DON BOSCO INSTITUTE OF TECHNOLOGY
Bengaluru, Karnataka – 74

TEAM COUNT: THREE

PROJECT TITLE: Customer journey analysis using clustering and dimensionality reduction enhancing user experience

TEAM LEAD NAME: AFTAF AYUB MANIYAR

TEAM LEAD CAN ID: CAN_33697094

1) NAME: AFTAF A MANIYAR
CA N ID: CAN_33697094
ROLE: MACHINE LEARNING

2) NAME: HARSHA N A
CAN ID: CAN_33738478
ROLE: DATA ANALYST

3) NAME: RAKSHITH K L
CAN ID: CAN_33116494
ROLE: PROJECT MANAGER

4)GOKUL KANNAN P
CAN ID:CAN_33196444
ROLE:INFORMATION

Advanced Market Segmentation Using Deep Clustering

Phase 3: Model Training and Evaluation

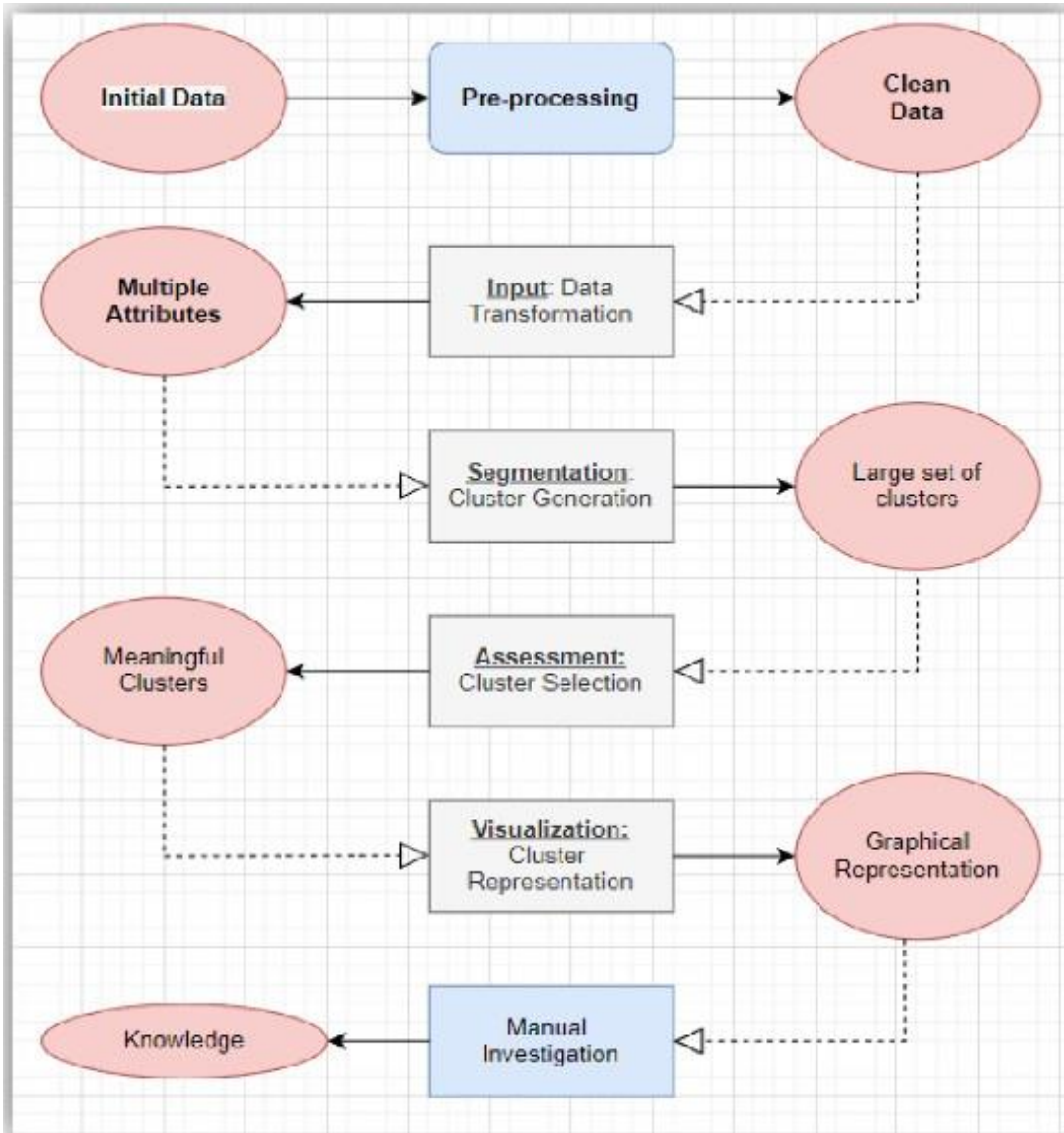
3.1 Overview of Model Training and Evaluation

In this phase, we focus on selecting suitable algorithms, training the models using the processed data, and evaluating their performance. We aim to choose algorithms that are well-suited for deep clustering and market segmentation tasks. Hyper parameter tuning is performed to optimize model performance, and various evaluation metrics are employed to assess the model's predictive capabilities. Cross-validation is also performed to ensure that the model generalizes well to unseen data.

3.2 Choosing Suitable Algorithms

For the **Advanced Market Segmentation using Deep Clustering** project, the key algorithms are:

1. **Autoencoder (for feature extraction and dimensionality reduction)** – This deep learning model is used to encode customer data into a lower-dimensional latent space.
2. **K-Means Clustering (for customer segmentation)** – After dimensionality reduction, K-Means clustering is applied to group customers into distinct segments.



Source code :

```
# Import necessary libraries
import numpy as np import
tensorflow as tf
from sklearn.preprocessing import StandardScaler from
sklearn.cluster import KMeans from sklearn.metrics import
silhouette_score, adjusted_rand_score

# Assume 'data' is the dataset that has been preprocessed (scaled, cleaned)

# Step 1: Train an Autoencoder for feature extraction autoencoder
= tf.keras.Sequential([
tf.keras.layers.InputLayer(input_shape=(data.shape[1],)),
tf.keras.layers.Dense(128, activation='relu'),
tf.keras.layers.Dense(64, activation='relu'),
```

```
    tf.keras.layers.Dense(32, activation='relu'), # Latent space


    tf.keras.layers.Dense(64, activation='relu'), tf.keras.layers.Dense(128,
    activation='relu'),
    tf.keras.layers.Dense(data.shape[1], activation='sigmoid')
])
autoencoder.compile(optimizer='adam', loss='mean_squared_error')
autoencoder.fit(data_scaled, data_scaled, epochs=50, batch_size=256, validation_split=0.2)

# Step 2: Extract latent features
latent_features = autoencoder.predict(data_scaled)

# Step 3: Apply K-Means clustering to the latent features
kmeans   =   KMeans(n_clusters=5,   random_state=42)
clusters = kmeans.fit_predict(latent_features)

# Step 4: Evaluate the clustering quality
silhouette_avg = silhouette_score(latent_features, clusters) print("Silhouette
Score:", silhouette_avg)
```

3.3 Hyperparameter Tuning

Hyperparameter tuning is a crucial step to ensure that the model performs optimally. In this project, we will perform **grid search** for the K-Means algorithm to find the best number of clusters. Additionally, the **autoencoder** model's architecture and training parameters (e.g., learning rate, batch size) can be tuned using techniques like **random search** or **Bayesian optimization**.

Source code for grid search for K-Means to find the best number of clusters

```
# Import necessary libraries
import numpy as np import
tensorflow as tf
from sklearn.preprocessing import StandardScaler

from sklearn.cluster import KMeans from sklearn.metrics import
silhouette_score, adjusted_rand_score

# Assume 'data' is the dataset that has been preprocessed (scaled, cleaned)

# Step 1: Train an Autoencoder for feature extraction autoencoder
= tf.keras.Sequential([
tf.keras.layers.InputLayer(input_shape=(data.shape[1],)),
tf.keras.layers.Dense(128, activation='relu'),
tf.keras.layers.Dense(64, activation='relu'),
```

```
    tf.keras.layers.Dense(32, activation='relu'), # Latent space
    tf.keras.layers.Dense(64,                    activation='relu'),
    tf.keras.layers.Dense(128,                   activation='relu'),
    tf.keras.layers.Dense(data.shape[1], activation='sigmoid')
])autoencoder.compile(optimizer='adam', loss='mean_squared_error') autoencoder.fit(data_scaled,
data_scaled, epochs=50, batch_size=256, validation_split=0.2)
# Step 2: Extract latent features
latent_features = autoencoder.predict(data_scaled)

# Step 3: Apply K-Means clustering to the latent features
kmeans   =   KMeans(n_clusters=5,   random_state=42)
clusters = kmeans.fit_predict(latent_features)

# Step 4: Evaluate the clustering quality
silhouette_avg = silhouette_score(latent_features, clusters) print("Silhouette
Score:", silhouette_avg)
```

3.4 Model Evaluation Metrics

The performance of the model is evaluated using several metrics that measure clustering quality and the reconstruction accuracy of the autoencoder. These include:

1. **Silhouette Score** – Measures how similar data points are within their cluster compared to other clusters. A higher score indicates better clustering.

   Source code :

```
silhouette_avg = silhouette_score(latent_features, clusters) print("Silhouette
Score:", silhouette_avg)
```

Adjusted Rand Index (ARI) – Measures the similarity between the predicted clusters and ground truth labels, adjusting for chance. ARI values closer to 1 indicate better alignment with true labels.

Source code :

```
from sklearn.metrics import adjusted_rand_score

true_labels = _ # Replace with your actual ground truth labels
ari_score   =   adjusted_rand_score(true_labels,   clusters)
print(f"Adjusted Rand Index: {ari_score:.2f}")
```

Mean Squared Error (MSE) – Measures the difference between the original and reconstructed data, indicating how well the autoencoder captures the data's structure.
Source code :

```
# Predict reconstructed data
reconstructed_data = autoencoder.predict(data_scaled)

# Calculate mean squared error (MSE) between original and reconstructed data
reconstruction_loss = np.mean(np.square(data_scaled - reconstructed_data))
print(f"Reconstruction Loss: {reconstruction_loss:.4f}")
```

3.5 Cross-Validation

Cross-validation is performed to assess the model's generalizability and ensure it is not overfitting to the training data. Since clustering does not inherently provide a validation set, techniques such as **K-Fold cross-validation** can be adapted by splitting the data into multiple subsets, training the model on some subsets, and testing it on others.

Source code:

```
from sklearn.model_selection import KFold from
sklearn.metrics import silhouette_score

# Define KFold cross-validation kf = KFold(n_splits=5,
shuffle=True, random_state=42) silhouette_scores = []

# Perform cross-validation for train_index, test_index
in kf.split(latent_features):
    X_train, X_test = latent_features[train_index], latent_features[test_index] y_train,
    y_test = clusters[train_index], clusters[test_index]

    # Fit KMeans on the training data
    kmeans = KMeans(n_clusters=best_n_clusters, random_state=42)
    kmeans.fit(X_train)

    # Predict clusters on the test set clusters_pred
    = kmeans.predict(X_test)

    # Evaluate clustering quality using Silhouette Score
    score = silhouette_score(X_test, clusters_pred)
    silhouette_scores.append(score)

# Average Silhouette Score across all folds avg_silhouette_score
= np.mean(silhouette_scores)
print(f"Average Silhouette Score from cross-validation: {avg_silhouette_score:.4f}")
```

3.6 Conclusion of Phase 3

In Phase 3, the model was trained using the autoencoder for dimensionality reduction and K-Means for clustering. We tuned the K-Means clustering algorithm's hyperparameters using grid search and evaluated the model's performance using several metrics, including silhouette score, adjusted Rand index, and reconstruction loss. Cross-validation was applied to assess the model's robustness and ensure generalizability. The evaluation metrics provided insights into the clustering quality and the effectiveness of the autoencoder in capturing the underlying data patterns.