PHASE 4
CUSTOMER SEGMENTATION USING DATA SCIENCE

Thoufeeq A - 71772117146
Sudharsan M - 71772117144
Sandhiya S - 71772117138
Srikanth B - 71772117143

Description:

This document focus on the preparation and analyzing the data through feature engineering, t-SNE dimensionality reduction, and K-Means clustering. Visualizations and metrics like Silhouette Score and Inertia gives us insights into the underlying dataset structures.

Feature Engineering:

Feature engineering involves data preprocessing and transformation to ensure that the data is in a suitable format for clustering. Here's a breakdown of the key feature engineering steps:

- **Data Import :**We start by importing the necessary libraries, including Pandas for data manipulation and Scikit-Learn for machine learning operations.

  Code:
  ```
  import pandas as pd
  import numpy as np
  from sklearn.preprocessing import StandardScaler
  from sklearn.impute import SimpleImputer
  ```

- **Data Loading:** The dataset is loaded from a CSV file named 'Mall_Customers.csv.' We assume that the dataset has a structure where the first column is 'CustomerID,' and the remaining columns are features.

  Code:
  ```
  data = pd.read_csv('Mall_Customers.csv')
  ```

- **Column Extraction:** We extract the 'CustomerID' column as it may be useful for future reference.

  Code:
  ```
  customer_ids = data['CustomerID']
  ```

- **Select Numeric Features:** To perform clustering, we select only the numeric columns from the dataset, as clustering algorithms require numerical data.

  Code:
  ```
  X = data.select_dtypes(include=[np.number])
  ```

- **Handling Missing Values:** The code uses SimpleImputer to impute missing values with the mean of each feature. This ensures that the data is complete and ready for analysis.

  Code: imputer = SimpleImputer(strategy='mean')
  X_imputed = imputer.fit_transform(X)

- **Standardization:** Standardization is performed on the imputed data using StandardScaler. Standardization is crucial for t-distributed Stochastic Neighbor Embedding (t-SNE) and K-Means clustering, as these methods are sensitive to the scale of the features.

  Code: scaler = StandardScaler()
  X_scaled = scaler.fit_transform(X_imputed)

Applying Cluster Algorithms:

This describes the application of clustering algorithms to the preprocessed data. It involves the following steps:

Code: from sklearn.manifold import TSNE
from sklearn.cluster import KMeans

- **Dimensionality Reduction with t-SNE:** The code applies t-distributed Stochastic Neighbor Embedding (t-SNE) to reduce the dimensionality of the data. t-SNE is used to transform the data into a two-dimensional space, making it easier to visualize and cluster.

  Code: tsne = TSNE(n_components=2, perplexity=30, n_iter=300)
  X_tsne = tsne.fit_transform(X_scaled)

- **K-Means Clustering:** K-Means clustering is applied to the t-SNE-reduced data to group similar data points into clusters. The number of clusters is set to 3 in this example, but you can adjust it according to your problem's requirements.

  Code: kmeans = KMeans(n_clusters=3)
  cluster_labels = kmeans.fit_predict(X_tsne)

Visualization:

Data visualization is a fundamental aspect of understanding and interpreting clustering results. The code provides visualizations for both the t-SNE representation of the data and the K-Means clustering results:

Code: import matplotlib.pyplot as plt

- **t-SNE Visualization:** A scatter plot is created to visualize the two-dimensional representation of the data obtained after t-SNE dimensionality reduction.

  Code: plt.scatter(X_tsne[:, 0], X_tsne[:, 1])
  plt.title('t-SNE Visualization')
  plt.xlabel('Dimension 1')
  plt.ylabel('Dimension 2')
  plt.show()

- **K-Means Clustering Visualization:** A scatter plot is used to visualize the K-Means clustering results on the testing data. Different clusters are differentiated by color, allowing for a visual assessment of the cluster boundaries.

  Code: plt.scatter(X_test[:, 0], X_test[:, 1], c=cluster_labels_test_predicted, cmap='rainbow')
  plt.title('K-Means Clustering (train and test)')
  plt.xlabel('Dimension 1')
  plt.ylabel('Dimension 2')
  plt.show()

Interpretation:

Interpreting the results of clustering is essential to gain insights from the data. This involves the calculation of two crucial metrics:

Code:from sklearn.metrics import silhouette_score

- **Silhouette Score:** The Silhouette Score is calculated for the testing data, providing a measure of how well the data points within the clusters are separated from each other. A higher Silhouette Score indicates better cluster quality.

  Code: silhouette = silhouette_score(X_test, cluster_labels_test_predicted)
  print(f"Silhouette Score: {silhouette}")

- **Inertia:** Inertia is calculated both for the training and testing data. Inertia measures the within-cluster sum of squares, providing insight into the compactness of the clusters. Lower inertia values are generally preferred as they indicate tighter clusters.

  Code: inertia_train = kmeans.inertia_
  print(f"Inertia for Training Data: {inertia_train}")
  inertia_test = kmeans.inertia_
  print(f"Inertia for Testing Data: {inertia_test}")