Lookalike Model Code

```
# Customer Segmentation / Clustering


import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import davies_bouldin_score
import matplotlib.pyplot as plt
import seaborn as sns


# Step 1: Load and Merge Data
def load_and_prepare_data(customers_path, transactions_path):
    customers_df = pd.read_csv(customers_path)
    transactions_df = pd.read_csv(transactions_path)


    # Aggregate transaction data
    transactions_agg = transactions_df.groupby("CustomerID").agg(
        total_spent=("TotalValue", "sum"),
        transaction_count=("TransactionID", "count"),
        avg_transaction_value=("TotalValue", "mean")
    ).reset_index()


    # Merge with customer profile data
    merged_df = pd.merge(customers_df, transactions_agg, on="CustomerID", how="inner")
    return merged_df


# Step 2: Preprocess Data
def preprocess_data(data):
    features = ["total_spent", "transaction_count", "avg_transaction_value"]
    scaler = StandardScaler()
```

```
    data_scaled = scaler.fit_transform(data[features])
    return data_scaled, features

# Step 3: Perform Clustering
def perform_clustering(data_scaled, n_clusters):
    kmeans = KMeans(n_clusters=n_clusters, random_state=42)
    cluster_labels = kmeans.fit_predict(data_scaled)
    return kmeans, cluster_labels

# Step 4: Evaluate Clustering
def evaluate_clustering(data_scaled, cluster_labels):
    db_index = davies_bouldin_score(data_scaled, cluster_labels)
    return db_index

# Step 5: Visualize Clusters
def visualize_clusters(data, cluster_labels, features):
    data["Cluster"] = cluster_labels
    sns.pairplot(data, hue="Cluster", vars=features, palette="viridis")
    plt.show()

# Main Function
def main():
    # File paths
    customers_path = "Customers.csv"
    transactions_path = "Transactions.csv"

    # Load and prepare data
    data = load_and_prepare_data(customers_path, transactions_path)

    # Preprocess data
    data_scaled, features = preprocess_data(data)
```

```
    # Clustering and evaluation
    best_db_index = float("inf")
    best_n_clusters = None
    best_labels = None

    for n_clusters in range(2, 11):
        kmeans, cluster_labels = perform_clustering(data_scaled, n_clusters)
        db_index = evaluate_clustering(data_scaled, cluster_labels)
        print(f"Number of Clusters: {n_clusters}, DB Index: {db_index:.4f}")

        if db_index < best_db_index:
            best_db_index = db_index
            best_n_clusters = n_clusters
            best_labels = cluster_labels

    # Print best clustering results
    print(f"\nOptimal Number of Clusters: {best_n_clusters}, Best DB Index: {best_db_index:.4f}")

    # Visualize clusters
    visualize_clusters(data, best_labels, features)

if __name__ == "__main__":
    main()
```