.195817        0.113376        0.111153
5     0.113376        0.095531        0.113376        0.114536
6     0.169570        0.209377        0.169570        0.174988
7     0.188302        0.221359        0.144595        0.207124
8     0.213277        0.228175        0.213277        0.243667
9     0.075913        0.048842        0.075913        0.073430
```

In [75]:

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

# Get feature importance from Decision Tree
dt_importance = dt_model.feature_importances_

# Get feature importance from Random Forest
rf_importance = rf_model.feature_importances_

# Convert to DataFrame for visualization
```

```
feature_importance_df = pd.DataFrame({
    "Feature": X.columns,
    "Decision Tree Importance": dt_importance,
    "Random Forest Importance": rf_importance
})

# Sort by importance (Random Forest)
feature_importance_df = feature_importance_df.sort_values(by="Random Forest Importance", ascending=False)

# 📌 Print Top Features
print("📊 Top Features based on Random Forest:")
print(feature_importance_df.head(10))

# 📌 Visualization of Feature Importance
plt.figure(figsize=(12, 6))
sns.barplot(x=feature_importance_df["Random Forest Importance"][:15],
            y=feature_importance_df["Feature"][:15],
            palette="viridis")

plt.xlabel("Feature Importance Score")
plt.ylabel("Features")
plt.title("Top 15 Important Features (Random Forest)")
plt.show()
```

```
📊 Top Features based on Random Forest:
            Feature  Decision Tree Importance  Random Forest Importance
6  Front Camera [MP]                  0.323210                  0.365029
0              Model                  0.385938                  0.334752
2             Memory                  0.075037                  0.068347
4           Battery_                  0.043459                  0.060567
3                RAM                  0.066478                  0.045294
5   Rear Camera [MP]                  0.032285                  0.044153
9         Processor_                  0.025824                  0.037650
8      Mobile Height                  0.022389                  0.030405
1             Colour                  0.025364                  0.013591
7            AI Lens                  0.000015                  0.000211
```



In [91]:

```
from pptx import Presentation
from pptx.util import Inches

# Create a PowerPoint Presentation
prs = Presentation()

# Add Title Slide
slide_layout = prs.slide_layouts[0]
slide = prs.slides.add_slide(slide_layout)
title = slide.shapes.title
subtitle = slide.placeholders[1]
title.text = "Mobile Phone Price Prediction"
subtitle.text = "By Your Name | Date"

# Add an Introduction Slide
slide_layout = prs.slide_layouts[1]
```

```
slide = prs.slides.add_slide(slide_layout)
title = slide.shapes.title
content = slide.placeholders[1]
title.text = "Introduction"
content.text = "Goal: Predict mobile phone prices using Machine Learning.\nTechniques Used: Data Cleaning, Feature Sele

# Save the PowerPoint
prs.save("Mobile_Price_Prediction_Report.pptx")
print("✅ PowerPoint Created Successfully!")
```

```
✅ PowerPoint Created Successfully!
```

In [83]:

```
!pip install python-pptx
```

```
Collecting python-pptx
  Downloading python_pptx-1.0.2-py3-none-any.whl.metadata (2.5 kB)
Requirement already satisfied: Pillow>=3.3.2 in c:\users\user\anaconda3\lib\site-packages (from python-pptx) (10.3.0)
Collecting XlsxWriter>=0.5.7 (from python-pptx)
  Downloading XlsxWriter-3.2.2-py3-none-any.whl.metadata (2.8 kB)
Requirement already satisfied: lxml>=3.1.0 in c:\users\user\anaconda3\lib\site-packages (from python-pptx) (5.2.1)
Requirement already satisfied: typing-extensions>=4.9.0 in c:\users\user\anaconda3\lib\site-packages (from python-pptx)
(4.11.0)
Downloading python_pptx-1.0.2-py3-none-any.whl (472 kB)
   ---------------------------------------- 0.0/472.8 kB ? eta -:--:--
   ---------------------------------------- 0.0/472.8 kB ? eta -:--:--
    --------------------------------------- 10.2/472.8 kB ? eta -:--:--
   -- ------------------------------------ 30.7/472.8 kB 325.1 kB/s eta 0:00:02
   --- ----------------------------------- 41.0/472.8 kB 279.3 kB/s eta 0:00:02
   ----- --------------------------------- 61.4/472.8 kB 363.1 kB/s eta 0:00:02
   ----------- -------------------------- 143.4/472.8 kB 607.9 kB/s eta 0:00:01
   ----------- -------------------------- 143.4/472.8 kB 607.9 kB/s eta 0:00:01
   ------------------ ------------------- 225.3/472.8 kB 655.6 kB/s eta 0:00:01
   ------------------ ------------------- 225.3/472.8 kB 655.6 kB/s eta 0:00:01
   ------------------------ ------------- 307.2/472.8 kB 731.4 kB/s eta 0:00:01
   ------------------------ ------------- 307.2/472.8 kB 731.4 kB/s eta 0:00:01
   --------------------------- ---------- 337.9/472.8 kB 655.4 kB/s eta 0:00:01
   ------------------------------- ------ 389.1/472.8 kB 713.5 kB/s eta 0:00:01
   -------------------------------- ----- 399.4/472.8 kB 637.9 kB/s eta 0:00:01
   --------------------------------- ---- 419.8/472.8 kB 624.4 kB/s eta 0:00:01
   ------------------------------------ - 450.6/472.8 kB 655.2 kB/s eta 0:00:01
   ------------------------------------ - 450.6/472.8 kB 655.2 kB/s eta 0:00:01
   -------------------------------------- 472.8/472.8 kB 603.8 kB/s eta 0:00:00
Downloading XlsxWriter-3.2.2-py3-none-any.whl (165 kB)
   ---------------------------------------- 0.0/165.1 kB ? eta -:--:--
   -- ------------------------------------- 10.2/165.1 kB ? eta -:--:--
   --------- ----------------------------- 41.0/165.1 kB 393.8 kB/s eta 0:00:01
   ---------------------------------------  163.8/165.1 kB 1.2 MB/s eta 0:00:01
   ---------------------------------------- 165.1/165.1 kB 1.1 MB/s eta 0:00:00
Installing collected packages: XlsxWriter, python-pptx
Successfully installed XlsxWriter-3.2.2 python-pptx-1.0.2
```

In [93]:

```python
import joblib

# Save the trained Random Forest model
joblib.dump(rf_model, "random_forest_mobile_price.pkl")
```

```
print("Model saved successfully!")
```

```
Model saved successfully!
```

In [95]:

```
# Load the saved model
rf_model_loaded = joblib.load("random_forest_mobile_price.pkl")

print("Model loaded successfully!")
```

```
Model loaded successfully!
```

In [111]:

```
# Ensure new_data has the same features as the training data
new_data.columns = X_train.columns
print(new_data.columns)
new_data = pd.DataFrame([[23, 20, 0.200000, 0.333333, 0.838710, 0.065, 0.083333, 1, 0.327457, 113]])  # Example data

# Predict the price using the loaded model
predicted_price = rf_model_loaded.predict(new_data)

print("Predicted Mobile Price:", predicted_price[0])
```

```
Index(['Model', 'Colour', 'Memory', 'RAM', 'Battery_', 'Rear Camera [MP]',
       'Front Camera [MP]', 'AI Lens', 'Mobile Height', 'Processor_'],
      dtype='object')
Predicted Mobile Price: 0.08264601206308753
```

In [99]:

```
mobile_data
```

Out[99]:

	Model	Colour	Memory	RAM	Battery_	Rear Camera [MP]	Front Camera [MP]	AI Lens	Mobile Height	Processor_	Prize
0	23	159	0.200000	0.333333	0.838710	0.065	0.083333	1	0.327457	113	0.079659
1	23	20	0.200000	0.333333	0.838710	0.065	0.083333	1	0.327457	113	0.079659
2	37	149	0.466667	1.000000	0.677419	0.250	0.266667	0	0.324252	75	0.138351
3	69	201	0.066667	0.000000	0.677419	0.040	0.083333	0	0.322115	56	0.059054
4	12	130	0.466667	1.000000	0.677419	0.250	0.083333	1	0.327457	14	0.100888
...	...	...	...	...	...	...	...	...	...	...	...
536	118	49	1.000000	1.000000	0.500000	0.250	0.200000	0	0.293536	89	0.987512
537	32	52	0.066667	0.000000	0.370968	0.025	0.033333	0	0.219017	68	0.063412
538	123	259	0.200000	0.333333	0.677419	0.250	0.133333	0	0.327457	35	0.113263
539	110	17	0.466667	1.000000	0.677419	0.250	0.533333	0	0.314103	11	0.475518
540	59	215	0.466667	0.333333	0.677419	0.250	0.133333	0	0.324786	57	0.188302

531 rows × 11 columns

In [113]:

```
actual_price = mobile_data["Prize"][0]  # Assuming this is the actual price
predicted_price = predicted_price[0]

error = actual_price - predicted_price
print(f"Actual Price: {actual_price}")
print(f"Predicted Price: {predicted_price}")
print(f"Prediction Error: {error}")
```

```
Actual Price: 0.07965883689856267
Predicted Price: 0.08264601206308753
Prediction Error: -0.0029871751645248606
```

In [105]:

```
print(X_train.columns)
```

```
Index(['Model', 'Colour', 'Memory', 'RAM', 'Battery_', 'Rear Camera [MP]',
       'Front Camera [MP]', 'AI Lens', 'Mobile Height', 'Processor_'],
      dtype='object')
```

In [117]:

```python
# Import necessary libraries
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error

# Initialize the Gradient Boosting model
gb_model = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth=3, random_state=42)

# Train the model
gb_model.fit(X_train, y_train)

# Predict on test data
y_pred_gb = gb_model.predict(X_test)

# Print first 10 predicted vs actual values
print("\n📊 Actual vs Predicted Prices (Gradient Boosting - First 10 values):")
print(pd.DataFrame({
    "Actual Price": y_test[:10].values,
    "Predicted (GB)": y_pred_gb[:10]
}))

# Evaluate the model
mae = mean_absolute_error(y_test, y_pred_gb)
rmse = mean_squared_error(y_test, y_pred_gb, squared=False)

print("\n📉 Model Evaluation (Gradient Boosting Regressor):")
print(f"✅ Mean Absolute Error (MAE): {mae:.4f}")
print(f"✅ Root Mean Squared Error (RMSE): {rmse:.4f}")
```

```
📊 Actual vs Predicted Prices (Gradient Boosting - First 10 values):
   Actual Price  Predicted (GB)
0      0.288203        0.286618
1      0.100888        0.111670
2      0.537956        0.537126
3      0.094644        0.079139
4      0.113376        0.117566
5      0.113376        0.105173
6      0.169570        0.166106
7      0.188302        0.213113
8      0.213277        0.254080
9      0.075913        0.074726

📉 Model Evaluation (Gradient Boosting Regressor):
✅ Mean Absolute Error (MAE): 0.0347
✅ Root Mean Squared Error (RMSE): 0.0965
```

In [119]:

```python
dt_model = DecisionTreeRegressor(max_depth=10, min_samples_split=5, min_samples_leaf=2, random_state=42)
dt_model.fit(X_train, y_train)
```

Out[119]:

```
▾                         DecisionTreeRegressor

DecisionTreeRegressor(max_depth=10, min_samples_leaf=2, min_samples_split=5,
                      random_state=42)
```

In [121]:

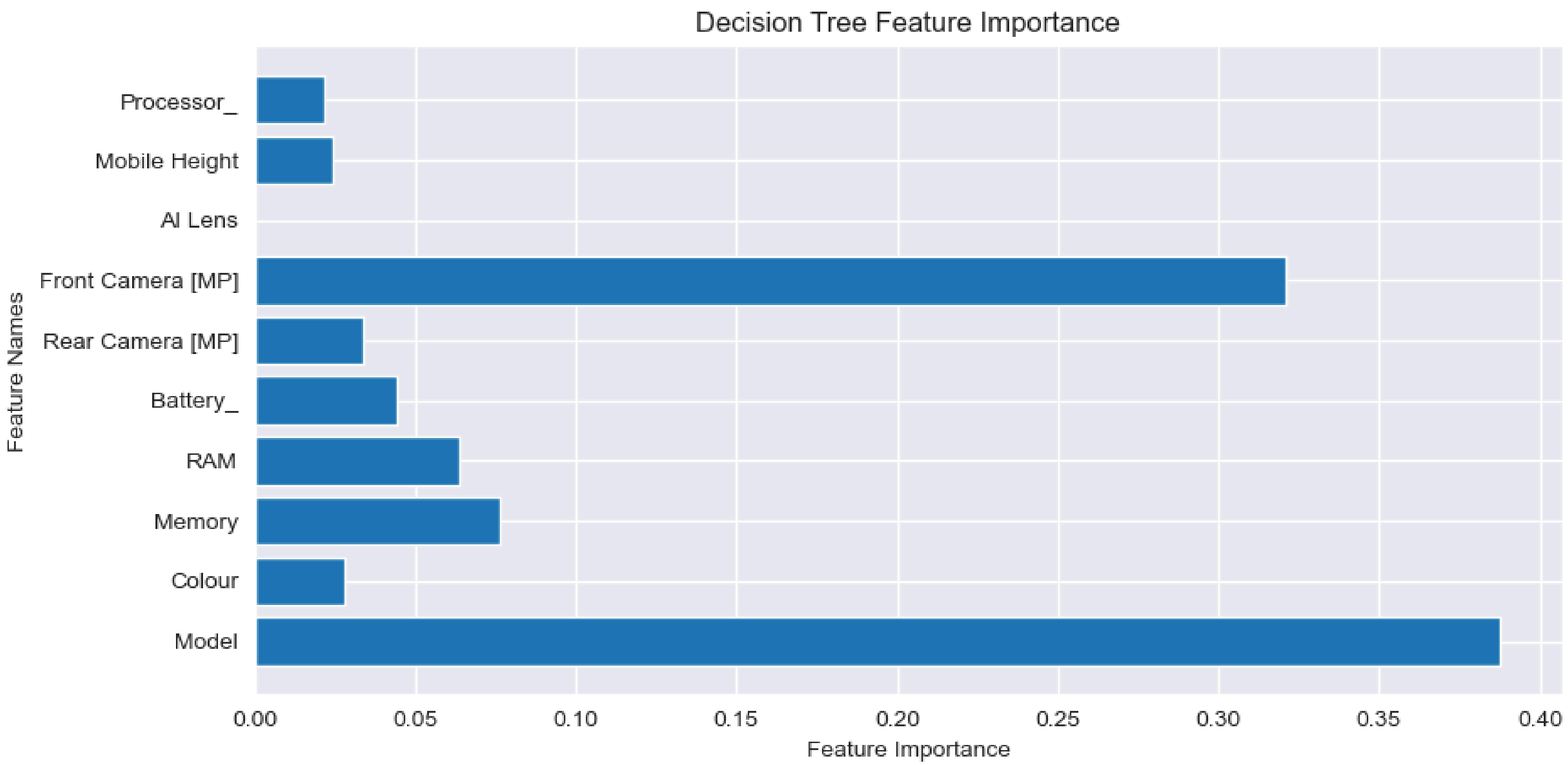
```python
import joblib
joblib.dump(dt_model, "final_mobile_price_model.pkl")
```

Out[121]:

```
['final_mobile_price_model.pkl']
```

In [123]:

```python
import matplotlib.pyplot as plt
feature_importance = dt_model.feature_importances_
plt.figure(figsize=(10,5))
plt.barh(X_train.columns, feature_importance)
plt.xlabel("Feature Importance")
plt.ylabel("Feature Names")
plt.title("Decision Tree Feature Importance")
plt.show()
```



```
In [125]: loaded_model = joblib.load("final_mobile_price_model.pkl")
          predicted_price = loaded_model.predict(new_data)
          print("Predicted Mobile Price:", predicted_price[0])

          Predicted Mobile Price: 0.09132286215188588
```

```
In [ ]:
```

```
In [146]: import os
          os.system("jupyter nbconvert --to pdf mobile_data.ipynb")

Out[146]: 1
```

```
In [141]: pip install pandoc
```

```
Requirement already satisfied: pandoc in c:\users\user\anaconda3\lib\site-packages (2.4)
Requirement already satisfied: plumbum in c:\users\user\anaconda3\lib\site-packages (from pandoc) (1.9.0)
Requirement already satisfied: ply in c:\users\user\anaconda3\lib\site-packages (from pandoc) (3.11)
Requirement already satisfied: pywin32 in c:\users\user\anaconda3\lib\site-packages (from plumbum->pandoc) (305.1)
Note: you may need to restart the kernel to use updated packages.
```

In [143]:

```
  Cell In[143], line 1
    sudo apt install texlive-xetex texlive-fonts-recommended
         ^
SyntaxError: invalid syntax
```

In [ ]: