In [14]:
```
pip install --upgrade seaborn matplotlib
```

In [29]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.decomposition import PCA
from sklearn.cluster import DBSCAN
```

In [33]:
```
# Load the dataset
file_path = r"C:\Users\nirav\Downloads\E-commerce Customer Behavior - Sheet1.csv"
data = pd.read_csv(file_path)
```

In [32]:
```
print("\n--- Checking Missing Values and Data Types ---\n")
print(data.info())
```

```
--- Checking Missing Values and Data Types ---

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 350 entries, 0 to 349
Data columns (total 11 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Customer ID               350 non-null    int64  
 1   Gender                    350 non-null    object 
 2   Age                       350 non-null    int64  
 3   City                      350 non-null    object 
 4   Membership Type           350 non-null    object 
 5   Total Spend               350 non-null    float64
 6   Items Purchased           350 non-null    int64  
 7   Average Rating            350 non-null    float64
 8   Discount Applied          350 non-null    bool   
 9   Days Since Last Purchase  350 non-null    int64  
 10  Satisfaction Level        348 non-null    object 
dtypes: bool(1), float64(2), int64(4), object(4)
memory usage: 27.8+ KB
None
```

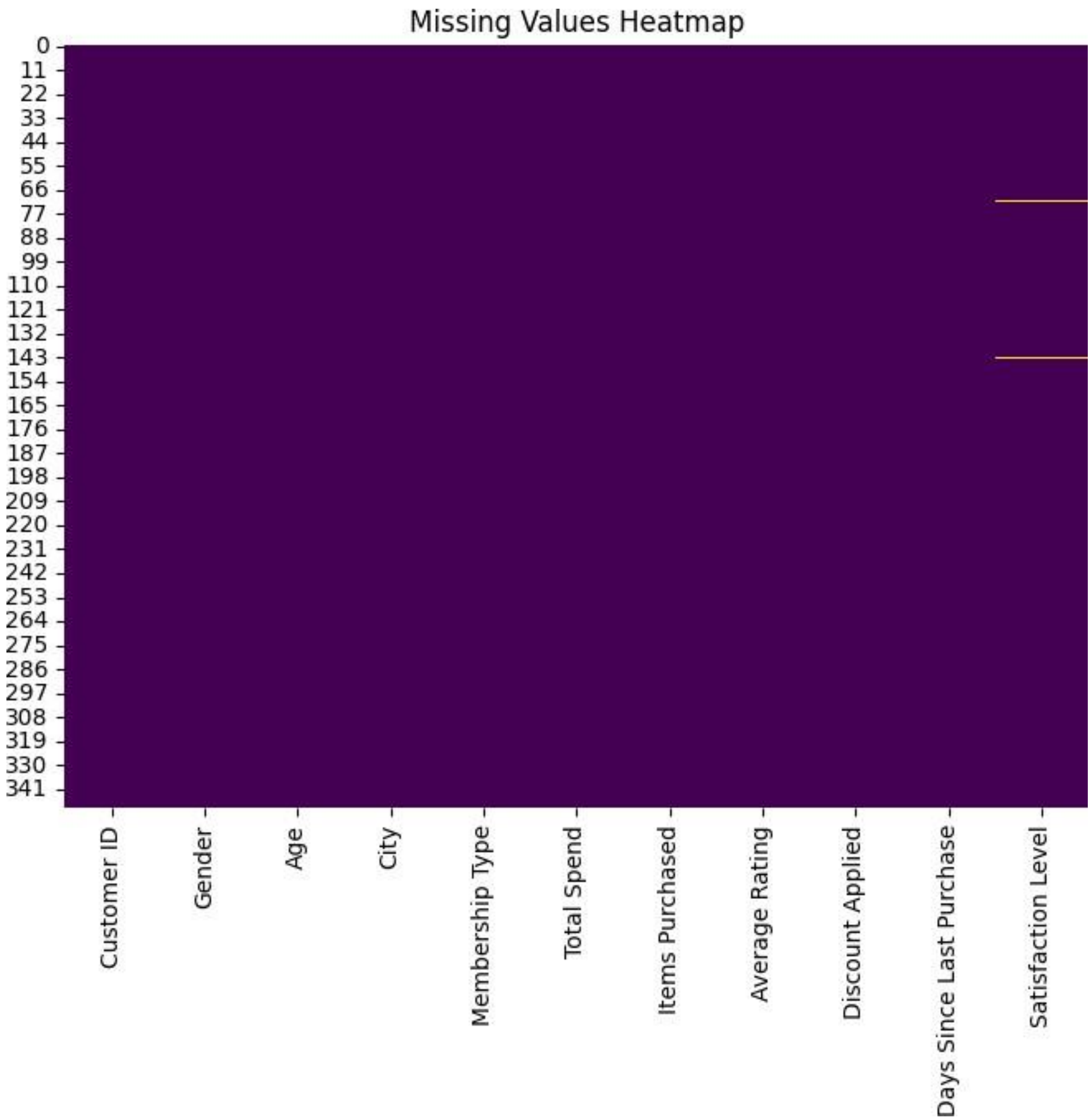
In [34]:

```
print("\n--- Summary Statistics ---\n")
print(data.describe())
```

```
--- Summary Statistics ---

       Customer ID         Age  Total Spend  Items Purchased  Average Rating  \
count   350.000000  350.000000   350.000000       350.000000      350.000000   
mean    275.500000   33.597143   845.381714        12.600000        4.019143   
std     101.180532    4.870882   362.058695         4.155984        0.580539   
min     101.000000   26.000000   410.800000         7.000000        3.000000   
25%     188.250000   30.000000   502.000000         9.000000        3.500000   
50%     275.500000   32.500000   775.200000        12.000000        4.100000   
75%     362.750000   37.000000  1160.600000        15.000000        4.500000   
max     450.000000   43.000000  1520.100000        21.000000        4.900000   

       Days Since Last Purchase  
count                350.000000  
mean                  26.588571  
std                   13.440813  
min                    9.000000  
25%                   15.000000  
50%                   23.000000  
75%                   38.000000  
max                   63.000000  
```

In [35]:

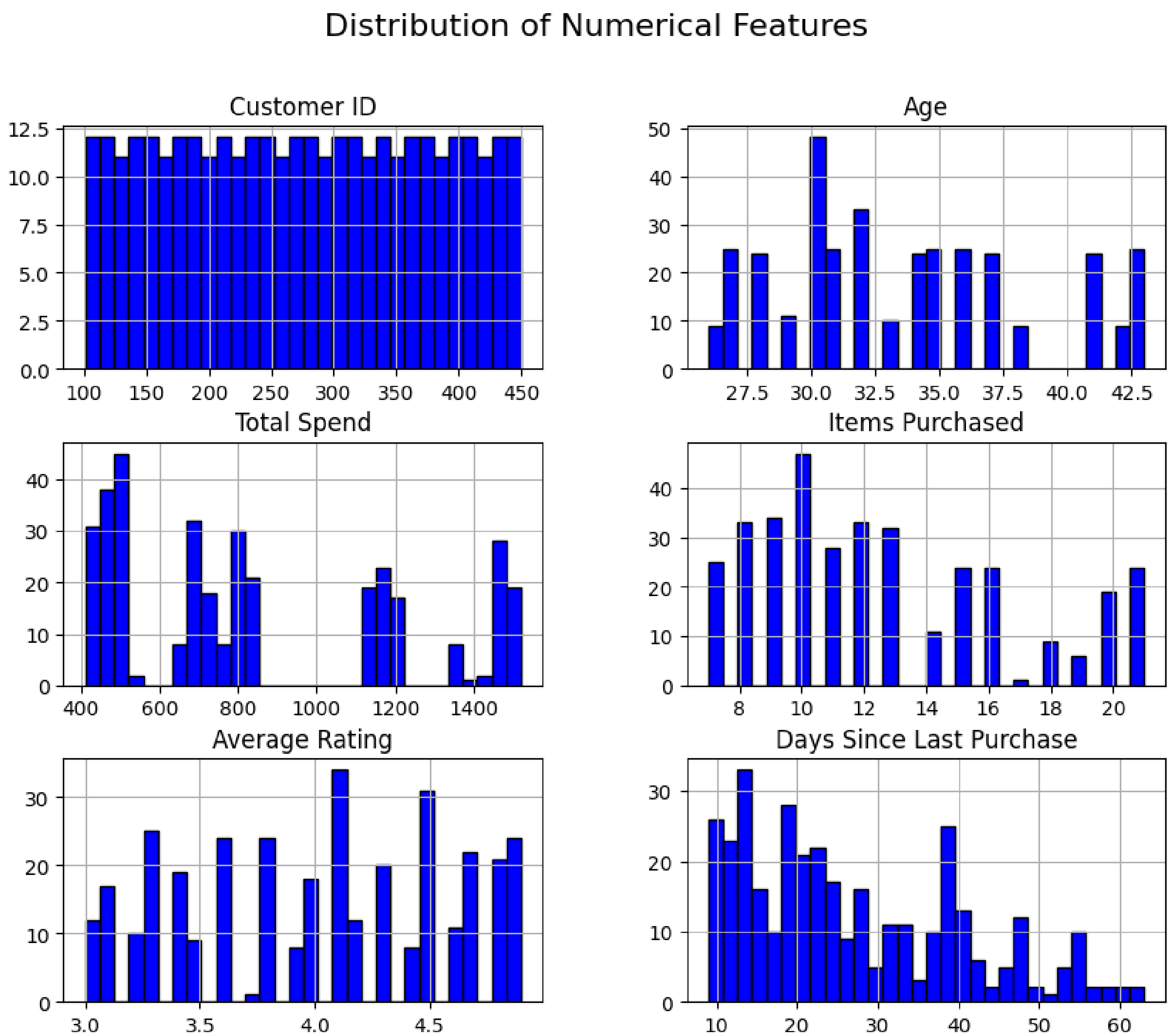
```
plt.figure(figsize=(8, 6))
sns.heatmap(data.isnull(), cbar=False, cmap='viridis')
plt.title("Missing Values Heatmap")
plt.show()
```



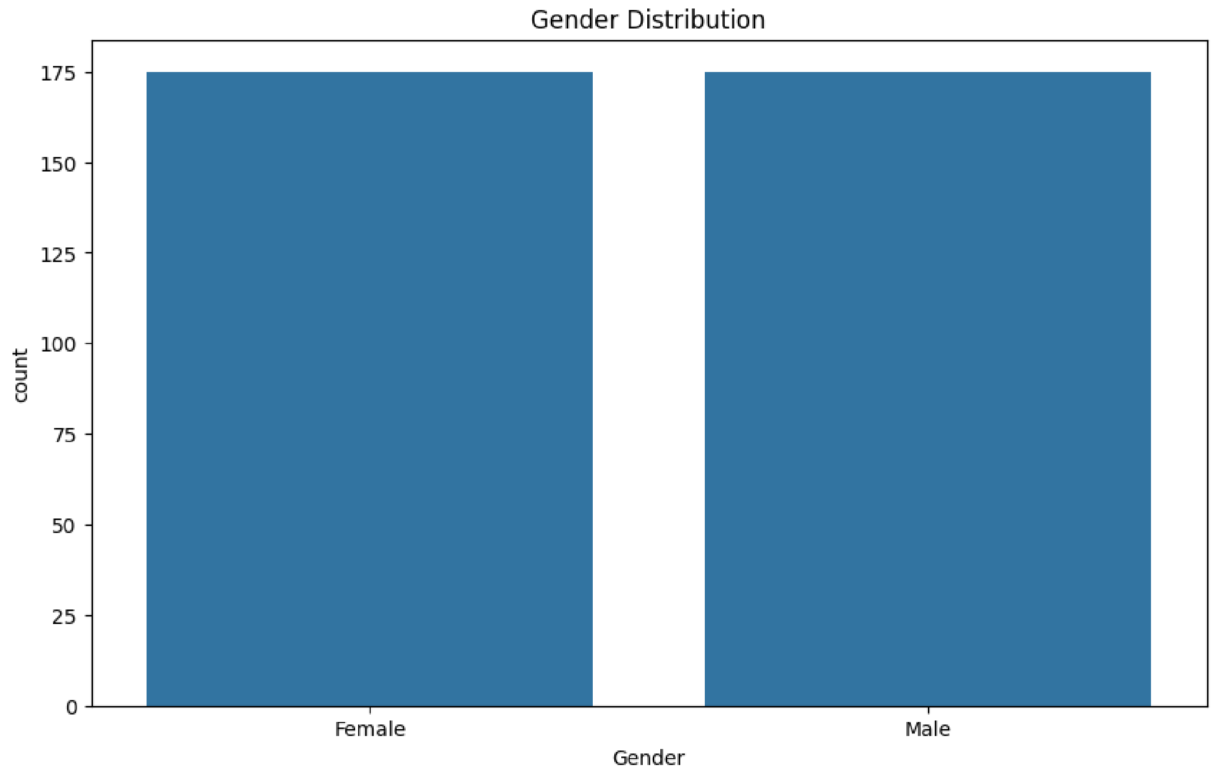
In [36]:

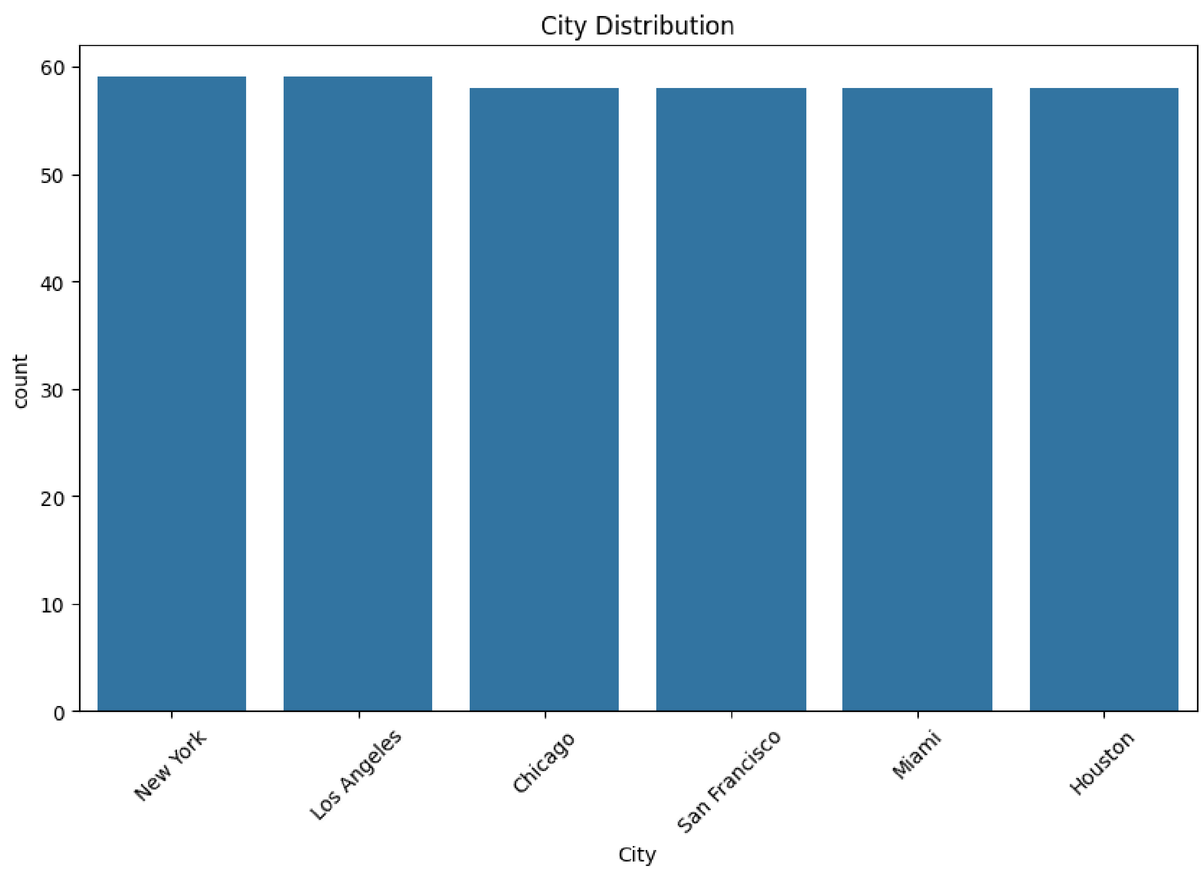
```
plt.figure(figsize=(10,  8))
data.hist(bins=30, figsize=(10, 8), color='blue', edgecolor='black')
plt.suptitle("Distribution of Numerical Features", fontsize=16)
plt.show()
```

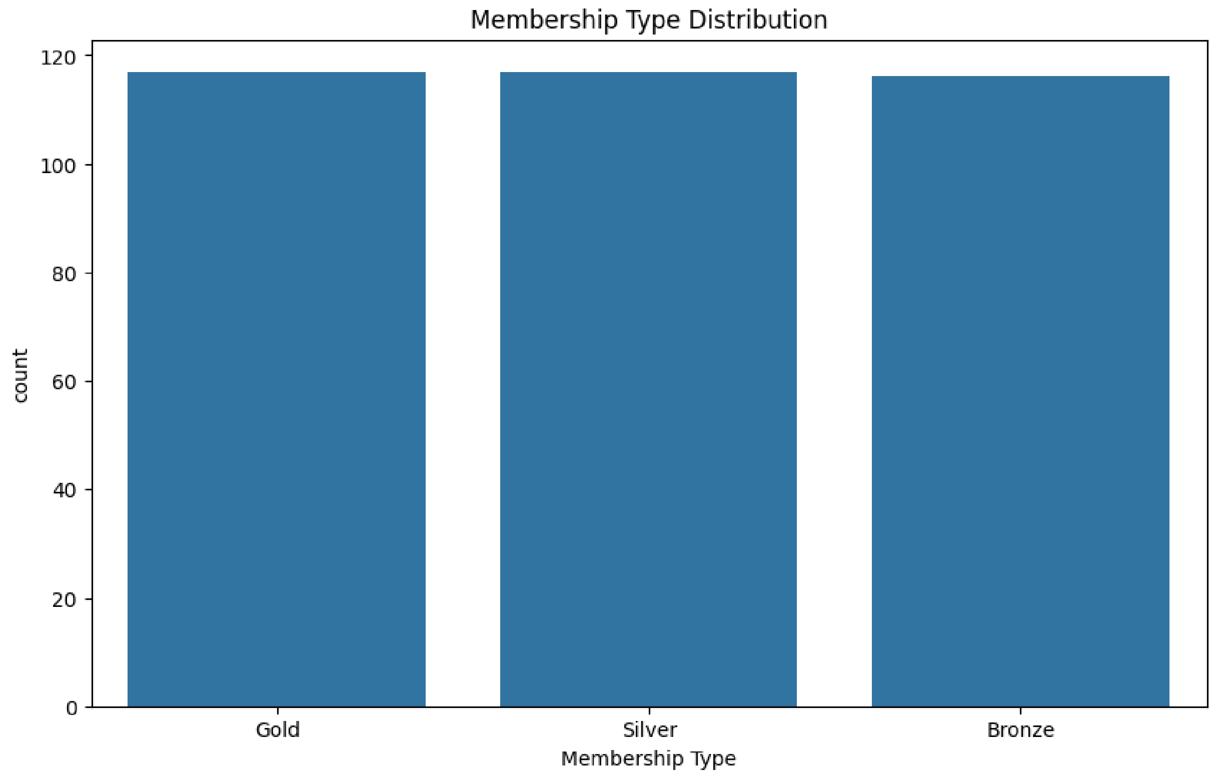
```
<Figure size 1000x800 with 0 Axes>
```

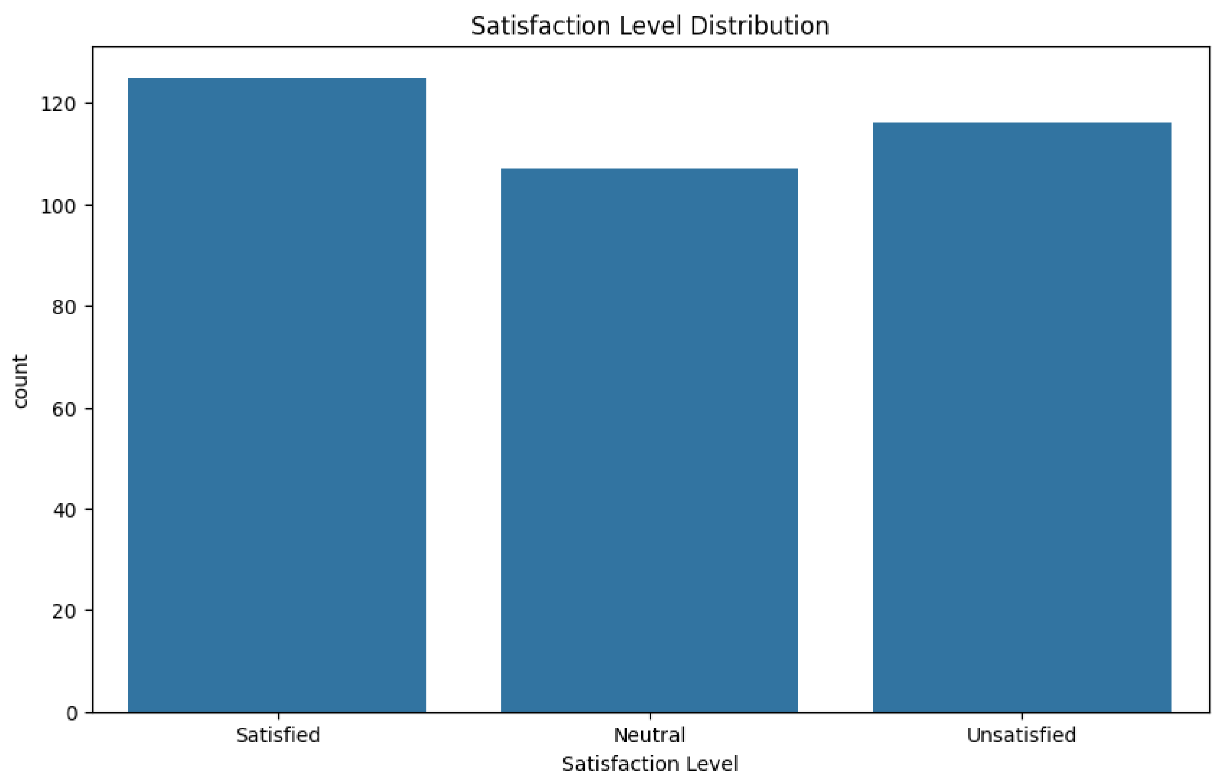


In [37]:
```
plt.figure(figsize=(10, 6))
sns.countplot(x='Gender', data=data)
plt.title("Gender Distribution")
plt.show()

plt.figure(figsize=(10, 6))
sns.countplot(x='City', data=data)
plt.title("City Distribution")
plt.xticks(rotation=45)
plt.show()

plt.figure(figsize=(10, 6))
sns.countplot(x='Membership Type', data=data)
plt.title("Membership Type Distribution")
plt.show()

plt.figure(figsize=(10, 6))
sns.countplot(x='Satisfaction Level', data=data)
plt.title("Satisfaction Level Distribution")
plt.show()
```


Gender Distribution
175
150
125
100
75
50
25
0
count
Female
Male
Gender


City Distribution
60
50
40
30
20
10
0
count
New York
Los Angeles
Chicago
San Francisco
Miami
Houston
City

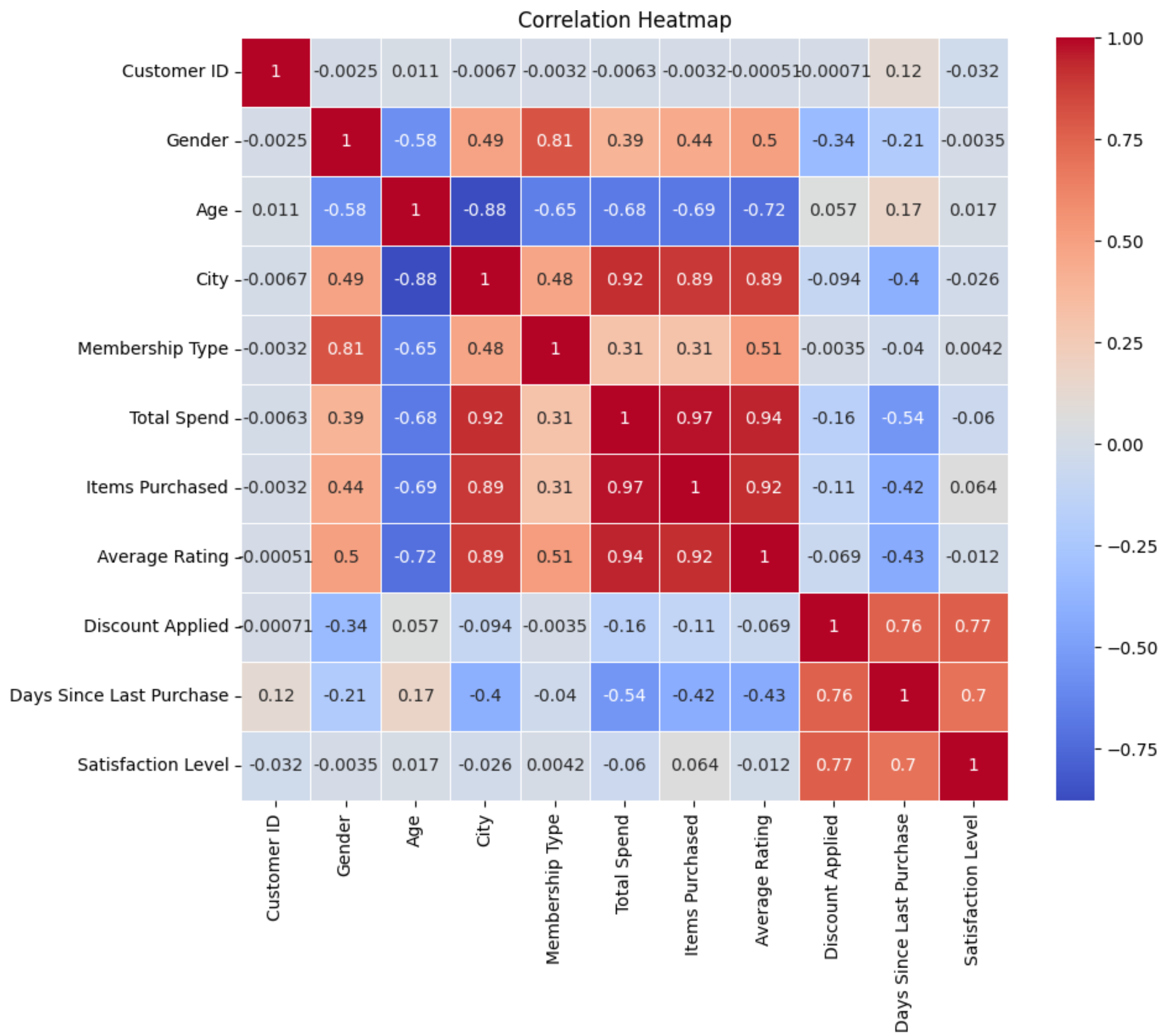




In [38]:

```
# Encode categorical variables for correlation matrix
label_encoder = LabelEncoder()
for column in ['Gender', 'City', 'Membership Type', 'Satisfaction Level']:
    data[column] = label_encoder.fit_transform(data[column])

plt.figure(figsize=(10, 8))
corr = data.corr()
sns.heatmap(corr, annot=True, cmap='coolwarm', linewidths=0.5)
```

```python
plt.title("Correlation Heatmap")
plt.show()
```



In [39]:

```python
# Encode categorical variables
label_encoder = LabelEncoder()
for column in ['Gender', 'City', 'Membership Type', 'Satisfaction Level']:
    data[column] = label_encoder.fit_transform(data[column])
```

In [40]:

```python
X = data.drop(columns=['Satisfaction Level'])  # Features
y = data['Satisfaction Level']  # Target
```

In [41]:

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta
```

In [42]:

```python
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [43]:

```python
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train_scaled, y_train)

# Predictions
y_pred_knn = knn.predict(X_test_scaled)
```

```python
# Evaluation metrics for KNN
print("K-Nearest Neighbors Performance:")
print("Accuracy:", accuracy_score(y_test, y_pred_knn))
print("Precision:", precision_score(y_test, y_pred_knn, average='weighted'))
print("Recall:", recall_score(y_test, y_pred_knn, average='weighted'))
print("F1 Score:", f1_score(y_test, y_pred_knn, average='weighted'))
print(classification_report(y_test, y_pred_knn))
```

```
K-Nearest Neighbors Performance:
Accuracy: 0.9904761904761905
Precision: 0.9907029478458049
Recall: 0.9904761904761905
F1 Score: 0.9904732855472521
              precision    recall  f1-score   support

           0       0.98      1.00      0.99        41
           1       1.00      0.97      0.99        40
           2       1.00      1.00      1.00        24

    accuracy                           0.99       105
   macro avg       0.99      0.99      0.99       105
weighted avg       0.99      0.99      0.99       105
```

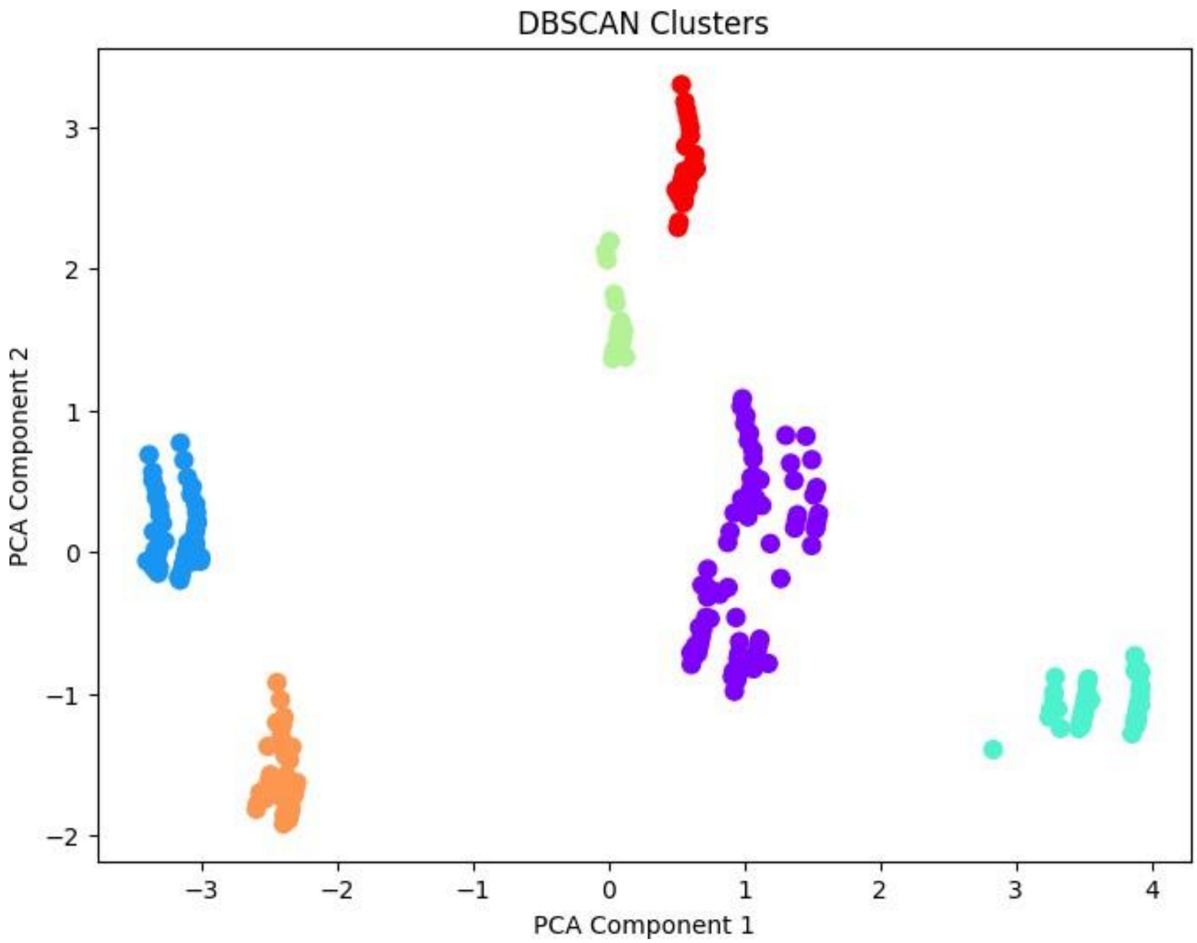
In [44]:
```python
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train_scaled, y_train)

# Predictions
y_pred_rf = rf.predict(X_test_scaled)

# Evaluation metrics for Random Forest
print("Random Forest Performance:")
print("Accuracy:", accuracy_score(y_test, y_pred_rf))
print("Precision:", precision_score(y_test, y_pred_rf, average='weighted'))
print("Recall:", recall_score(y_test, y_pred_rf, average='weighted'))
print("F1 Score:", f1_score(y_test, y_pred_rf, average='weighted'))
print(classification_report(y_test, y_pred_rf))
```

```
Random Forest Performance:
Accuracy: 0.9714285714285714
Precision: 1.0
Recall: 0.9714285714285714
F1 Score: 0.9851717902350814
              precision    recall  f1-score   support

           0       1.00      0.93      0.96        41
           1       1.00      1.00      1.00        40
           2       1.00      1.00      1.00        24
           3       0.00      0.00      0.00         0

    accuracy                           0.97       105
   macro avg       0.75      0.73      0.74       105
weighted avg       1.00      0.97      0.99       105
```

In [45]:
```python
nb = GaussianNB()
nb.fit(X_train_scaled,  y_train)

# Predictions
y_pred_nb = nb.predict(X_test_scaled)

# Evaluation metrics for Naive Bayes
print("Naive Bayes Performance:")
print("Accuracy:", accuracy_score(y_test,  y_pred_nb))
print("Precision:", precision_score(y_test, y_pred_nb, average='weighted'))
print("Recall:", recall_score(y_test, y_pred_nb, average='weighted'))
print("F1 Score:", f1_score(y_test, y_pred_nb, average='weighted'))
print(classification_report(y_test, y_pred_nb))
```

```
Naive Bayes Performance:
Accuracy: 0.9238095238095239
Precision: 0.9888435374149659
Recall: 0.9238095238095239
F1 Score: 0.9540750610703975
              precision    recall  f1-score   support

           0       0.97      0.83      0.89        41
           1       1.00      0.97      0.99        40
           2       1.00      1.00      1.00        24
           3       0.00      0.00      0.00         0

    accuracy                           0.92       105
   macro avg       0.74      0.70      0.72       105
weighted avg       0.99      0.92      0.95       105
```

In [46]:
```python
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_train_scaled)

# DBSCAN model
dbscan = DBSCAN(eps=0.5, min_samples=5)
dbscan_labels = dbscan.fit_predict(X_pca)

# Visualize the DBSCAN clusters
plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=dbscan_labels, cmap='rainbow', s=50)
plt.title('DBSCAN Clusters')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.show()
```



DBSCAN Clusters
PCA Component 2
PCA Component 1
3
2
1
0
−1
−2
−3
−2
−1
0
1
2
3
4