teps to Perform Customer Segmentation

Step 1: Understand the Data

For segmentation, you will use data from Customers.csv and Transactions.csv. Key features include:

Demographic data: e.g., Region, SignupDate.

Behavioral data: e.g., TotalValue, Purchase frequency.

Step 2: Preprocess the Data

Prepare the data for clustering.

1. Merge Datasets: Combine Customers.csv and Transactions.csv to create a unified dataset.

```
merged_data = transactions.merge(customers, on='CustomerID')
```

2. Aggregate Customer Features: Compute aggregated metrics for each customer:

Total Spend: Sum of TotalValue.

Average Order Value: Mean of TotalValue.

Purchase Frequency: Count of transactions.

Recency: Days since the last transaction.

```
# Aggregated features
customer_features = merged_data.groupby('CustomerID').agg({
    'TotalValue': ['sum', 'mean'],
    'TransactionID': 'count',
    'TransactionDate': lambda x: (pd.Timestamp.now() - pd.to_datetime(x).max()).days
}).reset_index()

# Rename columns
customer_features.columns = ['CustomerID', 'TotalSpend', 'AvgOrderValue', 'PurchaseFrequency', 'Recency']
```

3. Handle Missing Values: Fill or remove missing data.

```
customer_features.fillna(0, inplace=True)
```

4. Normalize the Data: Scale numerical features to ensure all features contribute equally to clustering.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaled_features = scaler.fit_transform(customer_features.drop('CustomerID', axis=1))
```

Step 3: Choose a Clustering Algorithm

Several clustering algorithms can be used for segmentation:

1. K-Means Clustering: Groups customers into k clusters by minimizing intra-cluster variance.

```
from sklearn.cluster import KMeans

# Determine the optimal number of clusters using the Elbow Method
inertia = []
for k in range(2, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(scaled_features)
    inertia.append(kmeans.inertia_)

# Plot the Elbow Curve
import matplotlib.pyplot as plt
plt.plot(range(2, 11), inertia, marker='o')
plt.title('Elbow Method')
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
plt.show()

# Fit K-Means with optimal clusters
kmeans = KMeans(n_clusters=4, random_state=42)
customer_features['Cluster'] = kmeans.fit_predict(scaled_features)
```

2. Hierarchical Clustering: Builds a hierarchy of clusters.

```
from scipy.cluster.hierarchy import dendrogram, linkage

# Perform hierarchical clustering
linked = linkage(scaled_features, method='ward')

# Plot dendrogram
plt.figure(figsize=(10, 7))
dendrogram(linked)
plt.title('Dendrogram')
plt.show()
```

3. DBSCAN (Density-Based Clustering): Groups customers based on density, identifying outliers as noise.

```
from sklearn.cluster import DBSCAN

dbscan = DBSCAN(eps=0.5, min_samples=5)
customer_features['Cluster'] = dbscan.fit_predict(scaled_features)
```

Step 4: Evaluate the Clustering

1. Davies-Bouldin Index: Measures the compactness and separation of clusters (lower is better).

```
from sklearn.metrics import davies_bouldin_score
```

```
db_index = davies_bouldin_score(scaled_features, customer_features['Cluster'])
print("Davies-Bouldin Index:", db_index)
```

2. Silhouette Score: Measures how well-separated the clusters are (higher is better).

```
from sklearn.metrics import silhouette_score

silhouette = silhouette_score(scaled_features, customer_features['Cluster'])
print("Silhouette Score:", silhouette)
```

Step 5: Visualize the Clusters

1. Scatter Plot: Visualize clusters using the first two principal components.

```
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
reduced_features = pca.fit_transform(scaled_features)

plt.scatter(reduced_features[:, 0], reduced_features[:, 1], c=customer_features['Cluster'],
cmap='viridis')
plt.title('Customer Segmentation')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.colorbar(label='Cluster')
plt.show()
```

2. Cluster Distribution: Check the number of customers in each cluster.

```
print(customer_features['Cluster'].value_counts())
```

Step 6: Interpret the Clusters

Analyze the characteristics of each cluster to derive actionable insights.

1. Group customers by cluster and compute summary statistics:

```
cluster_summary = customer_features.groupby('Cluster').agg({
    'TotalSpend': 'mean',
    'AvgOrderValue': 'mean',
    'PurchaseFrequency': 'mean',
    'Recency': 'mean'
})
print(cluster_summary)
```

2. Example Insights:

Cluster 0: High spenders with frequent purchases.

Cluster 1: Low spenders with infrequent purchases.

Cluster 2: Recently active customers with moderate spending.

Cluster 3: Dormant customers with low activity.

Step 7: Save the Results

Save the segmented dataset for further use.

```
customer_features.to_csv('customer_segments.csv', index=False)
```

Applications of Customer Segmentation

1. Marketing:

Personalize marketing campaigns for each segment.

Retarget dormant customers.

2. Product Recommendations:

Suggest products based on segment preferences.

3. Customer Retention:

Identify at-risk customers and design loyalty programs.

4. Revenue Growth:

Focus on high-value customers to maximize ROI.