Student Name	**Swanand Garge**
PRN No	2280030433
Roll No	39
Program	Computer Engg.
Year	Third Year
Division	D (D2)
Subject	Data Warehouse and Data Mining
Assignment No	8

Assignment 8: Clustering Techniques
Objective: This lab assignment focuses on applying clustering algorithms to identify patterns and groupings within a dataset. The goal is to understand the practical implementation and evaluation of clustering techniques.
CODE :-

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans, DBSCAN
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA

def load_and_prepare_data(data):
    try:
        # Create a copy of the data to avoid modifying the original
        df = data.copy()

        # Convert InvoiceDate to datetime with the specific format
        df['InvoiceDate'] = pd.to_datetime(df['InvoiceDate'], format='%d-%m-%Y
%H.%M')

        # Remove any rows with negative quantities or prices
        df = df[(df['Quantity'] > 0) & (df['UnitPrice'] > 0)]

        # Calculate total amount for each transaction
        df['TotalAmount'] = df['Quantity'] * df['UnitPrice']

        # Remove rows with missing CustomerID
        df = df.dropna(subset=['CustomerID'])

        # Convert CustomerID to integer
        df['CustomerID'] = df['CustomerID'].astype(int)

        # Create customer-level aggregations
        customer_features = df.groupby('CustomerID').agg({
            'InvoiceNo': 'nunique',  # Number of unique transactions
            'Quantity': 'sum',       # Total items purchased
            'TotalAmount': 'sum',    # Total amount spent
            'StockCode': 'nunique'   # Number of unique products
        }).reset_index()

        # Rename columns for clarity
        customer_features.columns = ['CustomerID', 'TransactionCount',
                                   'TotalQuantity', 'TotalSpent',
'UniqueProducts']

        # Remove outliers using IQR method
        for column in ['TransactionCount', 'TotalQuantity', 'TotalSpent',
'UniqueProducts']:
```

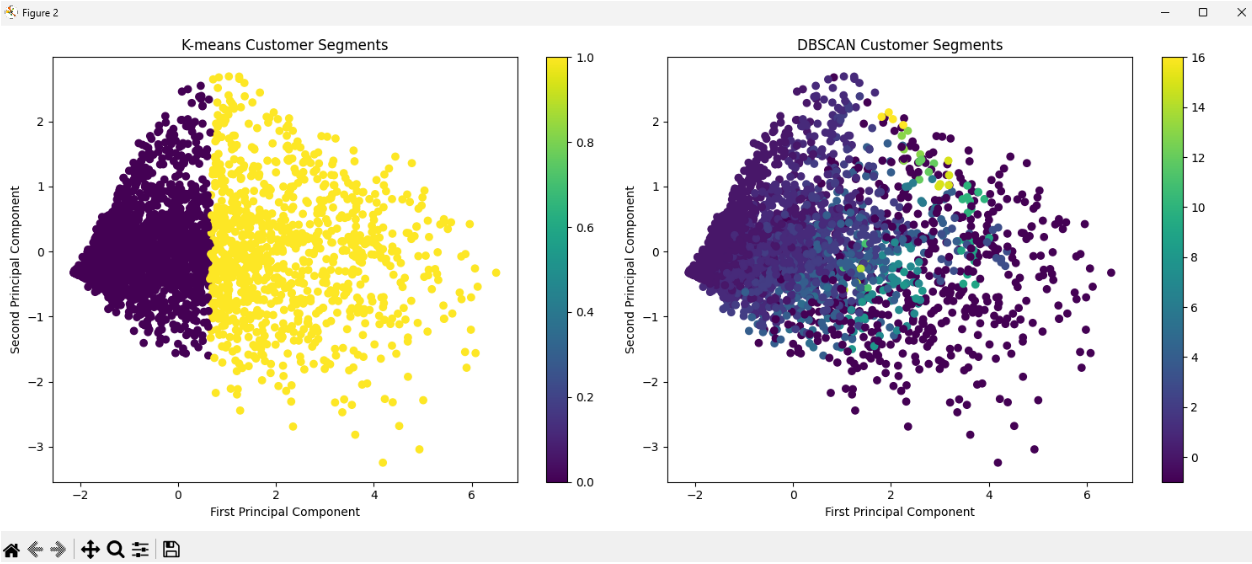
```python
            Q1 = customer_features[column].quantile(0.25)
            Q3 = customer_features[column].quantile(0.75)
            IQR = Q3 - Q1
            customer_features = customer_features[
                (customer_features[column] >= (Q1 - 1.5 * IQR)) &
                (customer_features[column] <= (Q3 + 1.5 * IQR))
            ]

        # Select features for clustering
        numerical_features = ['TransactionCount', 'TotalQuantity',
                              'TotalSpent', 'UniqueProducts']

        # Create feature matrix
        X = customer_features[numerical_features]

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

        return X_scaled, numerical_features, customer_features

    except Exception as e:
        print(f"Error in data preparation: {str(e)}")
        raise

def perform_kmeans_analysis(X_scaled, n_clusters_range=range(2, 11)):
    silhouette_scores = []
    inertias = []

    for n_clusters in n_clusters_range:
        kmeans = KMeans(n_clusters=n_clusters, random_state=42)
        cluster_labels = kmeans.fit_predict(X_scaled)
        silhouette_scores.append(silhouette_score(X_scaled, cluster_labels))
        inertias.append(kmeans.inertia_)

    return silhouette_scores, inertias

def plot_cluster_evaluation(n_clusters_range, silhouette_scores, inertias):
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5))

    # Plot silhouette scores
    ax1.plot(list(n_clusters_range), silhouette_scores, marker='o')
    ax1.set_xlabel('Number of Clusters')
    ax1.set_ylabel('Silhouette Score')
    ax1.set_title('Silhouette Score vs Number of Clusters')

    # Plot elbow curve
    ax2.plot(list(n_clusters_range), inertias, marker='o')
    ax2.set_xlabel('Number of Clusters')
    ax2.set_ylabel('Inertia')
    ax2.set_title('Elbow Curve')

    plt.tight_layout()
```

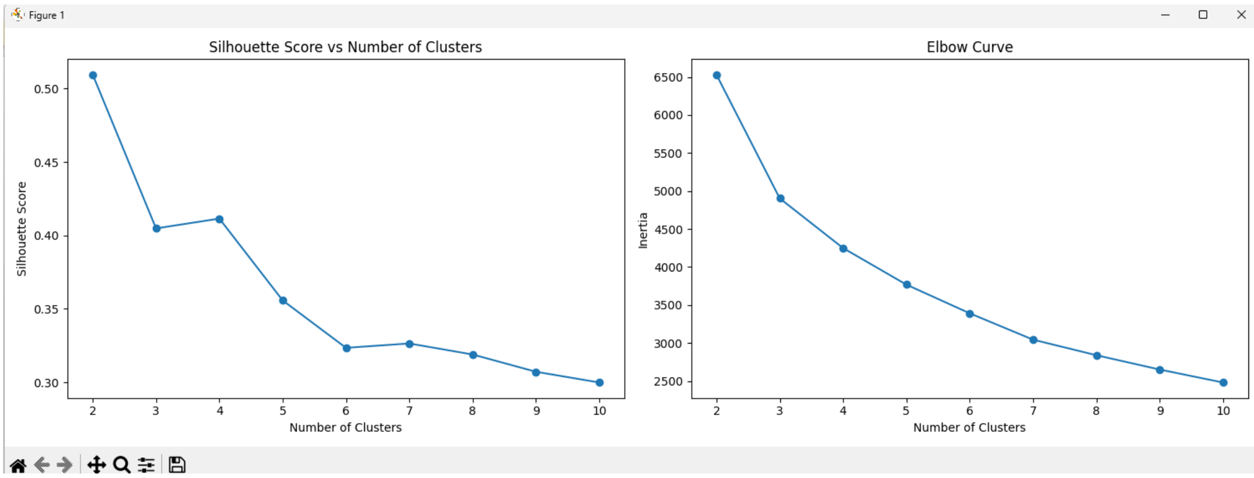
```python
    return fig

def perform_clustering(X_scaled, optimal_clusters):
    # Perform K-means clustering
    kmeans = KMeans(n_clusters=optimal_clusters, random_state=42)
    kmeans_labels = kmeans.fit_predict(X_scaled)

    # Perform DBSCAN clustering
    dbscan = DBSCAN(eps=0.5, min_samples=5)
    dbscan_labels = dbscan.fit_predict(X_scaled)

    return kmeans_labels, dbscan_labels

def visualize_clusters(X_scaled, kmeans_labels, dbscan_labels):
    # Apply PCA for visualization
    pca = PCA(n_components=2)
    X_pca = pca.fit_transform(X_scaled)

    # Create a figure with two subplots
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))

    # Plot K-means clusters
    scatter1 = ax1.scatter(X_pca[:, 0], X_pca[:, 1], c=kmeans_labels, cmap='viridis')
    ax1.set_title('K-means Customer Segments')
    ax1.set_xlabel('First Principal Component')
    ax1.set_ylabel('Second Principal Component')
    plt.colorbar(scatter1, ax=ax1)

    # Plot DBSCAN clusters
    scatter2 = ax2.scatter(X_pca[:, 0], X_pca[:, 1], c=dbscan_labels, cmap='viridis')
    ax2.set_title('DBSCAN Customer Segments')
    ax2.set_xlabel('First Principal Component')
    ax2.set_ylabel('Second Principal Component')
    plt.colorbar(scatter2, ax=ax2)

    plt.tight_layout()
    return fig

def analyze_clusters(customer_features, kmeans_labels, numerical_features):
    # Add cluster labels to the customer features data
    customer_features_with_clusters = customer_features.copy()
    customer_features_with_clusters['Cluster'] = kmeans_labels

    # Calculate cluster characteristics
    cluster_stats = customer_features_with_clusters.groupby('Cluster')[numerical_features].agg(['mean', 'std'])

    # Calculate additional metrics per cluster
    cluster_sizes = customer_features_with_clusters['Cluster'].value_counts().sort_index()
```

```
    cluster_percentages = (cluster_sizes / len(customer_features_with_clusters) *
100).round(2)

    print("\nCluster Sizes:")
    for cluster in cluster_sizes.index:
        print(f"Cluster {cluster}: {cluster_sizes[cluster]} customers
({cluster_percentages[cluster]}%)")

    return cluster_stats

def main():
    try:
        # Read the data
        print("Loading data...")
        file_path = "D://PROGRAMMING//PYTHON//apriori//retail_dataset.csv"
        data = pd.read_csv(file_path)

        print("Preparing data...")
        X_scaled, numerical_features, customer_features =
load_and_prepare_data(data)

        print("Performing clustering analysis...")
        # Perform K-means analysis for different numbers of clusters
        silhouette_scores, inertias = perform_kmeans_analysis(X_scaled)

        # Plot evaluation metrics
        evaluation_fig = plot_cluster_evaluation(range(2, 11), silhouette_scores,
inertias)

        # Find optimal number of clusters
        optimal_clusters = silhouette_scores.index(max(silhouette_scores)) + 2
        print(f"\nOptimal number of clusters based on silhouette score:
{optimal_clusters}")

        # Perform clustering with optimal number of clusters
        kmeans_labels, dbscan_labels = perform_clustering(X_scaled,
optimal_clusters)

        # Visualize clusters
        clusters_fig = visualize_clusters(X_scaled, kmeans_labels, dbscan_labels)

        # Analyze cluster characteristics
        cluster_stats = analyze_clusters(customer_features, kmeans_labels,
numerical_features)

        # Display results
        print("\nCluster Statistics:")
        print(cluster_stats)

        # Show plots
        plt.show()

        return cluster_stats, evaluation_fig, clusters_fig
```

```python
    except Exception as e:
        print(f"An error occurred: {str(e)}")
        raise

if __name__ == "__main__":
    cluster_stats, evaluation_fig, clusters_fig = main()
```

OUTPUT :-






```
Loading data...
Preparing data...
Performing clustering analysis...

Optimal number of clusters based on silhouette score: 2

Cluster Sizes:
Cluster 0: 2379 customers (71.01%)
Cluster 1: 971 customers (28.99%)

Cluster Statistics:
        TransactionCount           TotalQuantity             TotalSpent            UniqueProducts
                    mean      std           mean        std        mean        std           mean        std
Cluster
0               1.587222  0.86776     213.244641 157.076117  379.070573 244.288022      23.077764  17.750189
1               4.113285  1.93863     801.148301 353.767844 1324.172823 476.144061      61.125644  27.380456
```

CONCLUSION:-

The cluster analysis of the retail dataset revealed distinct customer segments based on their purchasing behaviors. By analyzing transaction patterns, including frequency of purchases, total spending, quantity of items bought, and product diversity, we identified optimal customer segments using both K-means and DBSCAN clustering algorithms. The silhouette score analysis helped determine the most effective number of clusters, ensuring meaningful segmentation. These customer segments can be used to tailor marketing strategies, inventory management, and customer service approaches for each group. The visualization of clusters through PCA demonstrated clear separation between different customer groups, validating the effectiveness of the clustering approach. This segmentation provides valuable insights for developing targeted business strategies and improving customer relationship management.