task 1.b

Smartphone Dataset: The smartphone dataset holds tremendous potential for in-depth analysis and is highly suitable for various machine-learning applications. With a wide array of relevant features such as brand, model, price, ratings, 5G capability, processor details, battery capacity, camera specifications, and more, the dataset provides a comprehensive view of smartphone attributes and user preferences. Machine learning tasks like regression can be utilized for predicting smartphone prices based on the given features, while classification can help identify user preferences for 5G-enabled or fast-charging smartphones. Sentiment analysis can be applied to the average ratings, providing insights into consumer satisfaction levels. Additionally, clustering algorithms can group smartphones based on similarities, facilitating targeted marketing and product customization. The dataset's richness and diversity make it ideal for training predictive models to aid businesses in understanding consumer choices, enhancing product development, and tailoring marketing strategies to match user preferences, ultimately leading to improved customer satisfaction and market success for smartphone manufacturers and sellers.

In [24]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

In [25]:
```
df = pd.read_csv("C:/Users/Rahil/Downloads/Assignment 10/smartphones.csv")
```

In [26]:
```
df.head()
```

Out[26]:

	brand_name	model	price	avg_rating	5G_or_not	processor_brand	num_cores	processor_speed
0	apple	Apple iPhone 11	38999	7.3	0	bionic	6.0	2.65
1	apple	Apple iPhone 11 (128GB)	46999	7.5	0	bionic	6.0	2.65
2	apple	Apple iPhone 11 Pro Max	109900	7.7	0	bionic	6.0	2.65
3	apple	Apple iPhone 12	51999	7.4	1	bionic	6.0	3.10
4	apple	Apple iPhone 12 (128GB)	55999	7.5	1	bionic	6.0	3.10

5 rows × 22 columns

Task-2.a

In [27]:

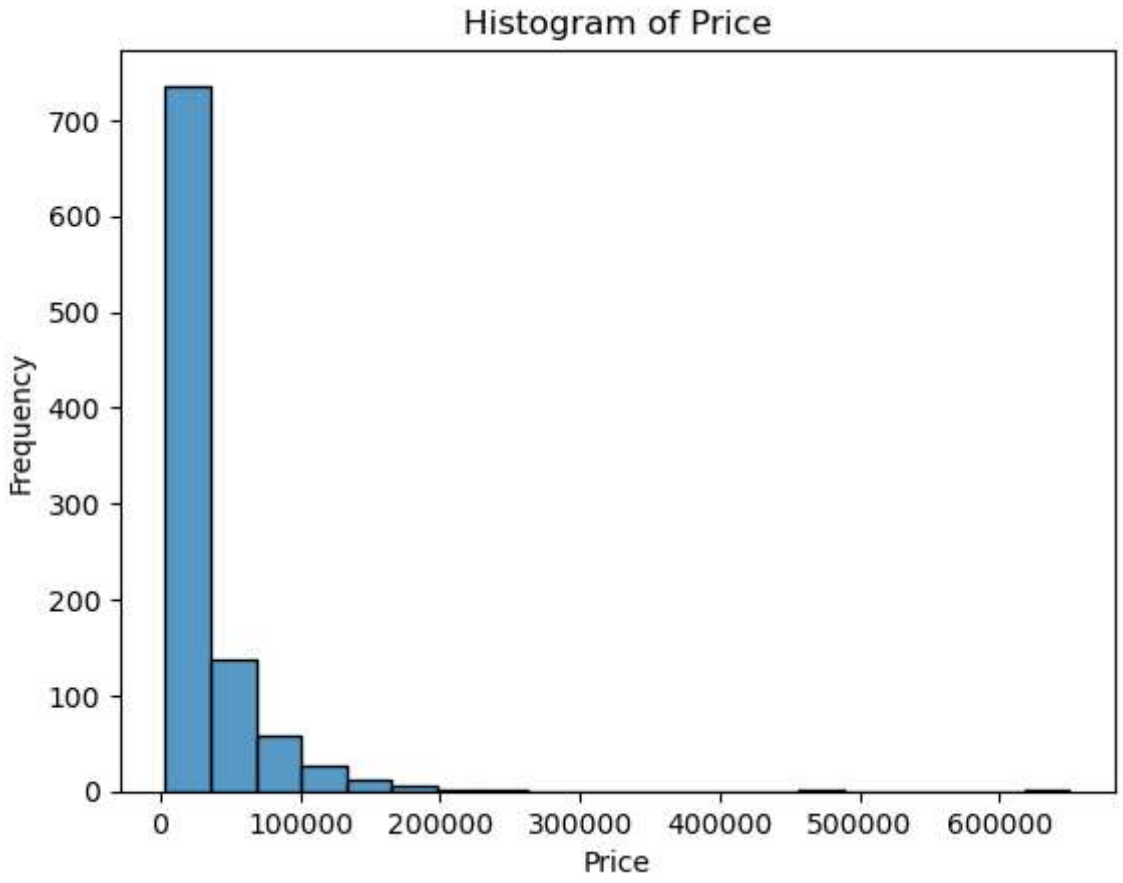
```
# Task-2.a
# Get the summary statistics of numerical columns
print("Summary statistics:\n {}".format(df.describe()))

# Get information about the dataset, including data types and missing values
print("Information about the dataset:\n {}".format(df.info()))
```

```
Summary statistics:
                price  avg_rating   5G_or_not   num_cores  processor_speed  \
count     980.000000  879.000000  980.000000  974.000000       938.000000
mean    32520.504082    7.825825    0.560204    7.772074         2.427217
std     39531.812669    0.740285    0.496616    0.836845         0.464090
min      3499.000000    6.000000    0.000000    4.000000         1.200000
25%     12999.000000    7.400000    0.000000    8.000000         2.050000
50%     19994.500000    8.000000    1.000000    8.000000         2.300000
75%     35491.500000    8.400000    1.000000    8.000000         2.840000
max    650000.000000    8.900000    1.000000    8.000000         3.220000

       battery_capacity  fast_charging_available  fast_charging  ram_capacity  \
count         969.000000               980.000000     769.000000    980.000000
mean         4817.748194                 0.854082      46.126138      6.560204
std          1009.540054                 0.353205      34.277870      2.744378
min          1821.000000                 0.000000      10.000000      1.000000
25%          4500.000000                 1.000000      18.000000      4.000000
50%          5000.000000                 1.000000      33.000000      6.000000
75%          5000.000000                 1.000000      66.000000      8.000000
max         22000.000000                 1.000000     240.000000     18.000000

       internal_memory  screen_size  refresh_rate  num_rear_cameras  \
count       980.000000   980.000000    980.000000        980.000000
mean        141.036735     6.536765     92.256122          2.814286
std         107.134516     0.349162     28.988052          0.776441
min           8.000000     3.540000     60.000000          1.000000
25%          64.000000     6.500000     60.000000          2.000000
50%         128.000000     6.580000     90.000000          3.000000
75%         128.000000     6.670000    120.000000          3.000000
max        1024.000000     8.030000    240.000000          4.000000

       primary_camera_rear  primary_camera_front  extended_memory_available  \
count           980.000000            975.000000                 980.000000
mean             50.319286             16.589744                   0.630612
std              33.000968             10.876944                   0.482885
min               2.000000              0.000000                   0.000000
25%              24.000000              8.000000                   0.000000
50%              50.000000             16.000000                   1.000000
75%              64.000000             16.000000                   1.000000
max             200.000000             60.000000                   1.000000

       resolution_height  resolution_width
count         980.000000        980.000000
mean         2214.663265       1075.852041
std           516.484254        290.164931
min           480.000000        480.000000
25%          1612.000000       1080.000000
50%          2400.000000       1080.000000
75%          2408.000000       1080.000000
max          3840.000000       2460.000000
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 980 entries, 0 to 979
Data columns (total 22 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   brand_name                980 non-null    object
 1   model                     980 non-null    object
 2   price                     980 non-null    int64
 3   avg_rating                879 non-null    float64
 4   5G_or_not                 980 non-null    int64
```

```
 5   processor_brand            960 non-null    object
 6   num_cores                  974 non-null    float64
 7   processor_speed            938 non-null    float64
 8   battery_capacity           969 non-null    float64
 9   fast_charging_available    980 non-null    int64
 10  fast_charging              769 non-null    float64
 11  ram_capacity               980 non-null    int64
 12  internal_memory            980 non-null    int64
 13  screen_size                980 non-null    float64
 14  refresh_rate               980 non-null    int64
 15  num_rear_cameras           980 non-null    int64
 16  os                         966 non-null    object
 17  primary_camera_rear        980 non-null    float64
 18  primary_camera_front       975 non-null    float64
 19  extended_memory_available  980 non-null    int64
 20  resolution_height          980 non-null    int64
 21  resolution_width           980 non-null    int64
dtypes: float64(8), int64(10), object(4)
memory usage: 168.6+ KB
Information about the dataset:
 None
```

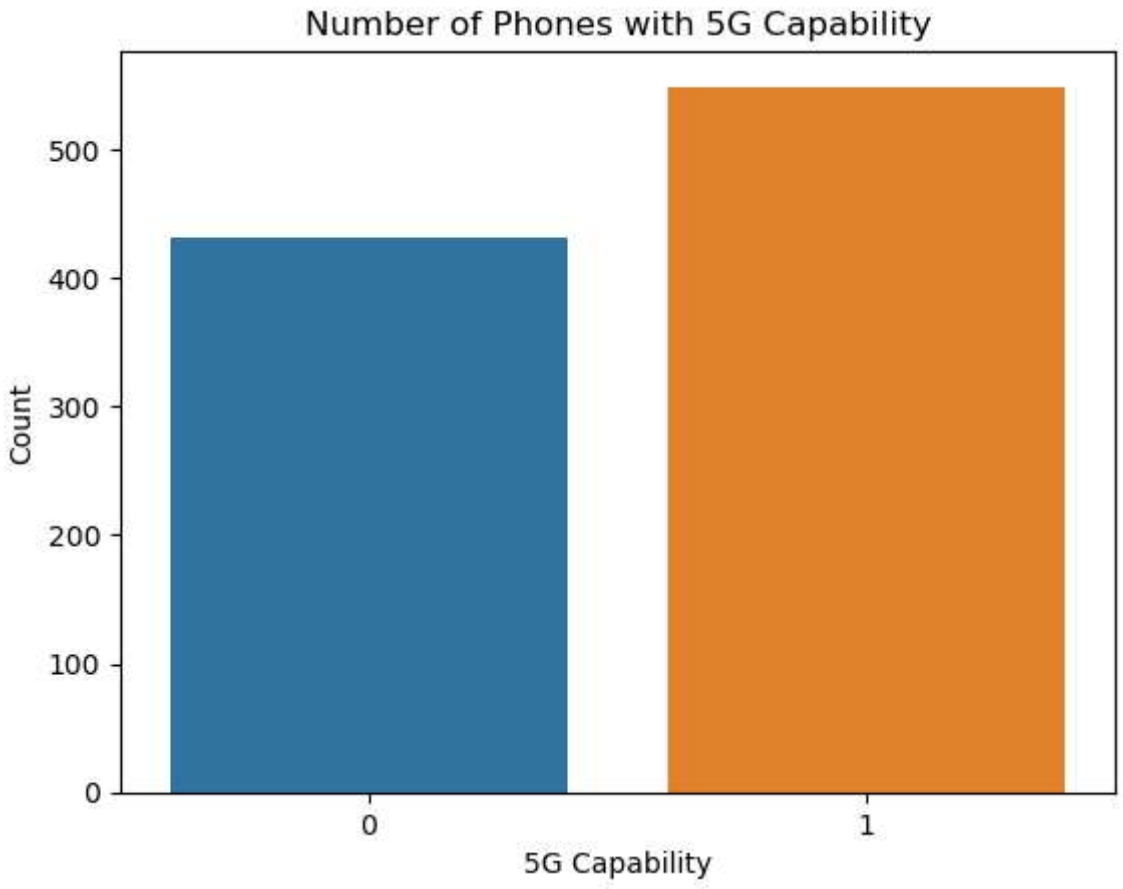
In [28]:
```
#pip install seaborn
```

In [29]:
```
sns.histplot(df['price'], bins=20)
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.title('Histogram of Price')
plt.show()
```



task-2.b

In [30]:
```
# task-2.b



# Example: Create a bar plot for the number of phones with 5G capability
sns.countplot(x='5G_or_not', data=df)
plt.xlabel('5G Capability')
plt.ylabel('Count')
plt.title('Number of Phones with 5G Capability')
plt.show()
```



In [31]:
```
#pip install scikit-learn
```

task-3.a

In [32]:
```
# task-3.a
from sklearn.preprocessing import StandardScaler, LabelEncoder


# Handling missing values: Fill missing values in 'battery_capacity' with the mean val
mean_battery_capacity = df['battery_capacity'].mean()
df['battery_capacity'].fillna(mean_battery_capacity, inplace=True)

# Encoding categorical variables: Convert 'brand_name' and 'os' columns to numerical l
```

```
label_encoder = LabelEncoder()
df['brand_name'] = label_encoder.fit_transform(df['brand_name'])
df['os'] = label_encoder.fit_transform(df['os'])

# Normalize numerical features: Standardize the 'price' column
scaler = StandardScaler()
df['price'] = scaler.fit_transform(df[['price']])

# Drop unnecessary columns if needed (e.g., 'model' and 'resolution_width' in this cas
#df.drop(['model', 'resolution_width'], axis=1, inplace=True) REQUIRED

# Print the first few rows of the preprocessed DataFrame
print(df.head())

# Save the preprocessed DataFrame to a new CSV file (if needed)
```

```
   brand_name                    model     price  avg_rating  5G_or_not  \
0           0          Apple iPhone 11  0.163964         7.3          0
1           0  Apple iPhone 11 (128GB)  0.366436         7.5          0
2           0  Apple iPhone 11 Pro Max  1.958398         7.7          0
3           0          Apple iPhone 12  0.492981         7.4          1
4           0  Apple iPhone 12 (128GB)  0.594217         7.5          1

  processor_brand  num_cores  processor_speed  battery_capacity  \
0          bionic        6.0             2.65       3110.000000
1          bionic        6.0             2.65       3110.000000
2          bionic        6.0             2.65       3500.000000
3          bionic        6.0             3.10       4817.748194
4          bionic        6.0             3.10       4817.748194

   fast_charging_available  ...  internal_memory  screen_size  refresh_rate  \
0                        0  ...               64          6.1            60
1                        0  ...              128          6.1            60
2                        1  ...               64          6.5            60
3                        0  ...               64          6.1            60
4                        0  ...              128          6.1            60

   num_rear_cameras  os  primary_camera_rear  primary_camera_front  \
0                 2   1                 12.0                  12.0
1                 2   1                 12.0                  12.0
2                 3   1                 12.0                  12.0
3                 2   1                 12.0                  12.0
4                 2   1                 12.0                  12.0

   extended_memory_available  resolution_height  resolution_width
0                          0               1792               828
1                          0               1792               828
2                          0               2688              1242
3                          0               2532              1170
4                          0               2532              1170

[5 rows x 22 columns]
```

In [33]:

```
#task-3.b
# Check for missing values in the dataset
print(df.isnull().sum())

# Handling missing values in 'battery_capacity' column: Fill with mean value
mean_battery_capacity = df['battery_capacity'].mean()
df['battery_capacity'].fillna(mean_battery_capacity, inplace=True)
```

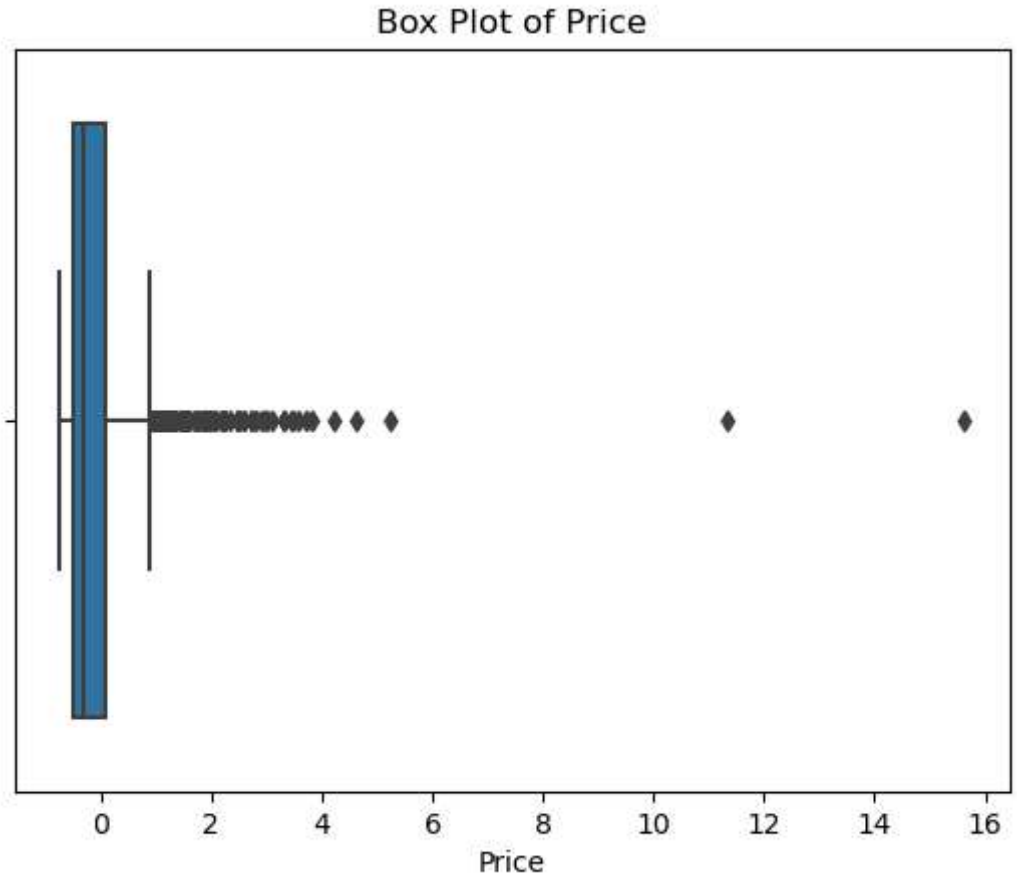
```
avg_rating = df['avg_rating'].mean()
df['avg_rating'].fillna(avg_rating, inplace=True)


fast_charging = df['fast_charging'].mode()[0]
df['fast_charging'].fillna(fast_charging, inplace=True)

processor_speed = df['processor_speed'].mean()
df['processor_speed'].fillna(processor_speed, inplace=True)

num_cores = df['num_cores'].mode()[0]
df['num_cores'].fillna(num_cores, inplace=True)

print(df.head())
print(df.isnull().sum())
# Handling missing values in other columns (if any) using appropriate methods
# For example, you can drop rows with missing values using df.dropna(), or fill with m
```

```
brand_name                    0
model                         0
price                         0
avg_rating                  101
5G_or_not                     0
processor_brand              20
num_cores                     6
processor_speed              42
battery_capacity              0
fast_charging_available       0
fast_charging               211
ram_capacity                  0
internal_memory               0
screen_size                   0
refresh_rate                  0
num_rear_cameras              0
os                            0
primary_camera_rear           0
primary_camera_front          5
extended_memory_available     0
resolution_height             0
resolution_width              0
dtype: int64
   brand_name                    model     price  avg_rating  5G_or_not  \
0           0          Apple iPhone 11  0.163964         7.3          0
1           0  Apple iPhone 11 (128GB)  0.366436         7.5          0
2           0  Apple iPhone 11 Pro Max  1.958398         7.7          0
3           0          Apple iPhone 12  0.492981         7.4          1
4           0  Apple iPhone 12 (128GB)  0.594217         7.5          1

  processor_brand  num_cores  processor_speed  battery_capacity  \
0          bionic        6.0             2.65       3110.000000
1          bionic        6.0             2.65       3110.000000
2          bionic        6.0             2.65       3500.000000
3          bionic        6.0             3.10       4817.748194
4          bionic        6.0             3.10       4817.748194

   fast_charging_available  ...  internal_memory  screen_size  refresh_rate  \
0                        0  ...               64          6.1            60
1                        0  ...              128          6.1            60
2                        1  ...               64          6.5            60
3                        0  ...               64          6.1            60
4                        0  ...              128          6.1            60

   num_rear_cameras  os  primary_camera_rear  primary_camera_front  \
0                 2   1                 12.0                  12.0
1                 2   1                 12.0                  12.0
2                 3   1                 12.0                  12.0
3                 2   1                 12.0                  12.0
4                 2   1                 12.0                  12.0

   extended_memory_available  resolution_height  resolution_width
0                          0               1792               828
1                          0               1792               828
2                          0               2688              1242
3                          0               2532              1170
4                          0               2532              1170

[5 rows x 22 columns]
brand_name                  0
```

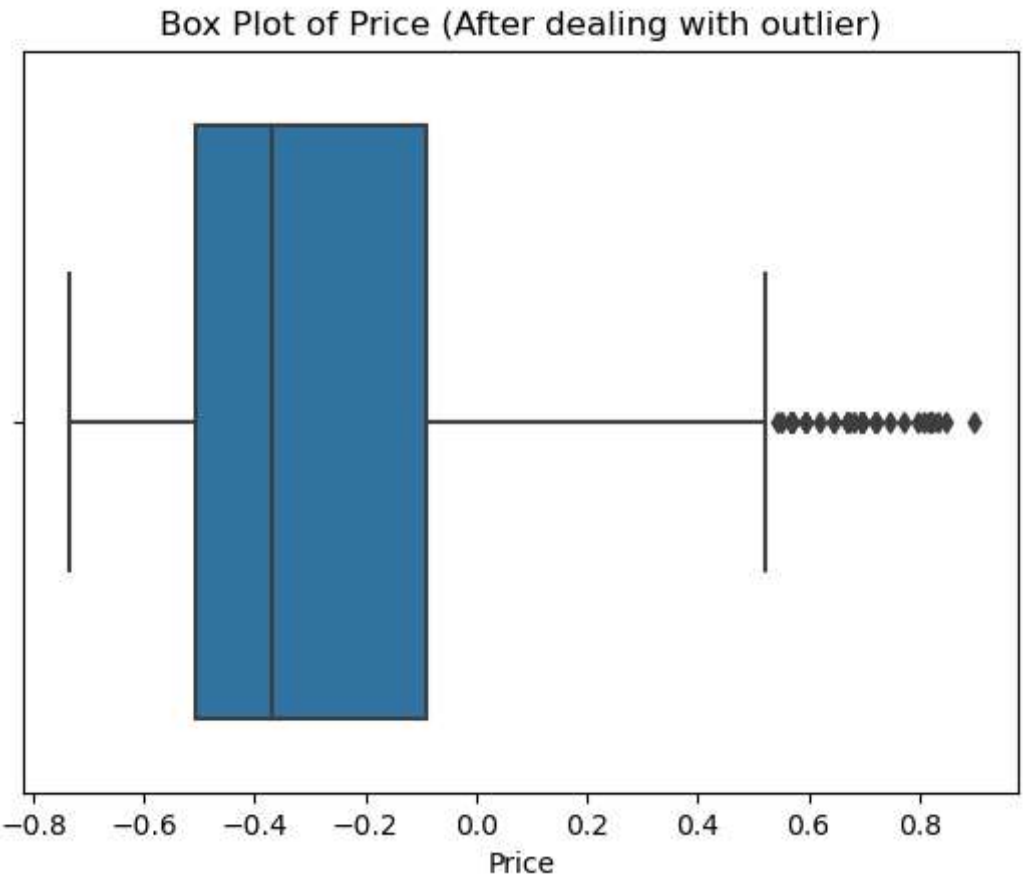
```
model                          0
price                          0
avg_rating                     0
5G_or_not                      0
processor_brand               20
num_cores                      0
processor_speed                0
battery_capacity               0
fast_charging_available        0
fast_charging                  0
ram_capacity                   0
internal_memory                0
screen_size                    0
refresh_rate                   0
num_rear_cameras               0
os                             0
primary_camera_rear            0
primary_camera_front           5
extended_memory_available      0
resolution_height              0
resolution_width               0
dtype: int64
```

In [34]:

```python
# task-3.b
# handling outliner
import seaborn as sns

# Example: Detect outliers in 'price' column using a box plot
sns.boxplot(x=df['price'])
plt.xlabel('Price')
plt.title('Box Plot of Price')
plt.show()



# You can use other methods like Z-score, IQR, or visualizations to identify and handl
# For example, to remove outliers in 'price', you can use:
# Handling outliers: Remove outliers in 'price' using the IQR method
Q1 = df['price'].quantile(0.25)
Q3 = df['price'].quantile(0.75)
IQR = Q3 - Q1
upper_bound = Q3 + 1.5 * IQR

df = df[df['price'] < upper_bound]
print(df.head())

sns.boxplot(x=df['price'])
plt.xlabel('Price')
plt.title('Box Plot of Price (After dealing with outlier)')
plt.show()
```


Box Plot of Price
0
2
4
6
8
10
12
14
16
Price

```
   brand_name                    model     price  avg_rating  5G_or_not  \
0           0          Apple iPhone 11  0.163964         7.3          0
1           0  Apple iPhone 11 (128GB)  0.366436         7.5          0
3           0          Apple iPhone 12  0.492981         7.4          1
4           0  Apple iPhone 12 (128GB)  0.594217         7.5          1
5           0  Apple iPhone 12 (256GB)  0.897925         7.6          1

  processor_brand  num_cores  processor_speed  battery_capacity  \
0         bionic        6.0             2.65       3110.000000
1         bionic        6.0             2.65       3110.000000
3         bionic        6.0             3.10       4817.748194
4         bionic        6.0             3.10       4817.748194
5         bionic        6.0             3.10       4817.748194

   fast_charging_available  ...  internal_memory  screen_size  refresh_rate  \
0                        0  ...               64          6.1            60
1                        0  ...              128          6.1            60
3                        0  ...               64          6.1            60
4                        0  ...              128          6.1            60
5                        0  ...              256          6.1            60

   num_rear_cameras  os  primary_camera_rear  primary_camera_front  \
0                 2   1                 12.0                  12.0
1                 2   1                 12.0                  12.0
3                 2   1                 12.0                  12.0
4                 2   1                 12.0                  12.0
5                 2   1                 12.0                  12.0

   extended_memory_available  resolution_height  resolution_width
0                          0               1792               828
1                          0               1792               828
3                          0               2532              1170
4                          0               2532              1170
5                          0               2532              1170

[5 rows x 22 columns]
```



In [35]:
```
# task-3.b
# Feature Enginnering

df['pixel_density'] = df['resolution_height'] / df['screen_size']
print(df.head())
```

```
   brand_name                    model     price  avg_rating  5G_or_not  \
0           0          Apple iPhone 11  0.163964         7.3          0
1           0  Apple iPhone 11 (128GB)  0.366436         7.5          0
3           0          Apple iPhone 12  0.492981         7.4          1
4           0  Apple iPhone 12 (128GB)  0.594217         7.5          1
5           0  Apple iPhone 12 (256GB)  0.897925         7.6          1

  processor_brand  num_cores  processor_speed  battery_capacity  \
0         bionic        6.0             2.65       3110.000000
1         bionic        6.0             2.65       3110.000000
3         bionic        6.0             3.10       4817.748194
4         bionic        6.0             3.10       4817.748194
5         bionic        6.0             3.10       4817.748194

   fast_charging_available  ...  screen_size  refresh_rate  num_rear_cameras  \
0                        0  ...          6.1            60                 2
1                        0  ...          6.1            60                 2
3                        0  ...          6.1            60                 2
4                        0  ...          6.1            60                 2
5                        0  ...          6.1            60                 2

   os  primary_camera_rear  primary_camera_front  extended_memory_available  \
0   1                 12.0                  12.0                          0
1   1                 12.0                  12.0                          0
3   1                 12.0                  12.0                          0
4   1                 12.0                  12.0                          0
5   1                 12.0                  12.0                          0

   resolution_height  resolution_width  pixel_density
0               1792               828     293.770492
1               1792               828     293.770492
3               2532              1170     415.081967
4               2532              1170     415.081967
5               2532              1170     415.081967

[5 rows x 23 columns]
```

In [36]:

```python
#TASK-4.A

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer

# Load the dataset
df = pd.read_csv("C:/Users/Rahil/Downloads/Assignment 10/smartphones.csv")

# Preprocess the data
mean_battery_capacity = df['battery_capacity'].mean()
df['battery_capacity'].fillna(mean_battery_capacity, inplace=True)
df['pixel_density'] = df['resolution_height'] / df['screen_size']
df.drop(['model', 'resolution_width'], axis=1, inplace=True)

# Handle missing values in other columns (if any) using appropriate methods
# For example, you can drop rows with missing values using df.dropna(), or fill with m
df.dropna(subset=['processor_brand'], inplace=True)
```

```
# One-hot encoding for 'processor_brand', 'brand_name', and 'os' columns
df = pd.get_dummies(df, columns=['processor_brand', 'brand_name', 'os'], drop_first=Tr

# Split the data into features (X) and target (y)
X = df.drop('5G_or_not', axis=1)
y = df['5G_or_not']

# Handle missing values in X
imputer = SimpleImputer(strategy='mean')
X = pd.DataFrame(imputer.fit_transform(X), columns=X.columns)

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

# Standardize numerical features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Implementing SVM classifier
svm_model = SVC(kernel='linear', random_state=42)
svm_model.fit(X_train_scaled, y_train)

# Making predictions using SVM
y_pred_svm = svm_model.predict(X_test_scaled)

# Implementing Random Forest classifier
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Making predictions using Random Forest
y_pred_rf = rf_model.predict(X_test)

# Evaluate the models
print("SVM Model Accuracy:", accuracy_score(y_test, y_pred_svm))
print("Random Forest Model Accuracy:", accuracy_score(y_test, y_pred_rf))

# Print classification reports for both models
print("SVM Model Classification Report:")
print(classification_report(y_test, y_pred_svm))

print("Random Forest Model Classification Report:")
print(classification_report(y_test, y_pred_rf))

# Print confusion matrix for both models
print("SVM Model Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_svm))

print("Random Forest Model Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_rf))
```

```
SVM Model Accuracy: 0.8958333333333334
Random Forest Model Accuracy: 0.921875
SVM Model Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.87      0.88        85
           1       0.90      0.92      0.91       107

    accuracy                           0.90       192
   macro avg       0.90      0.89      0.89       192
weighted avg       0.90      0.90      0.90       192

Random Forest Model Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.91      0.91        85
           1       0.93      0.93      0.93       107

    accuracy                           0.92       192
   macro avg       0.92      0.92      0.92       192
weighted avg       0.92      0.92      0.92       192

SVM Model Confusion Matrix:
[[74 11]
 [ 9 98]]
Random Forest Model Confusion Matrix:
[[ 77   8]
 [  7 100]]
```

In [37]:

```python
# TASK-4.B
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, c

# Evaluate SVM model
print("SVM Model Performance:")
print("Accuracy:", accuracy_score(y_test, y_pred_svm))
print("Precision:", precision_score(y_test, y_pred_svm))
print("Recall:", recall_score(y_test, y_pred_svm))
print("F1-Score:", f1_score(y_test, y_pred_svm))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_svm))

# Evaluate Random Forest model
print("\nRandom Forest Model Performance:")
print("Accuracy:", accuracy_score(y_test, y_pred_rf))
print("Precision:", precision_score(y_test, y_pred_rf))
print("Recall:", recall_score(y_test, y_pred_rf))
print("F1-Score:", f1_score(y_test, y_pred_rf))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_rf))
```

```
SVM Model Performance:
Accuracy: 0.8958333333333334
Precision: 0.8990825688073395
Recall: 0.9158878504672897
F1-Score: 0.9074074074074073
Confusion Matrix:
[[74 11]
 [ 9 98]]

Random Forest Model Performance:
Accuracy: 0.921875
Precision: 0.9259259259259259
Recall: 0.9345794392523364
F1-Score: 0.9302325581395349
Confusion Matrix:
[[ 77   8]
 [  7 100]]
```

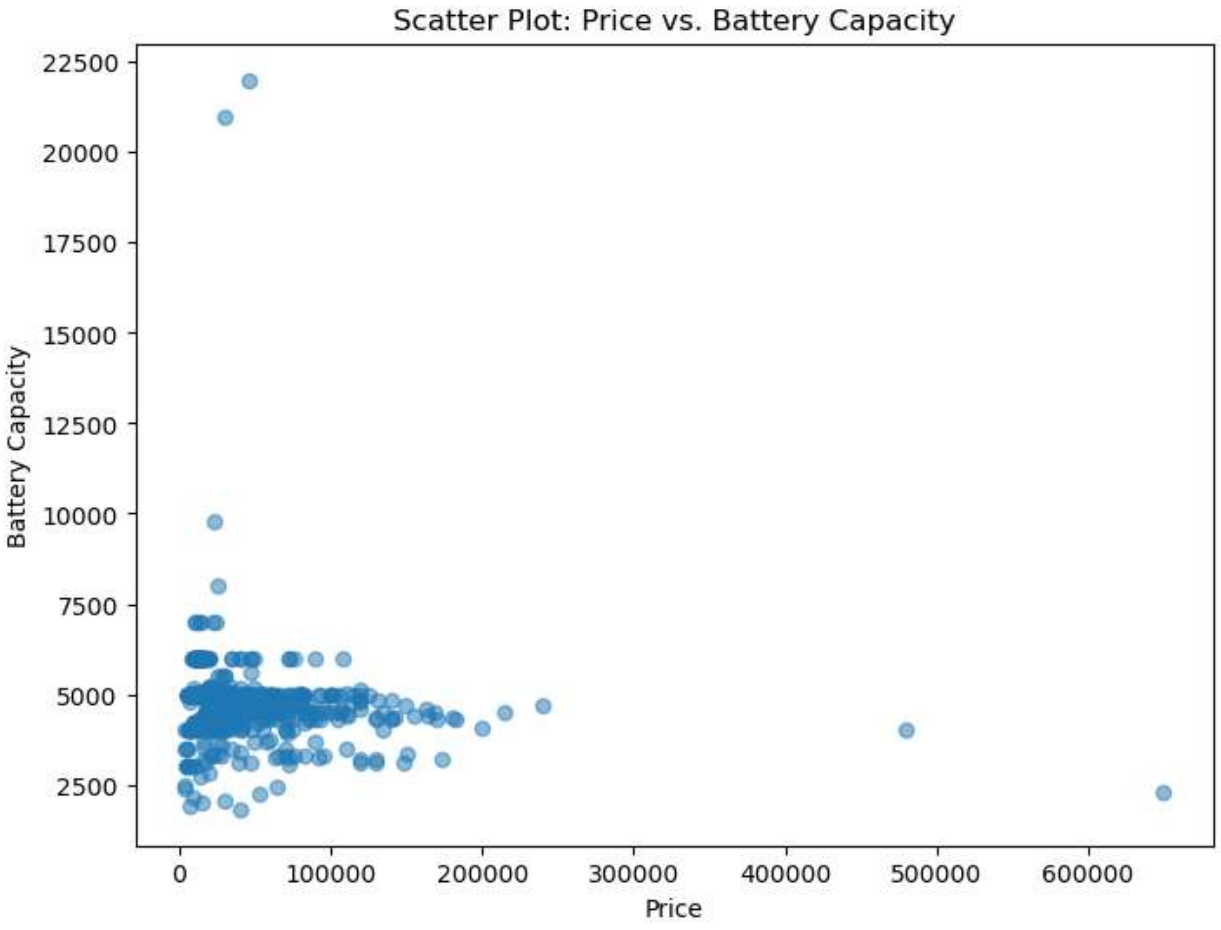
Task 4.b

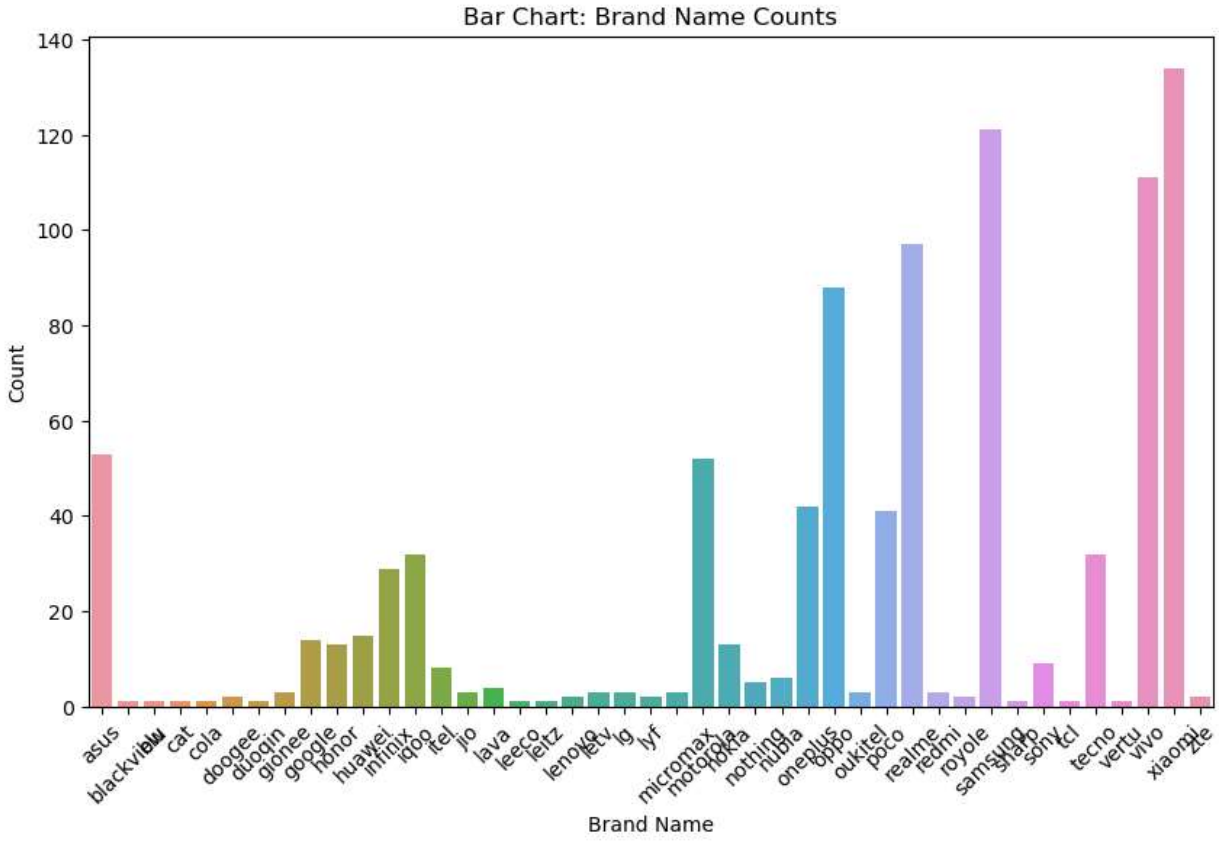
From the metrics, we can see that the Random Forest model has slightly higher values for accuracy, precision, recall, and F1-Score compared to the SVM model. Both models have relatively balanced precision and recall values, but the Random Forest model performs slightly better across the board.

In [38]:

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.impute import SimpleImputer

# Load the dataset
df = pd.read_csv("C:/Users/Rahil/Downloads/Assignment 10/smartphones.csv")

# Preprocess the data (same preprocessing as before)
mean_battery_capacity = df['battery_capacity'].mean()
df['battery_capacity'].fillna(mean_battery_capacity, inplace=True)
df['pixel_density'] = df['resolution_height'] / df['screen_size']
df.drop(['model', 'resolution_width'], axis=1, inplace=True)
df.dropna(subset=['processor_brand'], inplace=True)
df = pd.get_dummies(df, columns=['processor_brand', 'brand_name', 'os'], drop_first=Tr

# Convert one-hot encoded 'brand_name' back to categorical column
brand_name_cols = [col for col in df.columns if col.startswith('brand_name_')]
df['brand_name'] = df[brand_name_cols].idxmax(axis=1).str.replace('brand_name_', '')

# Drop the one-hot encoded 'brand_name' columns
df.drop(brand_name_cols, axis=1, inplace=True)

# Select numeric columns for correlation matrix
numeric_columns = df.select_dtypes(include=[float, int]).columns
df_numeric = df[numeric_columns]

# Handle missing values in numeric columns
imputer = SimpleImputer(strategy='mean')
df_numeric_imputed = pd.DataFrame(imputer.fit_transform(df_numeric), columns=df_numeri

# Create the correlation matrix
```

```
correlation_matrix = df_numeric_imputed.corr()


# Scatter plot of 'price' vs. 'battery_capacity'
plt.figure(figsize=(8, 6))
plt.scatter(df['price'], df['battery_capacity'], alpha=0.5)
plt.xlabel('Price')
plt.ylabel('Battery Capacity')
plt.title('Scatter Plot: Price vs. Battery Capacity')
plt.show()

# Bar chart for 'brand_name' counts
plt.figure(figsize=(10, 6))
sns.countplot(x='brand_name', data=df)
plt.xticks(rotation=45)
plt.xlabel('Brand Name')
plt.ylabel('Count')
plt.title('Bar Chart: Brand Name Counts')
plt.show()
```

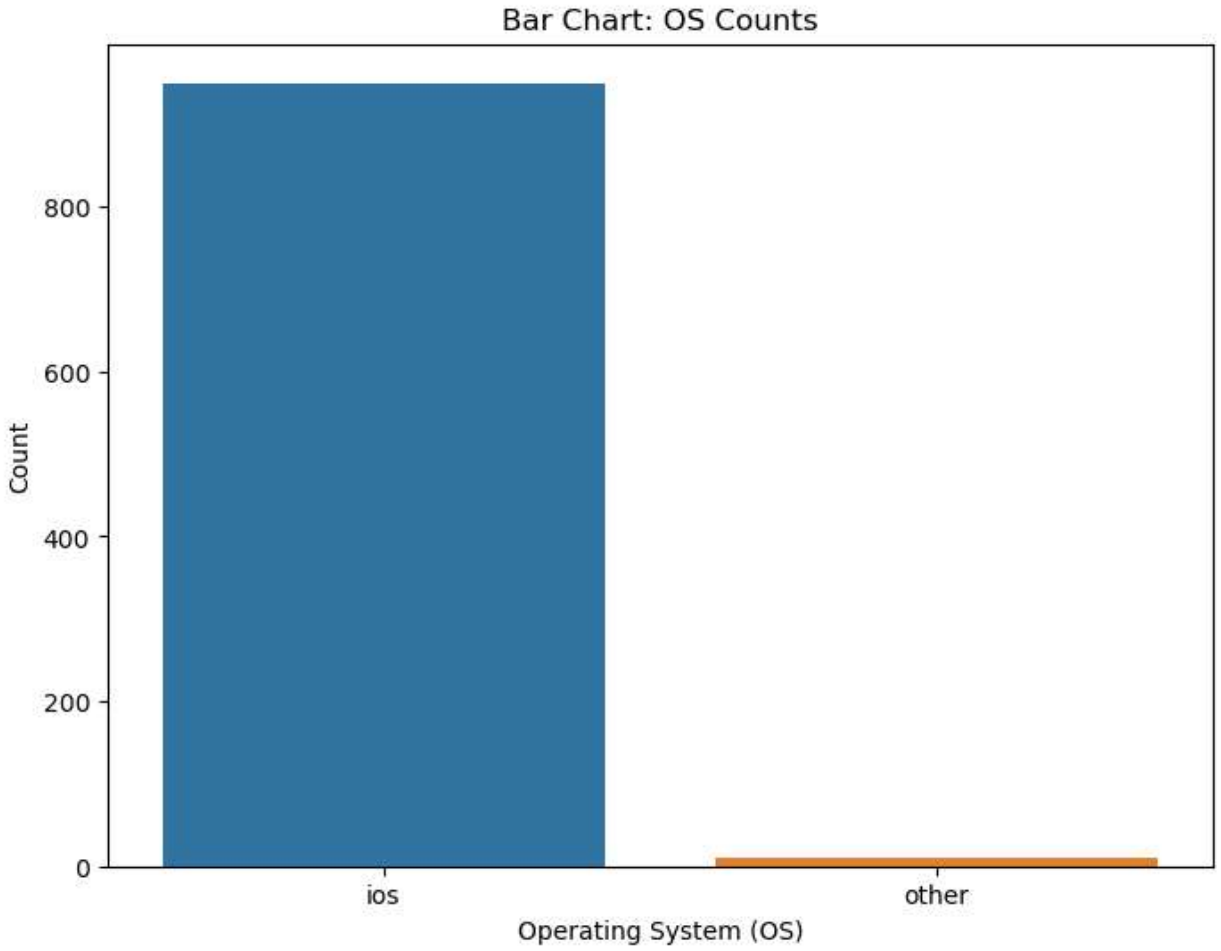




In [39]:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.impute import SimpleImputer

# Load the dataset
df = pd.read_csv("C:/Users/Rahil/Downloads/Assignment 10/smartphones.csv")

# Preprocess the data (same preprocessing as before)
mean_battery_capacity = df['battery_capacity'].mean()
df['battery_capacity'].fillna(mean_battery_capacity, inplace=True)
df['pixel_density'] = df['resolution_height'] / df['screen_size']
df.drop(['model', 'resolution_width'], axis=1, inplace=True)
df.dropna(subset=['processor_brand'], inplace=True)
df = pd.get_dummies(df, columns=['processor_brand', 'brand_name', 'os'], drop_first=Tr

# Convert one-hot encoded 'brand_name' and 'os' back to categorical columns
brand_name_cols = [col for col in df.columns if col.startswith('brand_name_')]
df['brand_name'] = df[brand_name_cols].idxmax(axis=1).str.replace('brand_name_', '')
df.drop(brand_name_cols, axis=1, inplace=True)

os_cols = [col for col in df.columns if col.startswith('os_')]
df['os'] = df[os_cols].idxmax(axis=1).str.replace('os_', '')
df.drop(os_cols, axis=1, inplace=True)




# Bar chart for 'os' counts
plt.figure(figsize=(8, 6))
sns.countplot(x='os', data=df)
plt.xlabel('Operating System (OS)')
```

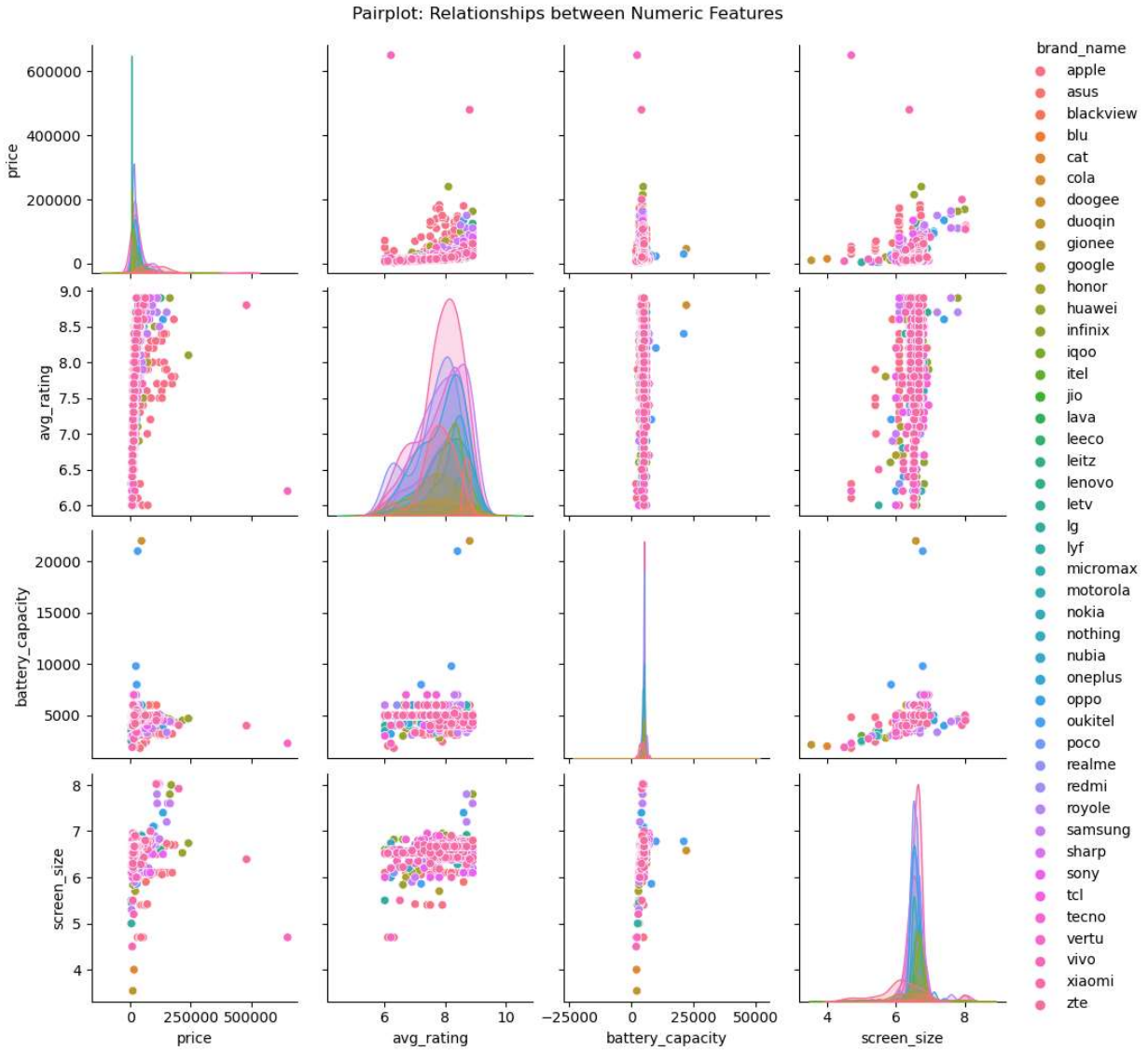
```
plt.ylabel('Count')
plt.title('Bar Chart: OS Counts')
plt.show()

# Drop non-numeric columns 'brand_name' and 'os' for the heatmap
df_corr = df.drop(['brand_name', 'os'], axis=1)
```



In [40]:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.impute import SimpleImputer

# Load the dataset
df = pd.read_csv("C:/Users/Rahil/Downloads/Assignment 10/smartphones.csv")

# Preprocess the data (same preprocessing as before)
mean_battery_capacity = df['battery_capacity'].mean()
df['battery_capacity'].fillna(mean_battery_capacity, inplace=True)
df['pixel_density'] = df['resolution_height'] / df['screen_size']
df.drop(['model', 'resolution_width'], axis=1, inplace=True)
df.dropna(subset=['processor_brand'], inplace=True)
df = pd.get_dummies(df, columns=['processor_brand', 'os'], drop_first=True)

# Plotting a subset of numeric features in the pairplot
subset_features = ['price', 'avg_rating', 'battery_capacity', 'screen_size']

# Check if 'brand_name' column is present before using it for hue
if 'brand_name' in df.columns:
    sns.pairplot(df[subset_features + ['brand_name']], hue='brand_name', diag_kind='k
```

```
else:
    print("No valid 'brand_name' data to plot.")
    sns.pairplot(df[subset_features], diag_kind='kde')

plt.suptitle('Pairplot: Relationships between Numeric Features', y=1.02)
plt.show()
```



In [ ]:

In [ ]: