PHASE 5
CUSTOMER SEGMENTATION USING DATA SCIENCE

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

DESCRIPTION:

The document and the program associated utilises customer data to perform segmentation, aiming to group customers into clusters based on their similarities. It starts by preparing and cleaning the data, followed by dimensionality reduction using t-SNE for visualisation. An initial K-Means clustering divides customers into three clusters, and a second round of K-Means clustering is performed to refine these segments. The program evaluates the clustering results using the Silhouette Score and Inertia metrics. The ultimate goal is to gain insights into customer behaviour and preferences for more targeted marketing and service strategies.

PROCESSING STEPS:

1.Importing Libraries:

The code starts by importing necessary libraries for data analysis, preprocessing, visualization, and clustering. This includes Pandas, NumPy, Matplotlib, and scikit-learn modules.

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.manifold import TSNE
from sklearn.impute import SimpleImputer
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split
from sklearn.metrics import silhouette_score
```

2. Loading the Dataset :

The code loads the "Mall_Customers.csv" dataset into a Pandas DataFrame named `data`. This dataset presumably contains customer information.

Code:

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

3. Data Preprocessing:

- customer_ids stores the "CustomerID" column.
- `X` is created to store the numerical columns from the dataset. This is typically done to exclude non-numeric or categorical features from clustering.
- Data imputation using the mean strategy is performed on `X` to handle missing values. The imputed data is stored in `X_imputed`.
- Standardisation is applied to the imputed data using `StandardScaler` to ensure that all features have the same scale. The standardized data is stored in `X_scaled`.

Code:

```
customer_ids = data['CustomerID']
X = data.select_dtypes(include=[np.number])

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

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_imputed)
```

4. Dimensionality Reduction :

t-SNE (t-distributed Stochastic Neighbour Embedding) is used for dimensionality reduction to two dimensions. The result is stored in `X_tsne`.

Code:

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

5. Data Visualization :

A scatter plot (Line 26-29) is created to visualize the t-SNE transformation, with points plotted in a two-dimensional space. This helps in understanding the data's distribution in a lower-dimensional space.

Code:

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

6. Clustering :

K-Means clustering with three clusters is applied to the t-SNE-transformed data, and the cluster labels are stored in the `cluster_labels` variable. This is an unsupervised machine learning technique that groups data points into clusters based on their similarity.

Code:

```
kmeans = KMeans(n_clusters=3)
cluster_labels = kmeans.fit_predict(X_tsne)
```

7. Updating the Dataset :

The cluster labels obtained from K-Means clustering are added as a new column, "Cluster," in the original dataset to associate each data point with its respective cluster.

Code:

```
data['Cluster'] = cluster_labels
```

8. Data Splitting :

The dataset is split into training and testing sets using `train_test_split`. This is a common practice in data science to evaluate the model's performance on unseen data.

Code:

```
X_train, X_test, cluster_labels_train, cluster_labels_test = train_test_split(X_tsne,
cluster_labels, test_size=0.2, random_state=42)
```

9. K-Means Clustering on Training Data:

A new K-Means model with three clusters is fitted to the training data (`X_train`) to further refine the cluster assignments.

Code:

```
kmeans = KMeans(n_clusters=3)
kmeans.fit(X_train)
```

10. Predicting Clusters on Test Data :

The model trained on the training data is used to predict cluster labels for the test data (`X_test`), and these predicted cluster labels are stored in `cluster_labels_test_predicted`.

Code:

```
cluster_labels_test_predicted = kmeans.predict(X_test)
```

11.Cluster Visualization :

Another scatter plot is created to visualize the K-Means clustering results on the test data. Data points are colored based on the predicted cluster labels.

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()
```

12.Silhouette Score :

The silhouette score for the test data is calculated using the `silhouette_score` function. This score is a measure of how well-separated the clusters are and can be used to evaluate the quality of the clustering.

Code:

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

13.Inertia Calculation :

Inertia values are calculated for both the training and testing data using the `inertia_` attribute of the K-Means model. Inertia measures how tightly the data points within a cluster are grouped together and can be used to assess the quality of K-Means clustering.

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}")
```

Conclusion:

This project focused on customer segmentation and analysis, enabling businesses to tailor marketing strategies. We successfully preprocessed the data, applied t-SNE dimensionality reduction, and employed K-Means clustering. Visualizations and metrics, including Silhouette Score and Inertia, provided insights into the dataset's structures.