In [49]:
```
print("Muhammad Saqib Anwar")
print("Semester: 7th")
print("Shift: Evening")
print("MIS Number: 35960")

import networkx as nx
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import re
from collections import Counter
import pandas as pd

df = pd.read_csv(r"C:\Users\User\Desktop\Smartphones DataSet\Smartphones_cleaned_da
df.head()
```

```
Muhammad Saqib Anwar
Semester: 7th
Shift: Evening
MIS Number: 35960
```

Out[49]:

	brand_name	model	price	rating	has_5g	has_nfc	has_ir_blaster	processor_brand
0	oneplus	OnePlus 11 5G	54999	89.0	True	True	False	snapdragon
1	oneplus	OnePlus Nord CE 2 Lite 5G	19989	81.0	True	False	False	snapdragon
2	samsung	Samsung Galaxy A14 5G	16499	75.0	True	False	False	exynos
3	motorola	Motorola Moto G62 5G	14999	81.0	True	False	False	snapdragon
4	realme	Realme 10 Pro Plus	24999	82.0	True	False	False	dimensity

5 rows × 26 columns

In [35]:
```
# 📦 Import necessary libraries
import pandas as pd
import numpy as np



# 👀 Preview the original columns before normalization
print("Original Price, RAM, and Battery Values:")
display(df[['price', 'ram_capacity', 'battery_capacity']].head())
```

```
# 🟧 Apply Min-Max Normalization
df['price_minmax'] = (df['price'] - df['price'].min()) / (df['price'].max() - df['p
df['ram_minmax'] = (df['ram_capacity'] - df['ram_capacity'].min()) / (df['ram_capac
df['battery_minmax'] = (df['battery_capacity'] - df['battery_capacity'].min()) / (d

# ✅ Display the result
print("📊 After Min-Max Normalization:")
display(df[['price', 'price_minmax', 'ram_capacity', 'ram_minmax', 'battery_capacit
```

Original Price, RAM, and Battery Values:

	price	ram_capacity	battery_capacity
0	54999	12	5000.0
1	19989	6	5000.0
2	16499	4	5000.0
3	14999	6	5000.0
4	24999	6	5000.0

📊 After Min-Max Normalization:

	price	price_minmax	ram_capacity	ram_minmax	battery_capacity	battery_minmax
0	54999	0.079660	12	0.647059	5000.0	0.157540
1	19989	0.025507	6	0.294118	5000.0	0.157540
2	16499	0.020108	4	0.176471	5000.0	0.157540
3	14999	0.017788	6	0.294118	5000.0	0.157540
4	24999	0.033256	6	0.294118	5000.0	0.157540
5	16999	0.020882	6	0.294118	5000.0	0.157540
6	65999	0.096674	6	0.294118	3279.0	0.072253
7	29999	0.040990	8	0.411765	4980.0	0.156549
8	26749	0.035963	8	0.411765	4500.0	0.132762
9	28999	0.039443	8	0.411765	4500.0	0.132762

In [36]:
```
print("Original values before normalization:")
print(df[['price', 'ram_capacity', 'battery_capacity']].head())

# 🟧 Step 4: Apply Z-Score Normalization
df['price_zscore'] = (df['price'] - df['price'].mean()) / df['price'].std()
df['ram_zscore'] = (df['ram_capacity'] - df['ram_capacity'].mean()) / df['ram_capac
df['battery_zscore'] = (df['battery_capacity'] - df['battery_capacity'].mean()) / d

# ✅ Step 5: Show the result
print("\n📊 Z-Score Normalized Data:")
print(df[['price', 'price_zscore', 'ram_capacity', 'ram_zscore', 'battery_capacity'
```

```
Original values before normalization:
   price  ram_capacity  battery_capacity
0  54999            12            5000.0
1  19989             6            5000.0
2  16499             4            5000.0
3  14999             6            5000.0
4  24999             6            5000.0

📊 Z-Score Normalized Data:
   price  price_zscore  ram_capacity  ram_zscore  battery_capacity  \
0  54999      0.568618            12    1.982160            5000.0
1  19989     -0.316998             6   -0.204128            5000.0
2  16499     -0.405281             4   -0.932890            5000.0
3  14999     -0.443225             6   -0.204128            5000.0
4  24999     -0.190265             6   -0.204128            5000.0
5  16999     -0.392633             6   -0.204128            5000.0
6  65999      0.846875             6   -0.204128            3279.0
7  29999     -0.063784             8    0.524635            4980.0
8  26749     -0.145996             8    0.524635            4500.0
9  28999     -0.089080             8    0.524635            4500.0

   battery_zscore
0        0.180530
1        0.180530
2        0.180530
3        0.180530
4        0.180530
5        0.180530
6       -1.524207
7        0.160719
8       -0.314746
9       -0.314746
```

In [37]:

```python
# 🧮 Function to apply Decimal Scaling Normalization
def decimal_scaling(column):
    # Drop NaNs temporarily if they exist
    max_abs = column.dropna().abs().max()
    j = len(str(int(max_abs))) if max_abs != 0 else 1  # avoid divide-by-zero
    return column / (10 ** j)

# ✅ Apply Decimal Scaling on numeric columns
df['price_decimal'] = decimal_scaling(df['price'])
df['ram_decimal'] = decimal_scaling(df['ram_capacity'])
df['battery_decimal'] = decimal_scaling(df['battery_capacity'])

# 📊 Display result
print("📊 Decimal Scaling Result:")
print(df[['price', 'price_decimal', 'ram_capacity', 'ram_decimal', 'battery_capacit
```

```
📊 Decimal Scaling Result:
   price  price_decimal  ram_capacity  ram_decimal  battery_capacity  \
0  54999       0.054999            12         0.12            5000.0
1  19989       0.019989             6         0.06            5000.0
2  16499       0.016499             4         0.04            5000.0
3  14999       0.014999             6         0.06            5000.0
4  24999       0.024999             6         0.06            5000.0
5  16999       0.016999             6         0.06            5000.0
6  65999       0.065999             6         0.06            3279.0
7  29999       0.029999             8         0.08            4980.0
8  26749       0.026749             8         0.08            4500.0
9  28999       0.028999             8         0.08            4500.0

   battery_decimal
0          0.05000
1          0.05000
2          0.05000
3          0.05000
4          0.05000
5          0.05000
6          0.03279
7          0.04980
8          0.04500
9          0.04500
```

In [38]:

```python
# 🧪 Step 1: Show original values before binning
print("Original values before binning:")
print(df[['price', 'ram_capacity', 'battery_capacity']].head(10))

# 📊 Step 2: Apply Equi-Width Binning (4 bins)
df['price_bin_width'] = pd.cut(df['price'], bins=4, labels=["Low", "Medium", "High"
df['ram_bin_width'] = pd.cut(df['ram_capacity'], bins=4, labels=["Low", "Medium", "
df['battery_bin_width'] = pd.cut(df['battery_capacity'], bins=4, labels=["Low", "Me

# 📊 Step 3: Apply Equi-Depth Binning safely
# Step 3a: Create depth bin edges with duplicates dropped
battery_bins = pd.qcut(df['battery_capacity'], q=4, duplicates='drop', retbins=True
# Step 3b: Make matching number of labels
labels = ["Bin " + str(i+1) for i in range(len(battery_bins) - 1)]
# Step 3c: Apply qcut with corrected labels
df['battery_bin_depth'] = pd.qcut(df['battery_capacity'], q=4, labels=labels, dupli

# Step 3d: Apply price & RAM depth bins (they work fine with 4 labels)
df['price_bin_depth'] = pd.qcut(df['price'], q=4, labels=["Low", "Medium", "High",
df['ram_bin_depth'] = pd.qcut(df['ram_capacity'], q=4, labels=["Low", "Medium", "Hi

# ✅ Step 4: Show result
print("\n📊 Binned Data (Equi-Width and Equi-Depth):")
print(df[[
    'price', 'price_bin_width', 'price_bin_depth',
    'ram_capacity', 'ram_bin_width', 'ram_bin_depth',
    'battery_capacity', 'battery_bin_width', 'battery_bin_depth'
]].head(10))
```

```
Original values before binning:
   price  ram_capacity  battery_capacity
0  54999            12            5000.0
1  19989             6            5000.0
2  16499             4            5000.0
3  14999             6            5000.0
4  24999             6            5000.0
5  16999             6            5000.0
6  65999             6            3279.0
7  29999             8            4980.0
8  26749             8            4500.0
9  28999             8            4500.0

📊 Binned Data (Equi-Width and Equi-Depth):
   price price_bin_width price_bin_depth  ram_capacity ram_bin_width  \
0  54999             Low       Very High            12          High
1  19989             Low          Medium             6        Medium
2  16499             Low          Medium             4           Low
3  14999             Low          Medium             6        Medium
4  24999             Low            High             6        Medium
5  16999             Low          Medium             6        Medium
6  65999             Low       Very High             6        Medium
7  29999             Low            High             8        Medium
8  26749             Low            High             8        Medium
9  28999             Low            High             8        Medium

  ram_bin_depth  battery_capacity battery_bin_width battery_bin_depth
0     Very High            5000.0               Low             Bin 2
1        Medium            5000.0               Low             Bin 2
2           Low            5000.0               Low             Bin 2
3        Medium            5000.0               Low             Bin 2
4        Medium            5000.0               Low             Bin 2
5        Medium            5000.0               Low             Bin 2
6        Medium            3279.0               Low             Bin 1
7          High            4980.0               Low             Bin 2
8          High            4500.0               Low             Bin 1
9          High            4500.0               Low             Bin 1
```

In [39]:

```python
# 🧮 Spearman correlation with 'price'
spearman_corr_price = df.corr(method='spearman', numeric_only=True)['price']

# 📋 Sort by absolute value
spearman_corr_price_sorted = spearman_corr_price.abs().sort_values(ascending=False)

# 🖥️ Display clean table
print("📊 Spearman Correlation with Price (sorted):\n")
print(spearman_corr_price_sorted)
```

```
📊 Spearman Correlation with Price (sorted):

price                        1.000000
price_zscore                 1.000000
price_minmax                 1.000000
price_decimal                1.000000
processor_speed              0.790753
rating                       0.771765
internal_memory              0.760902
ram_zscore                   0.750815
ram_capacity                 0.750815
ram_decimal                  0.750815
ram_minmax                   0.750815
has_nfc                      0.694446
has_5g                       0.664545
extended_memory_available    0.660470
fast_charging                0.650351
resolution_width             0.625008
resolution_height            0.607384
primary_camera_front         0.558234
refresh_rate                 0.556748
fast_charging_available      0.405345
primary_camera_rear          0.370463
extended_upto                0.352487
num_rear_cameras             0.328281
battery_zscore               0.323359
battery_minmax               0.323359
battery_decimal              0.323359
battery_capacity             0.323359
screen_size                  0.300630
num_front_cameras            0.144359
has_ir_blaster               0.026769
num_cores                    0.001262
Name: price, dtype: float64
```

In [43]:

```python
edges = df[['brand_name', 'processor_brand']].dropna()

# ✅ Create a directed graph
G = nx.DiGraph()

# ➕ Add edges: brand → processor used
for i, row in edges.iterrows():
    G.add_edge(row['brand_name'], row['processor_brand'])

# 🧠 Compute PageRank
pagerank_scores = nx.pagerank(G)

# 📋 Convert to DataFrame
pagerank_df = pd.DataFrame(pagerank_scores.items(), columns=['Node', 'PageRank']).s

# 🖥️ Show top results
print("📊 PageRank (Top nodes):\n")
print(pagerank_df.head(10))
```

📊 PageRank (Top nodes):

	Node	PageRank
1	snapdragon	0.178949
13	helio	0.104045
6	dimensity	0.067886
19	tiger	0.040014
15	unisoc	0.033988
21	google	0.018496
38	spreadtrum	0.017415
8	bionic	0.015155
46	fusion	0.015155
47	kirin	0.015155

In [46]:
```
edges_camera = df[['brand_name', 'primary_camera_rear']].dropna()

# ✅ Create a directed graph: brand → camera size
G_camera = nx.DiGraph()
for _, row in edges_camera.iterrows():
    brand = row['brand_name']
    camera = str(row['primary_camera_rear'])  # Convert to string in case of number
    G_camera.add_edge(brand, camera)

# 🧠 Apply PageRank
pagerank_camera = nx.pagerank(G_camera)

# 📊 Convert to DataFrame for display
pagerank_df = pd.DataFrame(pagerank_camera.items(), columns=['Node', 'PageRank'])

# 🔍 Filter to show only brands (not camera sizes)
brand_ranks = pagerank_df[pagerank_df['Node'].isin(df['brand_name'].unique())]

# ✅ Sort and display top brands by PageRank (camera importance)
brand_ranks = brand_ranks.sort_values(by='PageRank', ascending=False)

print("📸 Smartphone Brands Ranked by Camera PageRank:")
print(brand_ranks.head(10))
```

📸 Smartphone Brands Ranked by Camera PageRank:

	Node	PageRank
0	oneplus	0.009699
3	samsung	0.009699
4	motorola	0.009699
5	realme	0.009699
7	apple	0.009699
9	xiaomi	0.009699
11	nothing	0.009699
12	oppo	0.009699
13	vivo	0.009699
15	poco	0.009699

In [47]:
```
# Example: NLP on model names (simulating textual data)
df['model_cleaned'] = df['model'].str.lower().str.replace(r'[^a-z0-9 ]', '', regex=

# Split words and count most common
from collections import Counter
```

```
word_counts = Counter(" ".join(df['model_cleaned']).split())
common_words = word_counts.most_common(10)

print("📘 Most Common Words in Model Names:")
for word, count in common_words:
    print(f"{word}: {count}")
```

```
📘 Most Common Words in Model Names:
5g: 307
ram: 221
pro: 202
128gb: 135
xiaomi: 134
samsung: 132
galaxy: 132
vivo: 111
redmi: 103
realme: 97
```

In [51]:

```
df_filtered = df[['model', 'has_5g', 'has_nfc']].dropna()

# Convert to string for grouping
df_filtered['has_5g'] = df_filtered['has_5g'].astype(str)
df_filtered['has_nfc'] = df_filtered['has_nfc'].astype(str)

# 📊 Count 5G support by model
print("🔍 5G Support Count by Model:")
print(df_filtered.groupby(['model', 'has_5g']).size().unstack(fill_value=0))

# 📊 Count NFC support by model
print("\n🔍 NFC Support Count by Model:")
print(df_filtered.groupby(['model', 'has_nfc']).size().unstack(fill_value=0))
```

🔍 5G Support Count by Model:

has_5g	False	True
model		
Apple iPhone 11	1	0
Apple iPhone 11 (128GB)	1	0
Apple iPhone 11 Pro Max	1	0
Apple iPhone 12	0	1
Apple iPhone 12 (128GB)	0	1
...	...	...
itel S16	1	0
itel S16 Pro	1	0
itel Vision 1 (3GB RAM + 32GB)	1	0
itel Vision 3	1	0
itel Vision 3 (2GB RAM + 32GB)	1	0

[980 rows x 2 columns]

🔍 NFC Support Count by Model:

has_nfc	False	True
model		
Apple iPhone 11	0	1
Apple iPhone 11 (128GB)	0	1
Apple iPhone 11 Pro Max	0	1
Apple iPhone 12	0	1
Apple iPhone 12 (128GB)	0	1
...	...	...
itel S16	1	0
itel S16 Pro	1	0
itel Vision 1 (3GB RAM + 32GB)	1	0
itel Vision 3	1	0
itel Vision 3 (2GB RAM + 32GB)	1	0

[980 rows x 2 columns]