1. Consider the Smart Phone dataset and perform exploratory data analysis.

i. Identify the dimension, structure, and summary of the data set

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
df=pd.read_csv("smartphones_cleaned_v6.csv")
print(df.shape)
df.info()
print(df.describe())
```

Output:

```
PS C:\Users\manju\OneDrive\Desktop\MCA\2nd SEM\ML\Lab Programs> & C:/Users/manju/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/manju/OneDrive/Desktop/MCA/2nd S
EM/ML/Lab Programs/1.1.py"
(980, 26)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 980 entries, 0 to 979
Data columns (total 26 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   rating                    879 non-null    float64
 4   has_5g                    980 non-null    bool
 5   has_nfc                   980 non-null    bool
 6   has_ir_blaster            980 non-null    bool
 7   processor_brand           960 non-null    object
 8   num_cores                 974 non-null    float64
 9   processor_speed           938 non-null    float64
 10  battery_capacity          969 non-null    float64
 11  fast_charging_available   980 non-null    int64
 12  fast_charging             769 non-null    float64
 13  ram_capacity              980 non-null    float64
 14  internal_memory           980 non-null    float64
 15  screen_size               980 non-null    float64
 16  refresh_rate              980 non-null    int64
 17  num_rear_cameras          980 non-null    int64
 18  num_front_cameras         976 non-null    float64
 19  os                        966 non-null    object
 20  primary_camera_rear       980 non-null    float64
 21  primary_camera_front      975 non-null    float64
 22  extended_memory_available 980 non-null    int64
 23  extended_upto             500 non-null    float64
 24  resolution_width          980 non-null    int64
 25  resolution_height         980 non-null    int64
dtypes: bool(3), float64(12), int64(7), object(4)
dtypes: bool(3), float64(12), int64(7), object(4)
memory usage: 179.1+ KB
               price     rating   num_cores  processor_speed  ...  extended_memory_available  extended_upto  resolution_width  resolution_height
count     980.000000  879.000000  974.000000       938.000000  ...                 980.000000     500.000000        980.000000         980.000000
mean    32520.504082   78.258248    7.772074         2.427217  ...                   0.630612     736.064000       1075.852041        2214.663265
std     39531.812669    7.402854    0.836845         0.464090  ...                   0.482885     366.894911        290.164931         516.484254
min      3499.000000   60.000000    4.000000         1.200000  ...                   0.000000      32.000000        480.000000         480.000000
25%     12999.000000   74.000000    8.000000         2.050000  ...                   0.000000     512.000000       1080.000000        1612.000000
50%     19994.500000   80.000000    8.000000         2.300000  ...                   1.000000    1024.000000       1080.000000        2400.000000
75%     35491.500000   84.000000    8.000000         2.840000  ...                   1.000000    1024.000000       1080.000000        2408.000000
max    650000.000000   89.000000    8.000000         3.220000  ...                   1.000000    2048.000000       2460.000000        3840.000000

[8 rows x 19 columns]
```

ii. Pre-process the dataset and treat them (like missing values, 'na'?). Justify the treatment

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
df=pd.read_csv("smartphones_cleaned_v6.csv")#
Identify missing values
print("TOTAL MISSING VALUES ")
print(df.isnull().sum())
# Fill NaNs with the mean of each column df_filled_mean =
df.fillna(df.mean(numeric_only=True))#Handle 'na' entries
by replacing them with NaN df=df.dropna()
#Fill NaNs with a specific value 0
dfa=df.fillna(0)
print("\nDataset after filling missing values with the mean of each column:\n")
print(df_filled_mean)
print("\nDataset after filling missing values with 0: \n")
print(dfa)
print("\nDataset after replacing 'na' with NaN: \n")
print(df)

Output:

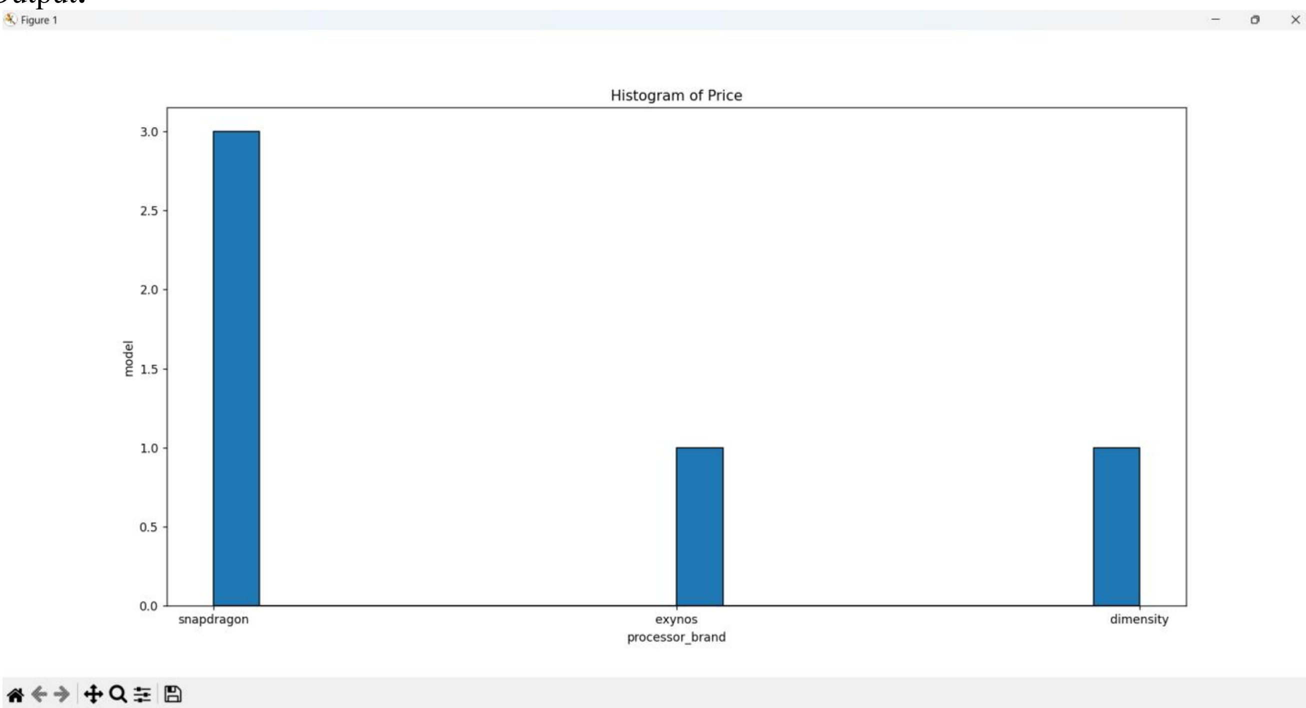
```
PS D:\ml> & C:/Users/student/AppData/Local/Programs/Python/Python312/python.exe d:/ml/p.py
TOTAL MISSING VALUES
brand_name                  0
model                       0
price                       0
rating                    101
has_5g                      0
has_nfc                     0
has_ir_blaster              0
processor_brand            20
num_cores                   6
processor_speed            42
battery_capacity           11
fast_charging_available     0
fast_charging             211
ram_capacity                0
internal_memory             0
screen_size                 0
refresh_rate                0
num_rear_cameras            0
num_front_cameras           4
os                         14
primary_camera_rear         0
primary_camera_front        5
extended_memory_available   0
extended_upto             480
resolution_width            0
resolution_height           0
dtype: int64

Dataset after filling missing values with the mean of each column:

    brand_name                      model  price  rating  has_5g  ...  primary_camera_front  extended_memory_available extended_upto  resolution_width  resolution_height
0      oneplus                OnePlus 11 5G  54999    89.0    True  ...                  16.0                          0       736.064              1440               3216
1      oneplus       OnePlus Nord CE 2 Lite 5G  19989    81.0    True  ...                  16.0                          1      1024.000              1080               2412
2      samsung          Samsung Galaxy A14 5G  16499    75.0    True  ...                  13.0                          1      1024.000              1080               2408
3     motorola          Motorola Moto G62 5G  14999    81.0    True  ...                  16.0                          1      1024.000              1080               2400
4       realme           Realme 10 Pro Plus  24999    82.0    True  ...                  16.0                          0       736.064              1080               2412
..         ...                          ...    ...     ...     ...  ...                   ...                        ...           ...               ...                ...
975   motorola      Motorola Moto Edge S30 Pro  34990    83.0    True  ...                  16.0                          0       736.064              1080               2460
976      honor                  Honor X8 5G  14990    75.0    True  ...                   8.0                          1      1024.000               720               1600
977       poco  POCO X4 GT 5G (8GB RAM + 256GB)  28990    85.0    True  ...                  16.0                          0       736.064              1080               2460
978   motorola          Motorola Moto G91 5G  19990    80.0    True  ...                  32.0                          1      1024.000              1080               2400
979    samsung         Samsung Galaxy M52s 5G  24990    74.0    True  ...                  32.0                          1      1024.000              1080               2400

[980 rows x 26 columns]

Dataset after filling missing values with 0:

    brand_name                              model  price  rating  ...  extended_memory_available  extended_upto  resolution_width resolution_height
1      oneplus              OnePlus Nord CE 2 Lite 5G  19989    81.0  ...                          1         1024.0              1080              2412
2      samsung                  Samsung Galaxy A14 5G  16499    75.0  ...                          1         1024.0              1080              2408
5      samsung  Samsung Galaxy F23 5G (6GB RAM + 128GB)  16999    80.0  ...                          1         1024.0              1080              2408
10      realme                        Realme 10 Pro  18999    82.0  ...                          1         1024.0              1080              2400
13        vivo          Vivo T1 5G (6GB RAM + 128GB)  16990    80.0  ...                          1         1024.0              1080              2408
```

```
..       ...                           ...    ...     ... ...                        ...       ...             ...              ...
944    realme                Realme Narzo 20  10499    72.0  ...                          1     256.0             720             1600
961      oppo                      OPPO A58x  13990    72.0  ...                          1    1024.0             720             1612
962    doogee                     Doogee S99  14999    84.0  ...                          1    1024.0            1080             2340
970    realme  Realme Narzo 50i Prime (4GB RAM + 64GB)   8720    64.0  ...                          1    1024.0             720             1600
976     honor                    Honor X8 5G  14990    75.0  ...                          1    1024.0             720             1600

[352 rows x 26 columns]

Dataset after replacing 'na' with NaN:

    brand_name                          model  price  rating  ...  extended_memory_available  extended_upto  resolution_width resolution_height
1      oneplus          OnePlus Nord CE 2 Lite 5G  19989    81.0  ...                          1    1024.0            1080             2412
2      samsung             Samsung Galaxy A14 5G  16499    75.0  ...                          1    1024.0            1080             2408
5      samsung  Samsung Galaxy F23 5G (6GB RAM + 128GB)  16999    80.0  ...                          1    1024.0            1080             2408
10      realme                  Realme 10 Pro  18999    82.0  ...                          1    1024.0            1080             2400
13        vivo        Vivo T1 5G (6GB RAM + 128GB)  16990    80.0  ...                          1    1024.0            1080             2408
..       ...                           ...    ...     ... ...                        ...       ...             ...              ...
944    realme                Realme Narzo 20  10499    72.0  ...                          1     256.0             720             1600
961      oppo                      OPPO A58x  13990    72.0  ...                          1    1024.0             720             1612
962    doogee                     Doogee S99  14999    84.0  ...                          1    1024.0            1080             2340
970    realme  Realme Narzo 50i Prime (4GB RAM + 64GB)   8720    64.0  ...                          1    1024.0             720             1600
976     honor                    Honor X8 5G  14990    75.0  ...                          1    1024.0             720             1600

[352 rows x 26 columns]
```

iii. Plot the histogram for continuous variables (at least two) to analyse the data.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
df=pd.read_csv("smartphones_cleaned_v6.csv")
plt.hist(df['processor_brand'].head(5), bins=20, edgecolor='black')
plt.ylabel('model')
plt.xlabel('processor_brand')
plt.title('Histogram of Price')
plt.show()
```

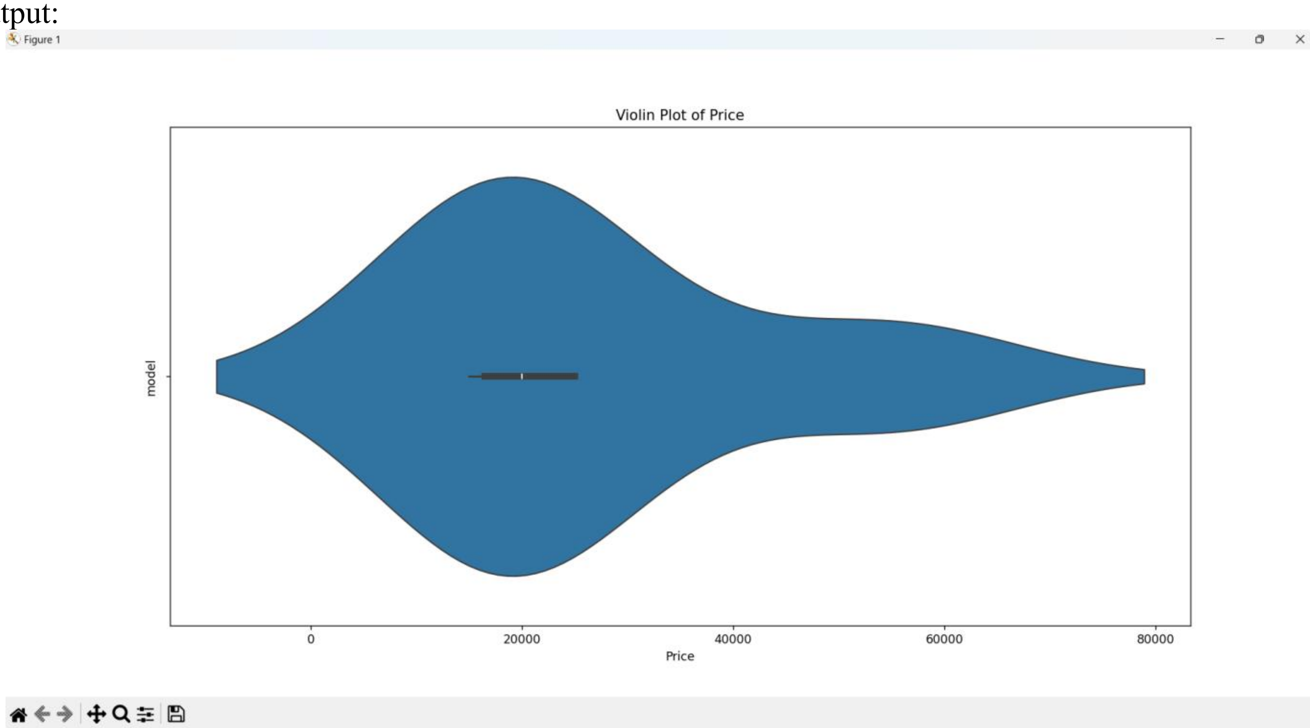
Output**:**



iv. Draw a violin plot do describe the distribution of a numerical variable to analyse the data.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
df=pd.read_csv("smartphones_cleaned_v6.csv")
plt.figure(figsize=(8,6))
sns.violinplot(x=df['price'].head(5)) plt.title('Violin
Plot of Price')
plt.xlabel('Price')
plt.ylabel('model')
plt.show()
```

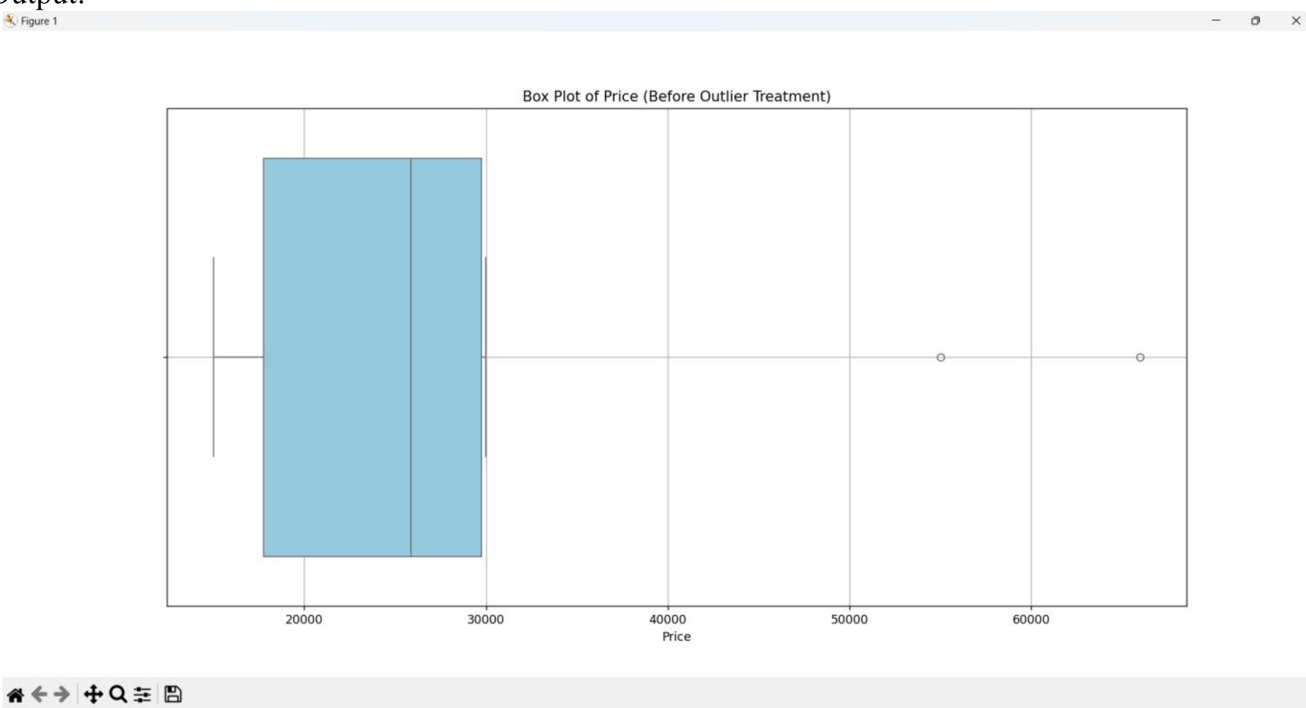
Output:



v. Recognize the outliers using box plot (Display the box plot before and after outlier treatment)

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
df=pd.read_csv("smartphones_cleaned_v6.csv")
plt.figure(figsize=(8, 6))
# sns.boxplot(x=df['price'].head(10), color='skyblue')
sns.pairplot(df.head(10))
plt.title('Box Plot of Price (Before Outlier Treatment)')
plt.xlabel('Price')
plt.grid(True)
plt.show()
```

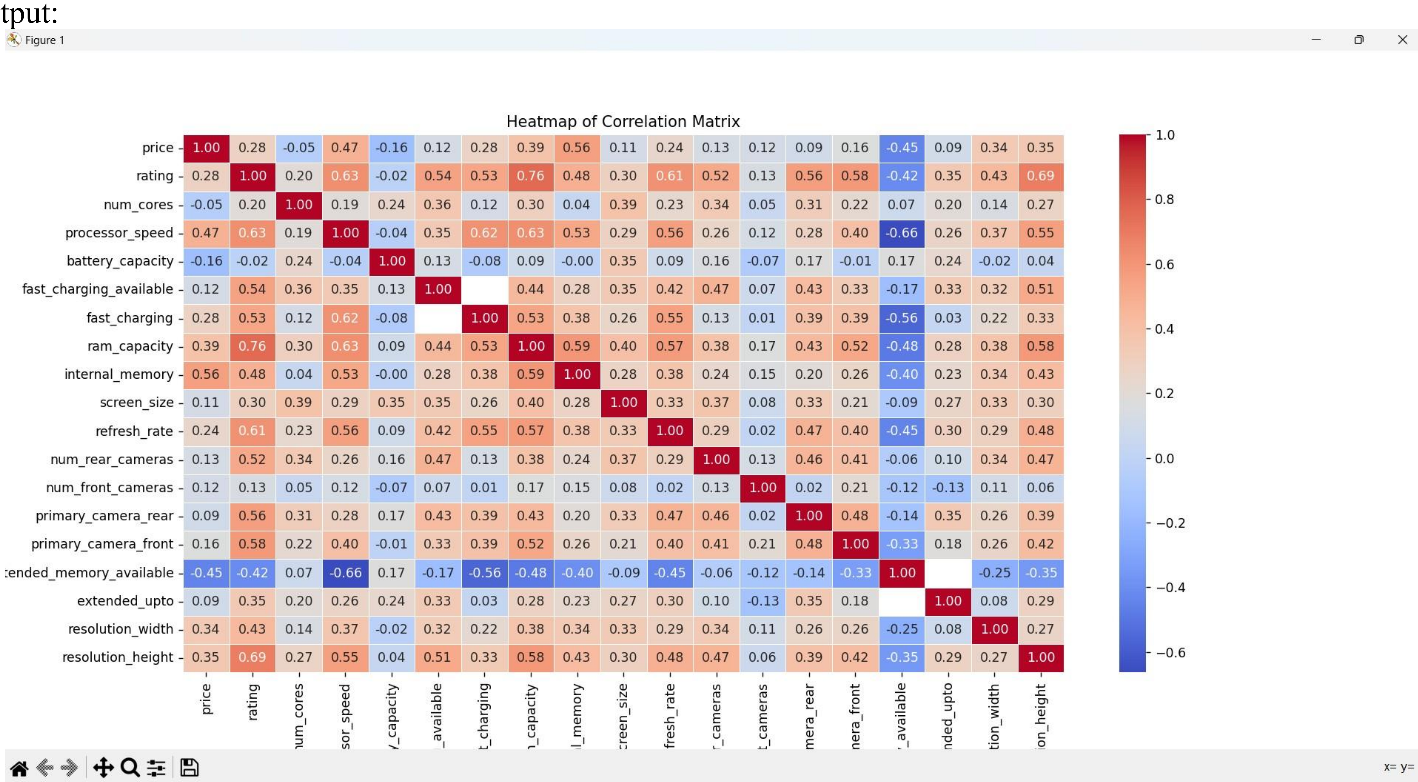
Output:



vi. Display a heat map to display the relationship among the attributes

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

df=pd.read_csv("smartphones_cleaned_v6.csv") numerical_df
= df.select_dtypes(include=['float64', 'int64'])
correlation_matrix = numerical_df.corr() plt.figure(figsize=(12,
10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f',
linewidths=0.5)
plt.title('Heatmap of Correlation Matrix')
plt.show()
```

Output:



vii. Standardize the continuous variable (if any)

```
import pandas as pd
from sklearn.preprocessing import StandardScalerdf
= pd.read_csv("smartphones_cleaned_v6.csv")
numerical_df = df.select_dtypes(include=['float64', 'int64'])
scaler = StandardScaler()
standardized_values = scaler.fit_transform(numerical_df)
standardized_df = pd.DataFrame(standardized_values, columns=numerical_df.columns)for
col in numerical_df.columns:
    df[col] = standardized_df[col]
print(df.head())
```

Output:

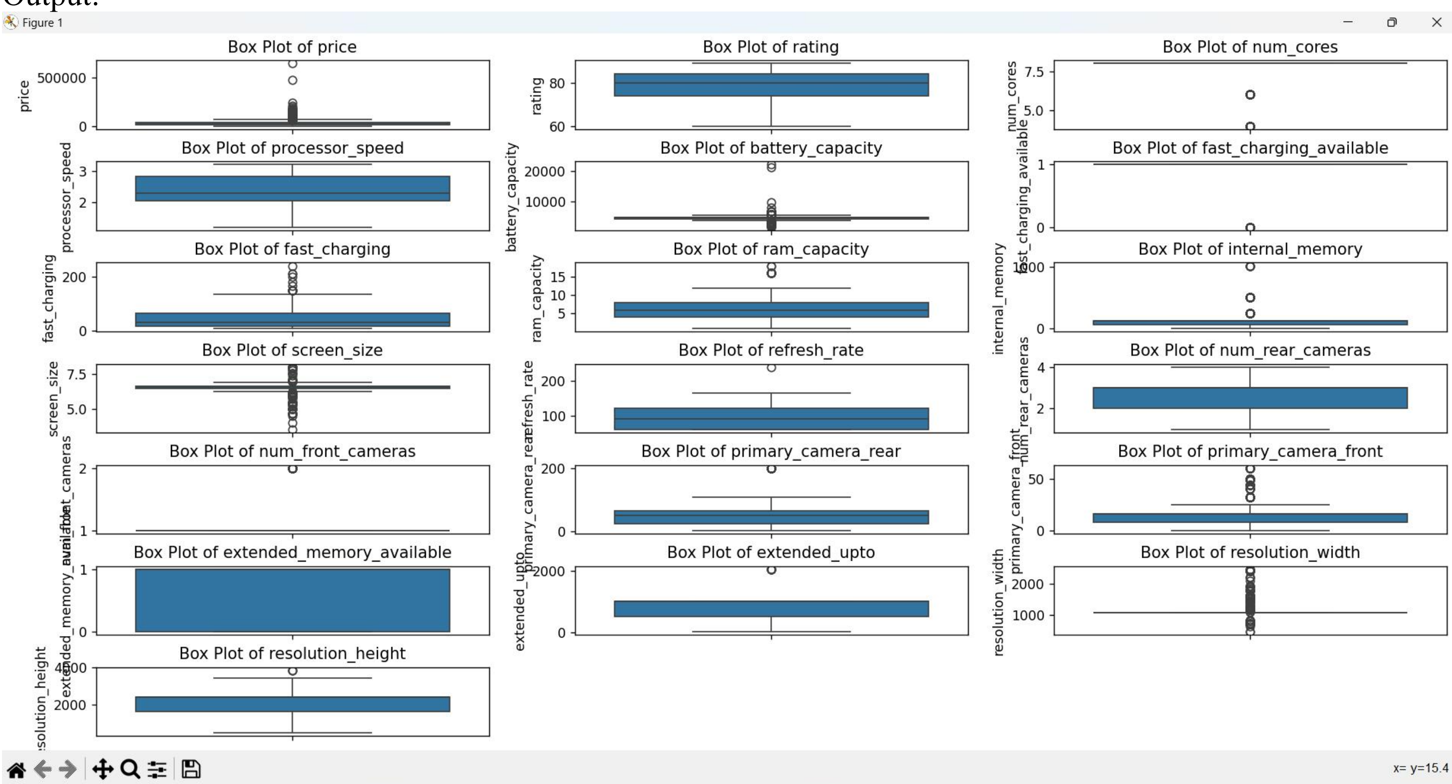
```
  brand_name                    model     price    rating  ...  extended_memory_available  extended_upto  resolution_width resolution_height
0    oneplus             OnePlus 11 5G  0.568908  1.451855  ...                  -1.306592            NaN          1.255610          1.939746
1    oneplus  OnePlus Nord CE 2 Lite 5G -0.317160  0.370575  ...                   0.765350       0.785577          0.014302          0.382272
2    samsung      Samsung Galaxy A14 5G -0.405488 -0.440385  ...                   0.765350       0.785577          0.014302          0.374523
3   motorola       Motorola Moto G62 5G -0.443452  0.370575  ...                   0.765350       0.785577          0.014302          0.359026
4     realme        Realme 10 Pro Plus -0.190362  0.505735  ...                  -1.306592            NaN          0.014302          0.382272
```

2. For the data set in Q1,

i. Show the distribution of continuous variables using Box Plot

```
import seaborn as sns
import pandas as pd

import matplotlib.pyplot as plt
df=pd.read_csv("smartphones_cleaned_v6.csv")

numerical_df = df.select_dtypes(include=['float64', 'int64'])
plt.figure(figsize=(15, 10))
for i, col in enumerate(numerical_df.columns):
 plt.subplot(len(numerical_df.columns) // 3 + 1, 3, i + 1)
 sns.boxplot(y=df[col])
 plt.title(f'Box Plot of {col}')
plt.tight_layout()
plt.show()
```

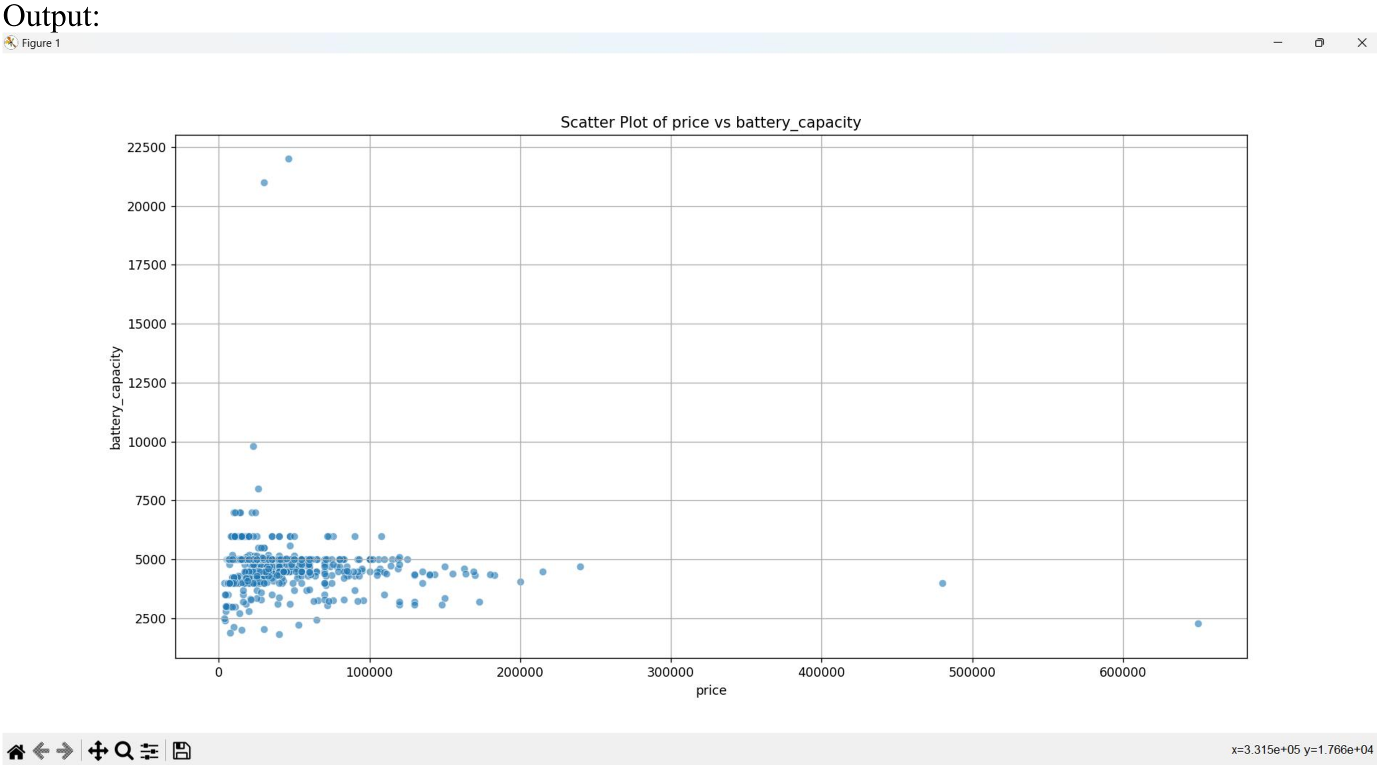
Output:



ii. Identify the relationship between two continuous variables using scatter plot

```
import pandas as pd
import matplotlib.pyplot as plt
df=pd.read_csv("smartphones_cleaned_v6.csv")
x_var = 'price' # Replace with your chosen variable
y_var = 'battery_capacity' # Replace with your chosen variable
# Create the scatter plot
plt.figure(figsize=(10, 6))
plt.scatter(df[x_var], df[y_var], alpha=0.6, edgecolors='w', linewidth=0.5)
plt.title(f'Scatter Plot of {x_var} vs {y_var}')
plt.xlabel(x_var)
plt.ylabel(y_var)
plt.grid(True)
# Show the plot
plt.show()
```

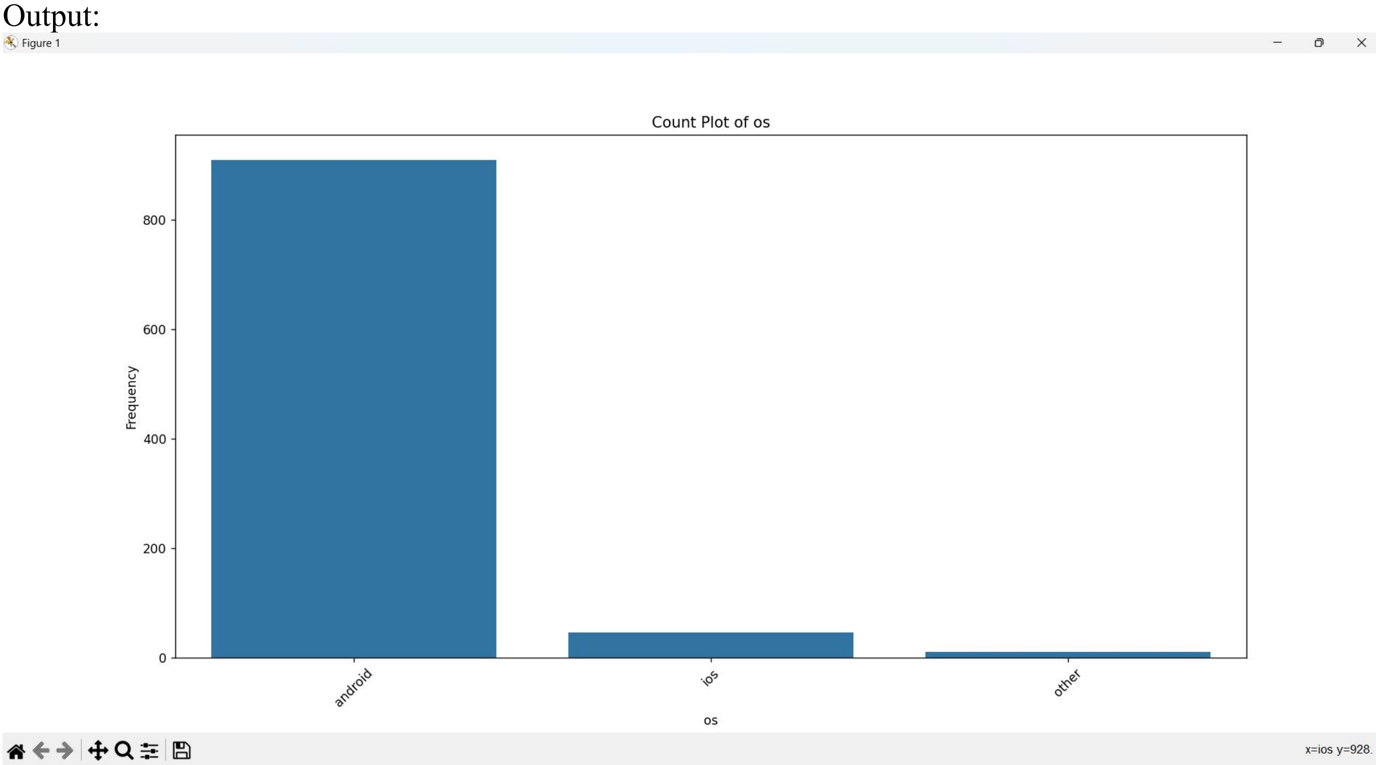
Output:



iii. Find and display the frequency of the categorical values using count plot

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
df=pd.read_csv("smartphones_cleaned_v6.csv")
categorical_var = 'os'
plt.figure(figsize=(10, 6))
sns.countplot(x=df[categorical_var],
order=df[categorical_var].value_counts().index)
plt.title(f'Count Plot of {categorical_var}')
plt.xlabel(categorical_var)
plt.ylabel('Frequency')
plt.xticks(rotation=45)
plt.show()
```

Output:



iv. Apply point plots to display one continuous and one categorical variable

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

df=pd.read_csv("smartphones_cleaned_v6.c
sv")continuous_var = 'price'
categorical_var
= 'os' # Create
the point plot
plt.figure(figsize=(10, 6))
sns.pointplot(x=categorical_var, y=continuous_var, data=df,
capsize=0.2,markers='o', linestyles='-')
plt.title(f'Point Plot of {continuous_var} vs
{categorical_var}')plt.xlabel(categorical_var)
plt.ylabel(continuo
us_var)
plt.xticks(rotation=
45) plt.show()
```

Output:

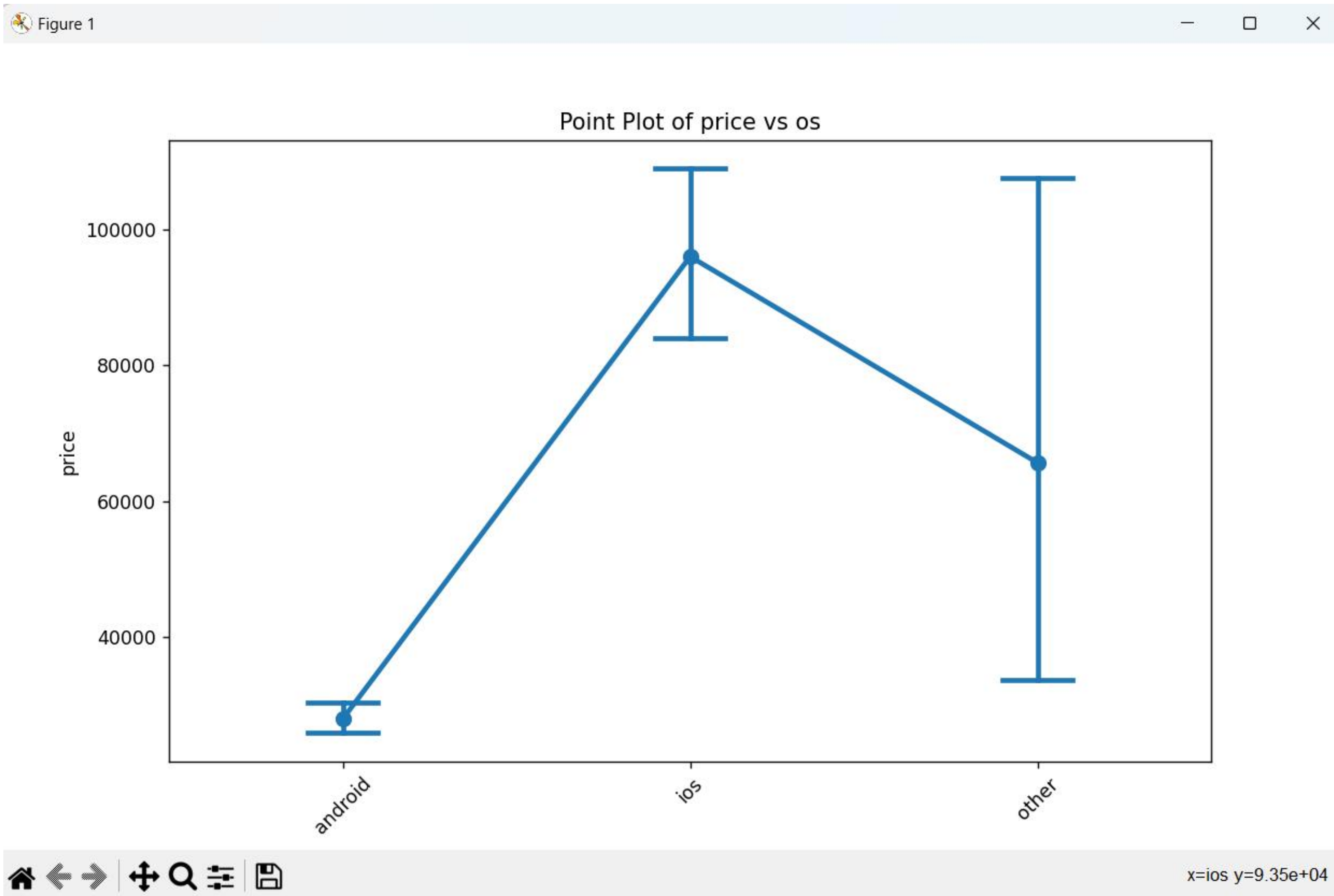


Figure 1
Point Plot of price vs os
price
100000
80000
60000
40000
android
ios
other
x=ios y=9.35e+04

3. For the Market-Basket dataset, apply Apriori algorithm and identify the best rules based on support and confidence.

```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules

# Load the dataset from CSV file
df = pd.read_csv(r"Groceries_dataset.csv")

# Split itemDescription column into individual items
df['itemDescription'] = df['itemDescription'].apply(lambda x: x.split('/'))

# Explode the itemDescription column into separate rows
df = df.explode('itemDescription')

# Group by Member_number and aggregate itemDescription into lists
transactions = df.groupby('Member_number')['itemDescription'].apply(list).tolist()

# Convert dataset to one-hot encoded DataFrame
te = TransactionEncoder()
te_ary = te.fit_transform(transactions)
df_encoded = pd.DataFrame(te_ary, columns=te.columns_)

# Apply Apriori algorithm to find frequent itemsets
frequent_itemsets = apriori(df_encoded, min_support=0.3, use_colnames=True)

# Generate association rules from the frequent itemsets
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.7)

# Print results
print("Frequent Itemsets:")
print(frequent_itemsets)

print("\nAssociation Rules:")
print(rules)
```

Output:

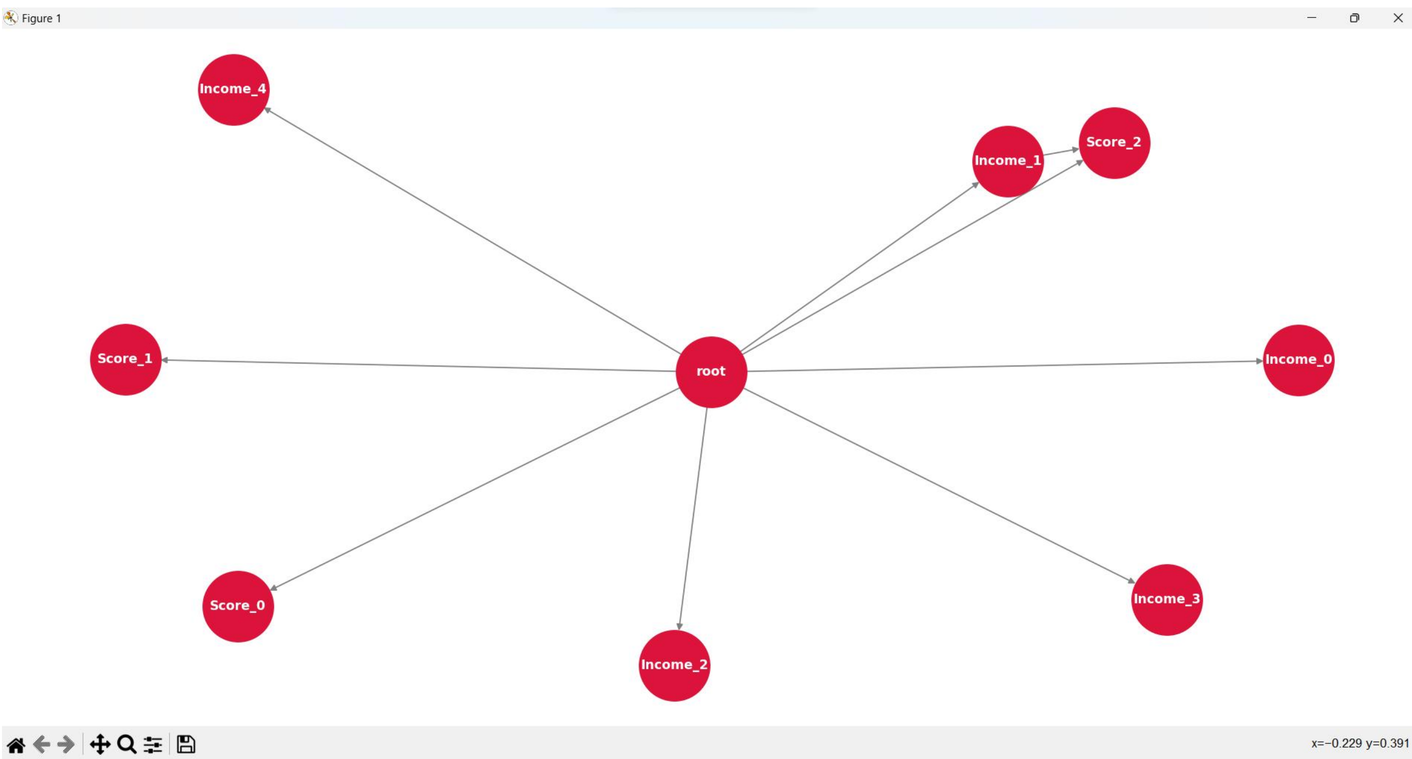
```
Frequent Itemsets:
    support           itemsets
0  0.349666             (buns)
1  0.376603  (other vegetables)
2  0.349666            (rolls)
3  0.313494             (soda)
4  0.458184       (whole milk)
5  0.349666      (rolls, buns)

Association Rules:
  antecedents consequents  antecedent support  consequent support   support  confidence      lift  leverage  conviction  zhangs_metric
0     (rolls)      (buns)            0.349666            0.349666  0.349666         1.0  2.859868    0.2274         inf            1.0
1      (buns)     (rolls)            0.349666            0.349666  0.349666         1.0  2.859868    0.2274         inf            1.0
PS D:\ml> 
```

4. For the data set given in Q3, apply FP-tree algorithm, show the tree construction and identify the best rules based on support and confidence.

```
import pandas as pd
import matplotlib.pyplot as plt
import networkx as nx
from mlxtend.frequent_patterns import fpgrowth
from mlxtend.preprocessing import TransactionEncoder
from sklearn.preprocessing import KBinsDiscretizer

df = pd.read_csv(r"C:\chithra\Mall_Customers.csv")
x = df.iloc[:, [2, 3]].values

kbins = KBinsDiscretizer(n_bins=5, encode='ordinal', strategy='uniform')
x_binned = kbins.fit_transform(x)
df_binned = pd.DataFrame(x_binned, columns=['Income_bin', 'Score_bin'])
print(df_binned.head())

transactions = df_binned.apply(lambda row: [f'Income_{int(row["Income_bin"])}',
f'Score_{int(row["Score_bin"])}'], axis=1).tolist()
te = TransactionEncoder()
te_ary = te.fit(transactions).transform(transactions)
df_encoded = pd.DataFrame(te_ary, columns=te.columns_)

frequent_itemsets = fpgrowth(df_encoded, min_support=0.1, use_colnames=True)
print("\n *******FP growth*******\n")
print(frequent_itemsets)

def construct_fp_tree(itemsets):
    tree = {}
    for itemset in itemsets['itemsets']:
        current_level = tree
        for item in sorted(itemset):
            if item not in current_level:
                current_level[item] = {}
            current_level = current_level[item]
    return tree

fp_tree = construct_fp_tree(frequent_itemsets)

def add_edges(graph, parent, children):
    for child, sub_children in children.items():
        graph.add_edge(parent, child)
        add_edges(graph, child, sub_children)

G = nx.DiGraph()
```

```
add_edges(G, 'root', fp_tree)
pos = nx.spring_layout(G, seed=42)  # positions for all nodes
plt.figure(figsize=(12, 10))
nx.draw(G, pos, with_labels=True, node_size=3000, node_color='crimson', font_size=10, font_weight='bold',
edge_color='gray',font_color='white')
plt.title('FP-Tree Visualization')
plt.show()
```

Output:



```
   Income_bin  Score_bin
0         0.0        0.0
1         0.0        0.0
2         0.0        0.0
3         0.0        0.0
4         1.0        0.0

 *******FP growth*******

   support             itemsets
0    0.250           (Income_0)
1    0.230            (Score_0)
2    0.315           (Income_1)
3    0.100           (Income_4)
4    0.125           (Income_3)
5    0.210           (Income_2)
6    0.330            (Score_1)
7    0.330            (Score_2)
8    0.135  (Score_2, Income_1)

   support             itemsets
0    0.250           (Income_0)
1    0.230            (Score_0)
2    0.315           (Income_1)
3    0.100           (Income_4)
4    0.125           (Income_3)
5    0.210           (Income_2)
6    0.330            (Score_1)

   support             itemsets
0    0.250           (Income_0)
1    0.230            (Score_0)
2    0.315           (Income_1)
3    0.100           (Income_4)

   support             itemsets
0    0.250           (Income_0)
1    0.230            (Score_0)
2    0.315           (Income_1)

   support             itemsets
0    0.250           (Income_0)
0    0.250           (Income_0)
1    0.230            (Score_0)
2    0.315           (Income_1)
2    0.315           (Income_1)
3    0.100           (Income_4)
4    0.125           (Income_3)
5    0.210           (Income_2)
5    0.210           (Income_2)
6    0.330            (Score_1)
7    0.330            (Score_2)
6    0.330            (Score_1)
7    0.330            (Score_2)
7    0.330            (Score_2)
8    0.135  (Score_2, Income_1)
8    0.135  (Score_2, Income_1)
```

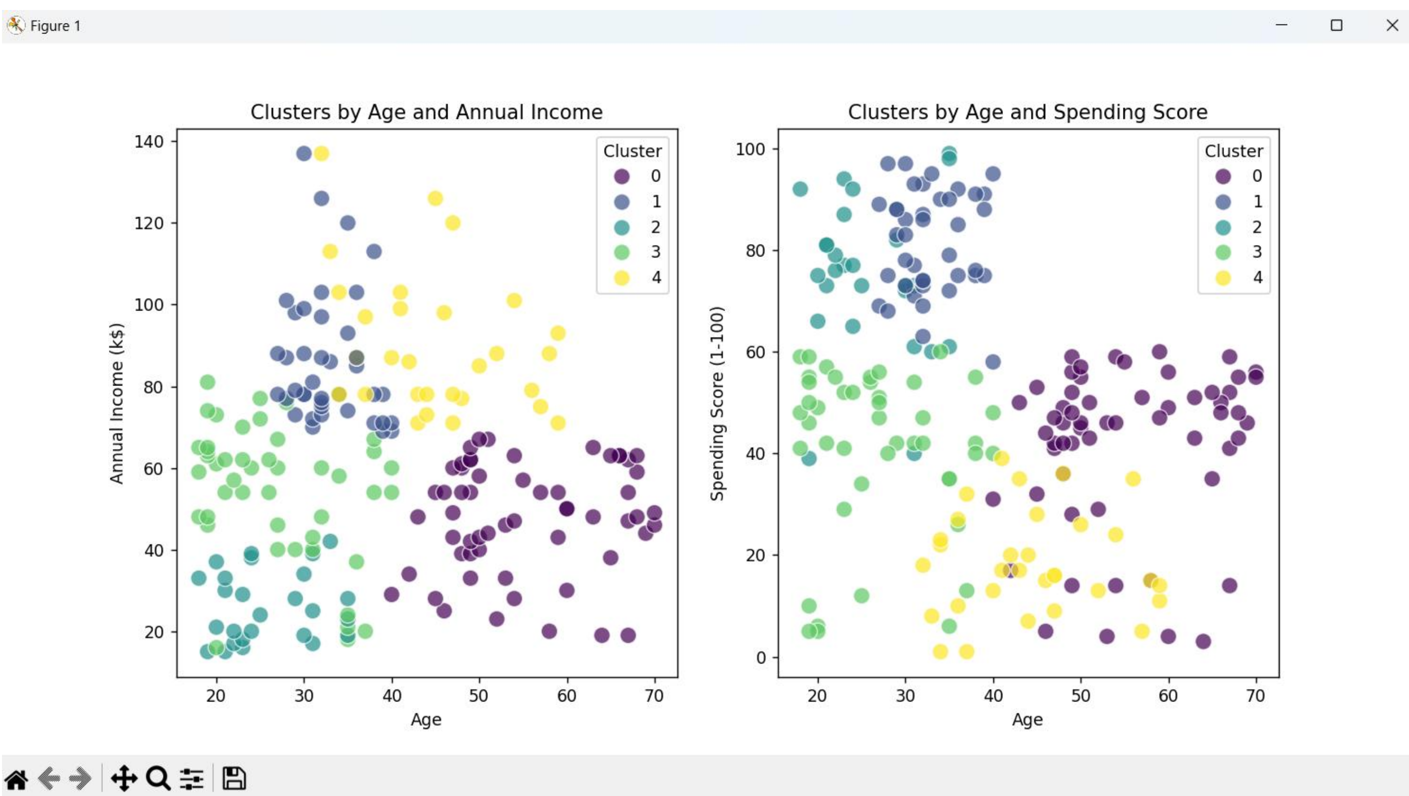
5. For the Mall-Customer data set, implement K-means clustering algorithm and visualize the clusters

```
import pandas as pd
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
file_path = (r"C:\Users\manju\OneDrive\Desktop\MCA\2nd SEM\ML\Lab Programs\Mall_Customers.csv")
# Replace with the path to your dataset
df = pd.read_csv(file_path)

# Display the first few rows of the dataframe
print(df.head())

# Preprocess the data
# Encode categorical variables
label_encoder = LabelEncoder()
df['Gender'] = label_encoder.fit_transform(df['Gender'])

# Select features for clustering
features = df[['Age', 'Annual Income (k$)', 'Spending Score (1-100)']]

# Feature Scaling
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)

# Apply K-means clustering
# Define the number of clusters (you can adjust this based on your needs)
kmeans = KMeans(n_clusters=5, random_state=42)
df['Cluster'] = kmeans.fit_predict(scaled_features)

# Visualize the clustering result
# 2D scatter plots for visualization
plt.figure(figsize=(12, 6))

# Plot Age vs Annual Income
plt.subplot(1, 2, 1)
sns.scatterplot(x='Age', y='Annual Income (k$)', hue='Cluster', palette='viridis', data=df, s=100, alpha=0.7)
plt.title('Clusters by Age and Annual Income')

# Plot Age vs Spending Score
plt.subplot(1, 2, 2)
sns.scatterplot(x='Age', y='Spending Score (1-100)', hue='Cluster', palette='viridis', data=df, s=100, alpha=0.7)
```

```
plt.title('Clusters by Age and Spending Score')

plt.show()

# Display the cluster centers
cluster_centers = scaler.inverse_transform(kmeans.cluster_centers_)
print("Cluster Centers (original scale):")
print(pd.DataFrame(cluster_centers, columns=['Age', 'Annual Income (k$)', 'Spending Score (1-100)']))

# Display the number of customers in each cluster
print("\nNumber of customers in each cluster:")
print(df['Cluster'].value_counts())
```

Output:

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40



6. For the Groceries dataset implement Agglomerative clustering algorithm and visualize the clusters.

```
# Hierarchial Clustering
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import pairwise_distances
import scipy.cluster.hierarchy as sch
from scipy.spatial.distance import squareform
import matplotlib.pyplot as plt

df = pd.read_csv("C:/Users/manju/OneDrive/Desktop/MCA/2nd SEM/ML/Lab Programs/Groceries_dataset
(Apriori).csv")

df['itemDescription'] = df['itemDescription'].apply(lambda x: x.split('/'))
df = df.explode('itemDescription')
transactions = df.groupby('Member_number')['itemDescription'].apply(list).tolist()

te = TransactionEncoder()
te_ary = te.fit_transform(transactions)
df_encoded = pd.DataFrame(te_ary, columns=te.columns_)

distance_matrix = pairwise_distances(df_encoded, metric='euclidean')
condensed_distance_matrix = squareform(distance_matrix)

linkage_matrix = sch.linkage(condensed_distance_matrix, method='ward')

plt.figure(figsize=(10, 7))
sch.dendrogram(linkage_matrix)
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Member Number')
plt.ylabel('Euclidean Distance')
plt.show()

num_clusters = 5
cluster_model = AgglomerativeClustering(n_clusters=num_clusters, affinity='euclidean', linkage='ward')
clusters = cluster_model.fit_predict(df_encoded)

df_clusters = pd.DataFrame({'Member_number': df['Member_number'].unique(), 'Cluster': clusters})
print("\nCluster Assignments:")
print(df_clusters)
```

Output:

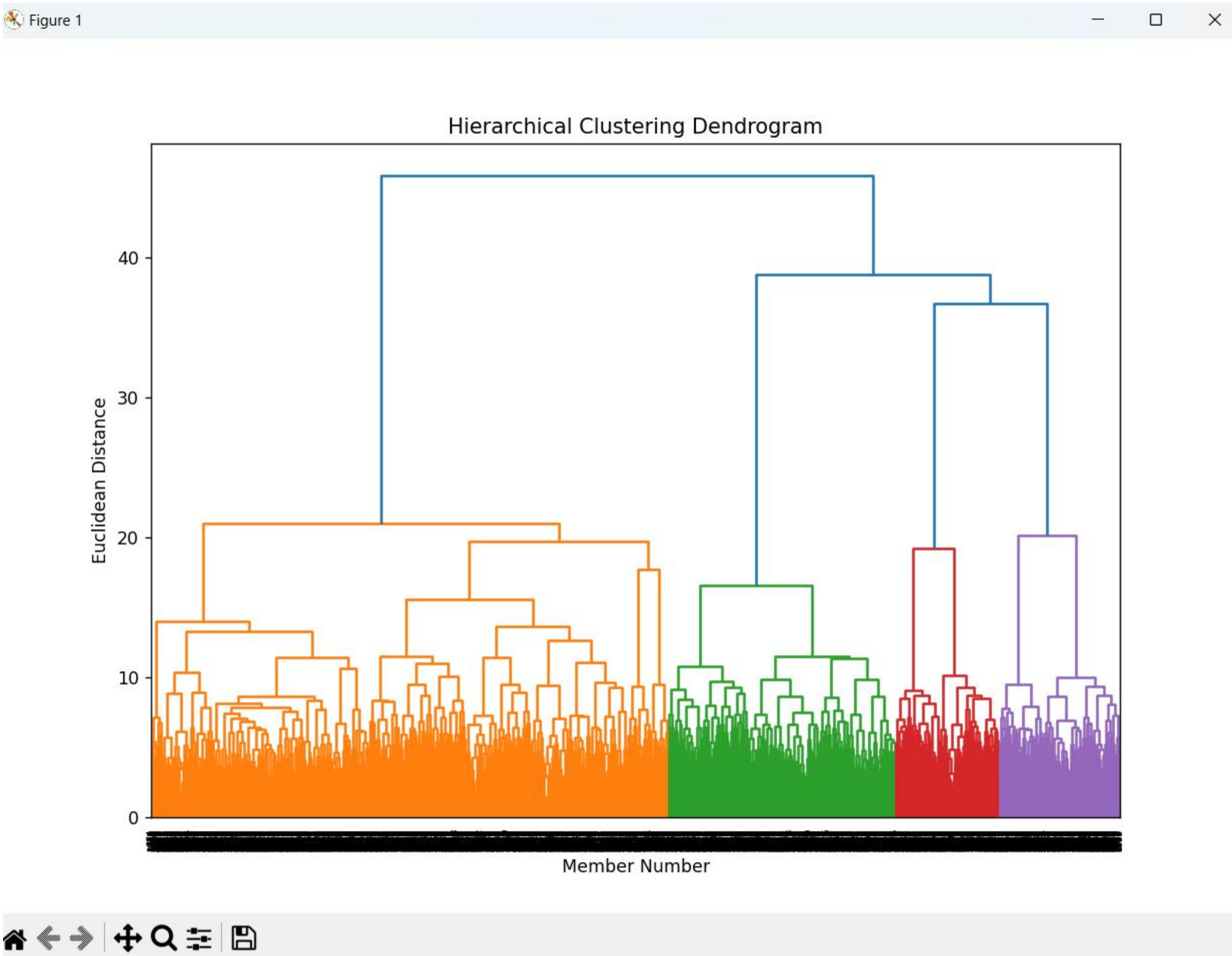


Figure 1
Hierarchical Clustering Dendrogram
Euclidean Distance
0
10
20
30
40
Member Number

7. For the Mall_Customers implement DBScan clustering algorithm and visualize the clusters.

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import DBSCAN
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset (assuming it's a CSV file)
# You can adjust the file path as per your requirement
dataset = pd.read_csv("C:/Users/manju/OneDrive/Desktop/MCA/2nd SEM/ML/Lab
Programs/Mall_Customers.csv")

# Display the first few rows of the dataset
print(dataset.head())

# Preprocessing: Convert 'Gender' to numerical values (Male=0, Female=1)
dataset['Gender'] = dataset['Gender'].map({'Male': 0, 'Female': 1})

# Selecting features for clustering (Age, Annual Income, Spending Score)
X = dataset[['Age', 'Annual Income (k$)', 'Spending Score (1-100)']]

# Standardize the features to bring them to the same scale
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Applying DBSCAN
dbscan = DBSCAN(eps=0.5, min_samples=5)
dbscan.fit(X_scaled)

# Getting the cluster labels
labels = dbscan.labels_

# Adding the cluster labels to the dataset
dataset['Cluster'] = labels

# Visualizing the clusters
plt.figure(figsize=(10, 6))
sns.scatterplot(data=dataset, x='Annual Income (k$)', y='Spending Score (1-100)', hue='Cluster', palette='Set1',
s=100)
plt.title('DBSCAN Clustering of Mall Customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend(loc='best')
plt.show()
```

Output:

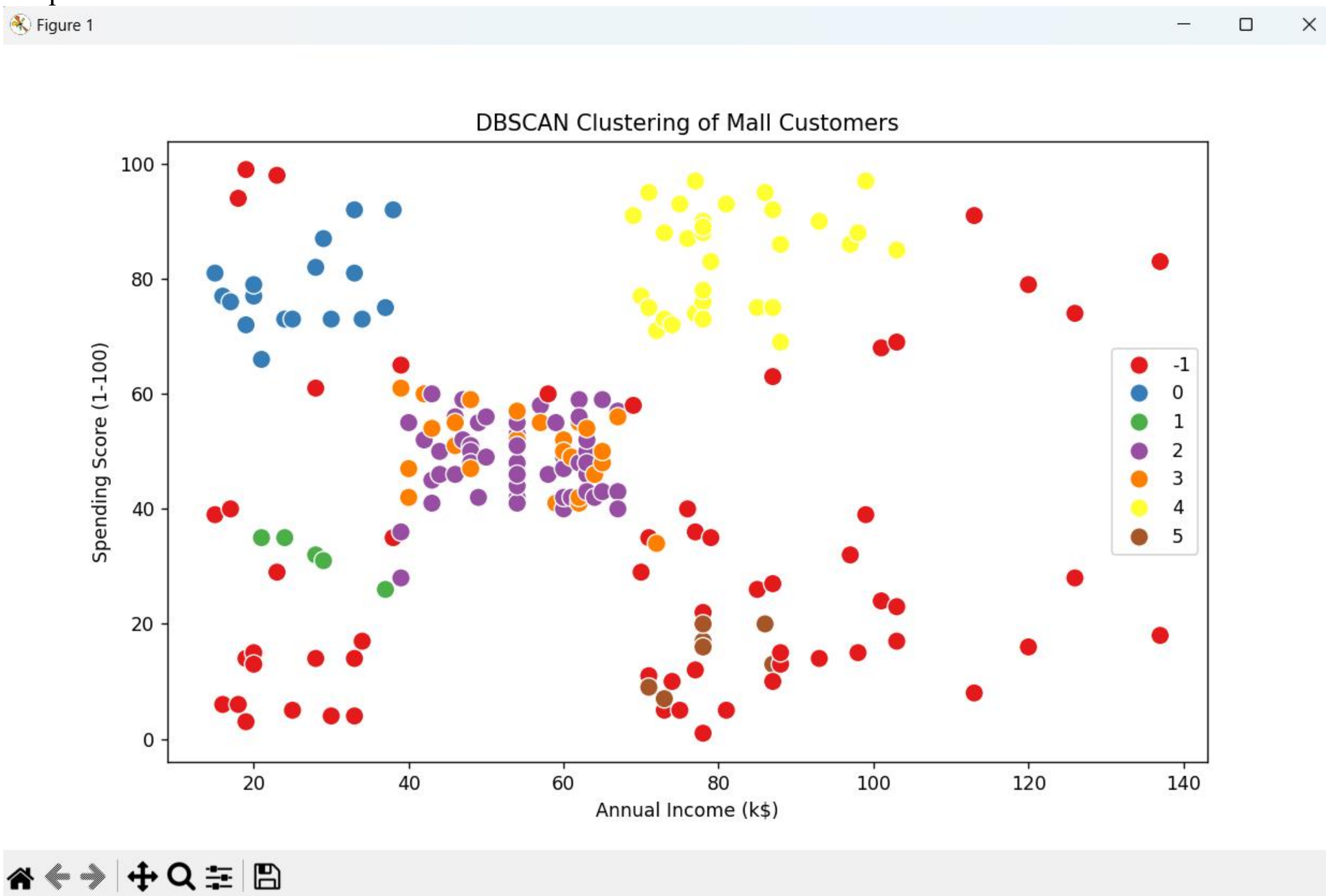


Figure 1
DBSCAN Clustering of Mall Customers
Spending Score (1-100)
Annual Income (k$)
0
20
40
60
80
100
120
140
-1
1
2
3
4
5

8. Implement KNN Classification algorithm on the Mall Customers. Analyse the model using different K values and display the performance of the model.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt

# Load the dataset (assuming it's a CSV file)
dataset = pd.read_csv("C:/Users/manju/OneDrive/Desktop/MCA/2nd SEM/ML/Lab
Programs/Mall_Customers.csv")

# Preprocessing: Convert 'Gender' to numerical values (Male=0, Female=1)
dataset['Gender'] = dataset['Gender'].map({'Male': 0, 'Female': 1})

# Select features (Age, Annual Income, Spending Score) and target (Gender)
X = dataset[['Age', 'Annual Income (k$)', 'Spending Score (1-100)']]
y = dataset['Gender']

# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=42)

# Analyze the model for different values of K
k_values = range(1, 21)  # We'll analyze for K from 1 to 20
accuracies = []

for k in k_values:
    # Initialize KNN with current K value
    knn = KNeighborsClassifier(n_neighbors=k)

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

    # Make predictions
    y_pred = knn.predict(X_test)

    # Calculate accuracy and store it
    acc = accuracy_score(y_test, y_pred)
    accuracies.append(acc)
```

```
    # Display performance metrics for the current K
    print(f"\nPerformance for K={k}:")
    print(f"Accuracy: {acc:.4f}")
    print("Confusion Matrix:")
    print(confusion_matrix(y_test, y_pred))
    print("Classification Report:")
    print(classification_report(y_test, y_pred))

# Visualizing the performance across different K values
plt.figure(figsize=(10, 6))
plt.plot(k_values, accuracies, marker='o', linestyle='-', color='b')
plt.title('KNN Model Accuracy for Different K Values')
plt.xlabel('K Value')
plt.ylabel('Accuracy')
plt.xticks(k_values)
plt.grid(True)
plt.show()
```

Output:

```
Performance for K=1:
Accuracy: 0.6000
Confusion Matrix:
[[14 10]
 [14 22]]
Classification Report:
              precision    recall  f1-score   support

           0       0.50      0.58      0.54        24
           1       0.69      0.61      0.65        36

    accuracy                           0.60        60
   macro avg       0.59      0.60      0.59        60
weighted avg       0.61      0.60      0.60        60
```

```
Performance for K=2:
Accuracy: 0.5333
Confusion Matrix:
[[20  4]
 [24 12]]
Classification Report:
              precision    recall  f1-score   support

           0       0.45      0.83      0.59        24
           1       0.75      0.33      0.46        36

    accuracy                           0.53        60
   macro avg       0.60      0.58      0.52        60
weighted avg       0.63      0.53      0.51        60
```

```
Performance for K=3:
Accuracy: 0.5333
Confusion Matrix:
[[ 9 15]
 [13 23]]
Classification Report:
              precision    recall  f1-score   support

           0       0.41      0.38      0.39        24
           1       0.61      0.64      0.62        36

    accuracy                           0.53        60
   macro avg       0.51      0.51      0.51        60
weighted avg       0.53      0.53      0.53        60
```

```
Performance for K=4:
Accuracy: 0.5000
Confusion Matrix:
[[16  8]
 [22 14]]
Classification Report:
              precision    recall  f1-score   support

           0       0.42      0.67      0.52        24
           1       0.64      0.39      0.48        36

    accuracy                           0.50        60
   macro avg       0.53      0.53      0.50        60
weighted avg       0.55      0.50      0.50        60
```

```
Performance for K=5:
Accuracy: 0.5000
Confusion Matrix:
[[ 9 15]
 [15 21]]
Classification Report:
              precision    recall  f1-score   support

           0       0.38      0.38      0.38        24
           1       0.58      0.58      0.58        36

    accuracy                           0.50        60
   macro avg       0.48      0.48      0.48        60
weighted avg       0.50      0.50      0.50        60
```

```
Performance for K=6:
Accuracy: 0.4667
Confusion Matrix:
[[13 11]
 [21 15]]
Classification Report:
              precision    recall  f1-score   support

           0       0.38      0.54      0.45        24
           1       0.58      0.42      0.48        36

    accuracy                           0.47        60
   macro avg       0.48      0.48      0.47        60
weighted avg       0.50      0.47      0.47        60
```

```
Performance for K=7:
Accuracy: 0.5667
Confusion Matrix:
[[ 6 18]
 [ 8 28]]
Classification Report:
              precision    recall  f1-score   support

           0       0.43      0.25      0.32        24
           1       0.61      0.78      0.68        36

    accuracy                           0.57        60
   macro avg       0.52      0.51      0.50        60
weighted avg       0.54      0.57      0.54        60
```

```
Performance for K=8:
Accuracy: 0.5000
Confusion Matrix:
[[10 14]
 [16 20]]
Classification Report:
              precision    recall  f1-score   support

           0       0.38      0.42      0.40        24
           1       0.59      0.56      0.57        36

    accuracy                           0.50        60
   macro avg       0.49      0.49      0.49        60
weighted avg       0.51      0.50      0.50        60
```

```
Performance for K=9:
Accuracy: 0.6000
Confusion Matrix:
[[ 6 18]
 [ 6 30]]
Classification Report:
              precision    recall  f1-score   support

           0       0.50      0.25      0.33        24
           1       0.62      0.83      0.71        36

    accuracy                           0.60        60
   macro avg       0.56      0.54      0.52        60
weighted avg       0.57      0.60      0.56        60
```

```
Performance for K=10:
Accuracy: 0.4667
Confusion Matrix:
[[ 8 16]
 [16 20]]
Classification Report:
              precision    recall  f1-score   support

           0       0.33      0.33      0.33        24
           1       0.56      0.56      0.56        36

    accuracy                           0.47        60
   macro avg       0.44      0.44      0.44        60
weighted avg       0.47      0.47      0.47        60
```

```
Performance for K=11:
Accuracy: 0.4833
Confusion Matrix:
[[ 4 20]
 [11 25]]
Classification Report:
              precision    recall  f1-score   support

           0       0.27      0.17      0.21        24
           1       0.56      0.69      0.62        36

    accuracy                           0.48        60
   macro avg       0.41      0.43      0.41        60
weighted avg       0.44      0.48      0.45        60
```

```
Performance for K=12:
Accuracy: 0.5000
Confusion Matrix:
[[ 9 15]
 [15 21]]
Classification Report:
              precision    recall  f1-score   support

           0       0.38      0.38      0.38        24
           1       0.58      0.58      0.58        36

    accuracy                           0.50        60
   macro avg       0.48      0.48      0.48        60
weighted avg       0.50      0.50      0.50        60
```

```
Performance for K=13:
Accuracy: 0.5833
Confusion Matrix:
[[ 4 20]
 [ 5 31]]
Classification Report:
              precision    recall  f1-score   support

           0       0.44      0.17      0.24        24
           1       0.61      0.86      0.71        36

    accuracy                           0.58        60
   macro avg       0.53      0.51      0.48        60
weighted avg       0.54      0.58      0.52        60
```

```
Performance for K=14:
Accuracy: 0.5833
Confusion Matrix:
[[ 8 16]
 [ 9 27]]
Classification Report:
              precision    recall  f1-score   support

           0       0.47      0.33      0.39        24
           1       0.63      0.75      0.68        36

    accuracy                           0.58        60
   macro avg       0.55      0.54      0.54        60
weighted avg       0.56      0.58      0.57        60
```

```
Performance for K=15:
Accuracy: 0.6000
Confusion Matrix:
[[ 6 18]
 [ 6 30]]
Classification Report:
              precision    recall  f1-score   support

           0       0.50      0.25      0.33        24
           1       0.62      0.83      0.71        36

    accuracy                           0.60        60
   macro avg       0.56      0.54      0.52        60
weighted avg       0.57      0.60      0.56        60
```

```
Performance for K=16:
Accuracy: 0.5833
Confusion Matrix:
[[ 7 17]
 [ 8 28]]
Classification Report:
              precision    recall  f1-score   support

           0       0.47      0.29      0.36        24
           1       0.62      0.78      0.69        36

    accuracy                           0.58        60
   macro avg       0.54      0.53      0.53        60
weighted avg       0.56      0.58      0.56        60
```

```
Performance for K=17:
Accuracy: 0.6000
Confusion Matrix:
[[ 5 19]
 [ 5 31]]
Classification Report:
              precision    recall  f1-score   support

           0       0.50      0.21      0.29        24
           1       0.62      0.86      0.72        36

    accuracy                           0.60        60
   macro avg       0.56      0.53      0.51        60
weighted avg       0.57      0.60      0.55        60
```

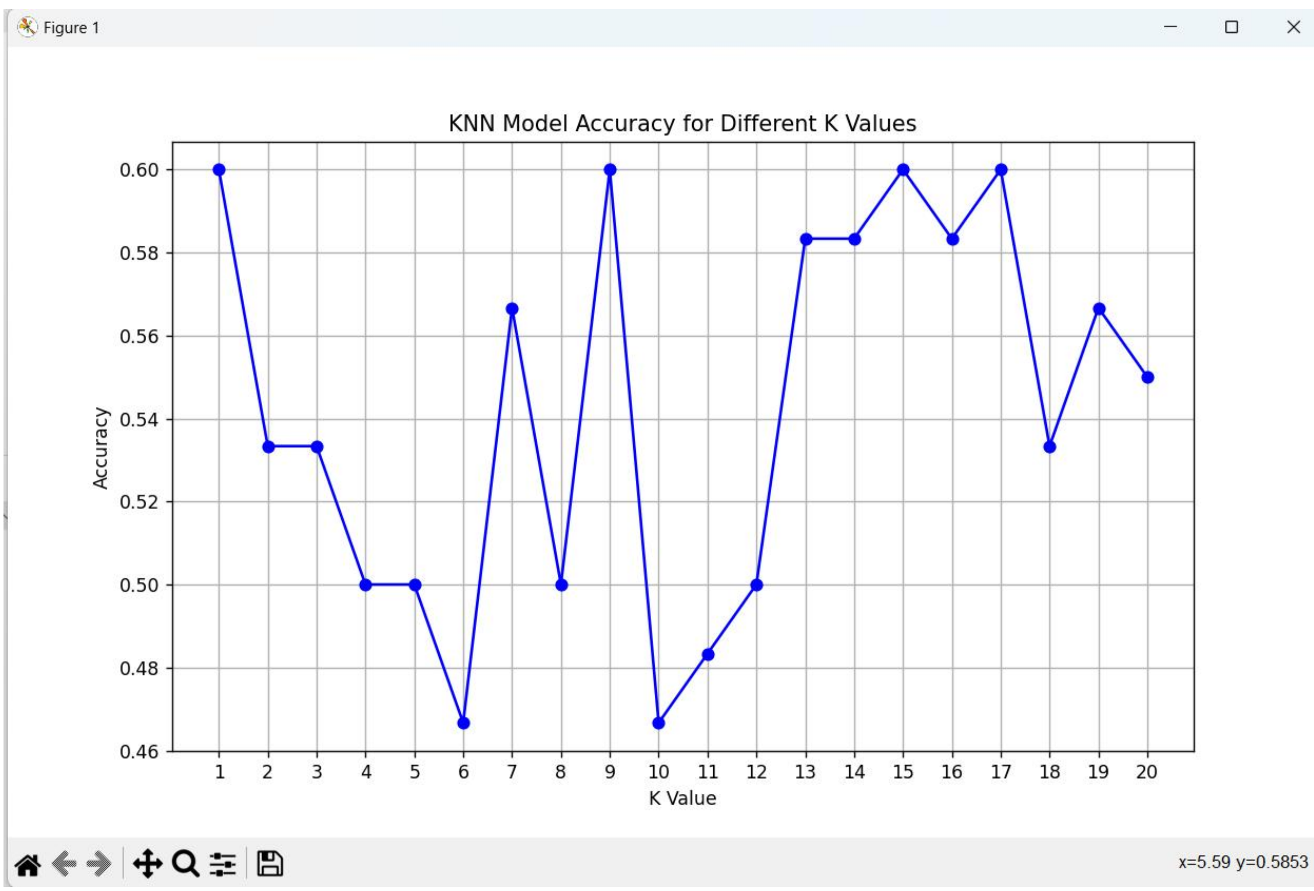
```
Performance for K=18:
Accuracy: 0.5333
Confusion Matrix:
[[ 6 18]
 [10 26]]
Classification Report:
              precision    recall  f1-score   support

           0       0.38      0.25      0.30        24
           1       0.59      0.72      0.65        36

    accuracy                           0.53        60
   macro avg       0.48      0.49      0.47        60
weighted avg       0.50      0.53      0.51        60
```

```
Performance for K=19:
Accuracy: 0.5667
Confusion Matrix:
[[ 4 20]
 [ 6 30]]
Classification Report:
              precision    recall  f1-score   support

           0       0.40      0.17      0.24        24
           1       0.60      0.83      0.70        36

    accuracy                           0.57        60
   macro avg       0.50      0.50      0.47        60
weighted avg       0.52      0.57      0.51        60
```

```
Performance for K=20:
Accuracy: 0.5500
Confusion Matrix:
[[ 5 19]
 [ 8 28]]
Classification Report:
              precision    recall  f1-score   support
weighted avg       0.52      0.57      0.51        60
```

```
Performance for K=20:
Accuracy: 0.5500
Confusion Matrix:
[[ 5 19]
 [ 8 28]]
weighted avg       0.52      0.57      0.51        60


Performance for K=20:
Accuracy: 0.5500
Confusion Matrix:
weighted avg       0.52      0.57      0.51        60


Performance for K=20:
Accuracy: 0.5500
weighted avg       0.52      0.57      0.51        60


Performance for K=20:
weighted avg       0.52      0.57      0.51        60


weighted avg       0.52      0.57      0.51        60

weighted avg       0.52      0.57      0.51        60
weighted avg       0.52      0.57      0.51        60
```

```
Performance for K=20:
Accuracy: 0.5500
Confusion Matrix:
[[ 5 19]
weighted avg       0.52      0.57      0.51        60


Performance for K=20:
Accuracy: 0.5500
Confusion Matrix:
[[ 5 19]
 [ 8 28]]
Classification Report:


Performance for K=20:
Accuracy: 0.5500
Confusion Matrix:
[[ 5 19]
Performance for K=20:
Accuracy: 0.5500
Confusion Matrix:
[[ 5 19]
[[ 5 19]
 [ 8 28]]
Classification Report:
              precision    recall  f1-score   support

           0       0.38      0.21      0.27        24
           1       0.60      0.78      0.67        36

    accuracy                           0.55        60
   macro avg       0.49      0.49      0.47        60
weighted avg       0.51      0.55      0.51        60
```



9. Implement Naïve Bayes Classification algorithm on the Online Retail. Analyse the efficiency of the algorithm using different metrics.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset (assuming it's a CSV file)
dataset = pd.read_csv("C:/Users/manju/OneDrive/Desktop/MCA/2nd SEM/ML/Lab
Programs/Online_Retail.csv", encoding='latin1')

# Display the first few rows of the dataset
print(dataset.head())

# Step 1: Data Preprocessing
# Dropping rows with missing CustomerID as we need it for classification
dataset = dataset.dropna(subset=['CustomerID'])

# Selecting features for classification
# Let's assume we are classifying based on the Country where the customer is from
# We'll use 'Quantity', 'UnitPrice', and 'InvoiceDate' features (with feature engineering)
dataset['InvoiceDate'] = pd.to_datetime(dataset['InvoiceDate'])
dataset['InvoiceDay'] = dataset['InvoiceDate'].dt.day
dataset['InvoiceMonth'] = dataset['InvoiceDate'].dt.month
dataset['InvoiceHour'] = dataset['InvoiceDate'].dt.hour

# Select relevant features and target (Country)
X = dataset[['Quantity', 'UnitPrice', 'InvoiceDay', 'InvoiceMonth', 'InvoiceHour']]
y = dataset['Country']

# Step 2: Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Step 3: Applying Naive Bayes Classifier (GaussianNB for continuous features)
model = GaussianNB()

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

# Make predictions on the test data
y_pred = model.predict(X_test)

# Step 4: Analyzing Model Efficiency

# Accuracy score
accuracy = accuracy_score(y_test, y_pred)
```

```
print(f"Accuracy: {accuracy:.4f}")

# Classification Report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Confusion Matrix
print("\nConfusion Matrix:")
conf_matrix = confusion_matrix(y_test, y_pred)
print(conf_matrix)

# Plotting Confusion Matrix
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=model.classes_,
yticklabels=model.classes_)
plt.title('Confusion Matrix of Naive Bayes Classification')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

Output:

```
  InvoiceNo StockCode                          Description  Quantity          InvoiceDate  UnitPrice  CustomerID         Country
0    536365    85123A   WHITE HANGING HEART T-LIGHT HOLDER         6  2010-12-01 08:26:00       2.55     17850.0  United Kingdom
1    536365     71053                  WHITE METAL LANTERN         6  2010-12-01 08:26:00       3.39     17850.0  United Kingdom
2    536365    84406B       CREAM CUPID HEARTS COAT HANGER         8  2010-12-01 08:26:00       2.75     17850.0  United Kingdom
3    536365    84029G  KNITTED UNION FLAG HOT WATER BOTTLE         6  2010-12-01 08:26:00       3.39     17850.0  United Kingdom
4    536365    84029E       RED WOOLLY HOTTIE WHITE HEART.         6  2010-12-01 08:26:00       3.39     17850.0  United Kingdom
Accuracy: 0.0766
```

```
                      precision    recall  f1-score   support

           Australia       0.00      0.00      0.00       370
             Austria       0.00      0.00      0.00       116
             Bahrain       0.00      0.60      0.00         5
             Belgium       0.01      0.09      0.01       625
              Brazil       0.27      1.00      0.43         9
              Canada       0.00      0.00      0.00        44
     Channel Islands       0.00      0.00      0.00       230
              Cyprus       0.00      0.00      0.00       186
      Czech Republic       0.00      0.00      0.00         9
             Denmark       0.01      0.01      0.01       124
                EIRE       0.00      0.00      0.00      2322
  European Community       0.00      0.00      0.00        16
             Finland       0.00      0.00      0.00       213
              France       0.00      0.00      0.00      2525
             Germany       0.02      0.75      0.05      2785
              Greece       0.00      0.00      0.00        45
             Iceland       0.00      0.00      0.00        53
              Israel       0.00      0.00      0.00        75
               Italy       0.00      0.00      0.00       261
               Japan       0.00      0.00      0.00        96
             Lebanon       0.29      1.00      0.45        15
           Lithuania       0.00      1.00      0.01         9
               Malta       0.00      0.00      0.00        34
         Netherlands       0.11      0.00      0.01       719
              Norway       0.00      0.00      0.00       340
              Poland       0.00      0.00      0.00        92
            Portugal       0.00      0.00      0.00       469
                 RSA       0.12      1.00      0.21        23
        Saudi Arabia       0.00      0.00      0.00         2
           Singapore       0.03      0.02      0.02        64
               Spain       0.00      0.00      0.00       755
              Sweden       0.00      0.00      0.00       149
         Switzerland       0.02      0.00      0.01       536
                 USA       0.01      0.53      0.01        93
United Arab Emirates       0.00      0.00      0.00        20
      United Kingdom       0.80      0.07      0.12    108561

            accuracy                           0.08    122049
           macro avg       0.05      0.18      0.04    122049
        weighted avg       0.71      0.08      0.11    122049


Confusion Matrix:
[[   0    0   12 ...    0  178    0]
 [   0    0    1 ...    0    8    8]
 [   0    0    3 ...    0    0    2]
 ...
 [   0    0    0 ...    0    3    0]
 [   0    0 3046 ...   20 7063 3705]
 [   0    0   12 ...    0    0   29]]
PS C:\Users\manju>
```