Task 3: Customer Segmentation / Clustering

Code Explanation

```
# Import necessary libraries

import pandas as pd  # For data manipulation and analysis

from sklearn.preprocessing import StandardScaler  # For feature scaling

from sklearn.cluster import KMeans  # For K-Means clustering

from sklearn.metrics import davies_bouldin_score  # For evaluating clustering performance

import matplotlib.pyplot as plt  # For plotting

import seaborn as sns  # For enhanced data visualization
```

1. Library Imports:

- Pandas: Used for data manipulation and analysis, particularly for handling CSV files and DataFrames.
- Scikit-learn: Provides tools for machine learning, including clustering algorithms (K-Means) and metrics for evaluating clustering performance (Davies-Bouldin Index).
- Matplotlib and Seaborn: Used for data visualization to help understand the clustering results visually.

```
# Load the data from CSV files

customers = pd.read_csv('Customers.csv')  # Load customer data

transactions = pd.read_csv('Transactions.csv')  # Load transaction data
```

2. Data Loading:

- The customer and transaction data are loaded from CSV files into pandas DataFrames. This allows for easy manipulation and analysis of the data.

```
# Data Preprocessing

# Merge transactions with customers to get a complete transaction history

merged_data = transactions.merge(customers, on='CustomerID', how='left')  # Merge on CustomerID
```

3. Data Preprocessing:

- The transaction data is merged with customer data to create a comprehensive dataset that includes transaction history for each customer. This is crucial for understanding customer behavior and transaction patterns.

```
# Feature Engineering

# Create a feature for total transaction value per customer

customer_transaction_value = merged_data.groupby('CustomerID')['TotalValue'].sum().reset_index()

customer_transaction_value.columns = ['CustomerID', 'TotalTransactionValue']  # Rename columns for clarity
```

4. Feature Engineering:

- Total Transaction Value : This feature is created by grouping the merged data by `CustomerID` and summing the `TotalValue` of transactions. This helps in understanding how much each customer spends in total, which is a key indicator of customer value.

```
# Create a feature for the number of transactions per customer

customer_transaction_count = merged_data.groupby('CustomerID')['TransactionID'].count().reset_index()

customer_transaction_count.columns = ['CustomerID', 'TransactionCount']  # Rename columns for clarity
```

5. Transaction Count :

- This feature counts the number of transactions for each customer. It provides insight into customer engagement and frequency of purchases, which are important for segmentation.

```
# Merge the calculated features back into the customer data

customer_features = customers.merge(customer_transaction_value, on='CustomerID', how='left')

customer_features = customer_features.merge(customer_transaction_count, on='CustomerID', how='left')


# Fill any NaN values with 0 (for customers with no transactions)

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

6. Merging Features :

- The calculated features (total transaction value and transaction count) are merged back into the customer DataFrame. Any missing values (NaN) are filled with 0 to ensure that all customers have complete data for clustering.

```
# Select relevant features for clustering

features = customer_features[['TotalTransactionValue', 'TransactionCount']]  # Select features for clustering
```

7. Feature Selection :

- The relevant features for clustering are selected. In this case, we focus on `TotalTransactionValue` and `TransactionCount`, as they are key indicators of customer behavior.

```
# Standardize the features to have a mean of 0 and a standard deviation of 1

scaler = StandardScaler()  # Initialize the scaler

scaled_features = scaler.fit_transform(features)  # Fit and transform the features
```

8. Standardization :

- Standardization is performed to ensure that all features contribute equally to the clustering algorithm. K-Means is sensitive to the scale of the data, so scaling is essential for effective clustering.

```
# Clustering

# Choose the number of clusters (between 2 and 10)

n_clusters = 5  # Set the number of clusters for K-Means

kmeans = KMeans(n_clusters=n_clusters, random_state=42)  # Initialize K-Means with the specified number of clusters

customer_features['Cluster'] = kmeans.fit_predict(scaled_features)  # Fit the model and predict cluster labels
```

9. Clustering :

- The K-Means algorithm is chosen for clustering due to its simplicity and effectiveness in partitioning data into distinct groups. The number of clusters is set to 5, but this can be adjusted based on the analysis. The model is fitted to the scaled features, and cluster labels are assigned to each customer.

```
# Evaluation: Calculate the Davies-Bouldin Index

db_index = davies_bouldin_score(scaled_features, customer_features['Cluster'])  # Calculate DB Index

print(f'Davies-Bouldin Index for {n_clusters} clusters: {db_index}')  # Print the DB Index
```

10. Evaluation :

- The Davies-Bouldin Index (DB Index) is calculated to evaluate the clustering quality. A lower DB Index indicates better clustering, as it measures the average similarity ratio of each cluster with its most similar cluster. This metric helps assess how well the clusters are separated.

```
# Visualization

# Set the style for seaborn

sns.set(style="whitegrid")  # Set the style for the plots


# Create a scatter plot of the clusters

plt.figure(figsize=(10, 6))  # Set the figure size

sns.scatterplot(x='TotalTransactionValue', y='TransactionCount', hue='Cluster', data=customer_features, palette='viridis', s=100)

plt.title(f'Customer Segmentation using K-Means Clustering (n_clusters={n_clusters})')  # Set the title

plt.xlabel('Total Transaction Value')  # Set the x-axis label

plt.ylabel('Transaction Count')  # Set the y-axis label

plt.legend(title='Cluster')  # Add a legend

plt.show()  # Display the plot
```

11. Visualization :

- A scatter plot is created to visualize the clusters based on total transaction value and transaction count. Different colors represent different clusters, allowing for an intuitive understanding of how customers are grouped.

```
# Optional: Visualize cluster centers

centers = scaler.inverse_transform(kmeans.cluster_centers_)  # Inverse transform the cluster centers to original scale

plt.figure(figsize=(8, 5))  # Set the figure size
plt.scatter(features['TotalTransactionValue'], features['TransactionCount'], c=customer_features['Cluster'], cmap='viridis', s=100)  # Scatter plot of customers

plt.scatter(centers[:, 0], centers[:, 1], c='red', marker='X', s=200, label='Centroids')  # Plot the centroids

plt.title('Customer Segmentation with Cluster Centers')  # Set the title

plt.xlabel('Total Transaction Value')  # Set the x-axis label

plt.ylabel('Transaction Count')  # Set the y-axis label

plt.legend()  # Add a legend

plt.show()  # Display the plot
```

12. Cluster Centers Visualization :

- The cluster centers (centroids) are visualized on the scatter plot to show the center of each cluster. This helps in understanding the average characteristics of each customer segment.

Why I Chose This Approach

- Clustering Algorithm : K-Means was chosen for its simplicity and effectiveness in partitioning data into distinct groups. It is widely used for customer segmentation due to its efficiency and ease of interpretation.

- Feature Selection : Total transaction value and transaction count were selected as they are key indicators of customer behavior and engagement. These features provide valuable insights into customer spending patterns.

- Standardization : Standardizing the features ensures that all features contribute equally to the clustering process, preventing any single feature from dominating the results due to its scale.

- Evaluation Metric : The Davies-Bouldin Index was chosen as it provides a quantitative measure of clustering quality, helping to assess how well the clusters are separated.

- Visualization : Visualizing the clusters allows for an intuitive understanding of customer segments, making it easier to interpret the results and derive actionable insights.

This approach provides me a comprehensive framework for customer segmentation, allowing businesses to identify distinct customer groups and tailor their marketing strategies accordingly.