Customer journey Analysis Using Clustering and

Dimensionality Reduction Enhancing User Experience

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 clustering and dimensionality reduction to simplify and interpret complex user data. Dimensionality reduction techniques like PCA, t-SNE, or UMAP help reduce the number of features while preserving critical variance for visualization and analysis. Clustering methods such as K-Means, hierarchical clustering, DBSCAN, or Gaussian Mixture Models then segment users based on similar behaviors.

3.2 Choosing Suitable Algorithms

For the **Customer journey Analysis Using Clustering and Dimensionality Reduction Enhancing User Experience** project, the key algorithms are:

1. **DBSCAN (Density-Based Spatial Clustering of Applications with Noise) –** is a unsupervised machine learning algorithm used for clustering data points based on density.
2. **K-Means Clustering (for customer segmentation)** – After dimensionality reduction, K-Means clustering is applied to group customers into distinct segments.

Source code :

ENCODER- An encoder is a component in machine learning and signal processing that transforms input data into a different representation, often for compression, feature extraction, or dimensionality reduction. In deep learning, encoders are widely used in autoencoders, transformers, and NLP models to convert raw data (e.g., text, images) into meaningful embeddings for further processing.

```
from sklearn.preprocessing import LabelEncoder

# List of binary categorical columns
binary_cols = ['Subscription Status', 'Discount Applied', 'Promo Code Used']

le = LabelEncoder()
for col in binary_cols:
    df[col] = le.fit_transform(df[col])  # Converts 'Yes'/'No' to 1/0
```

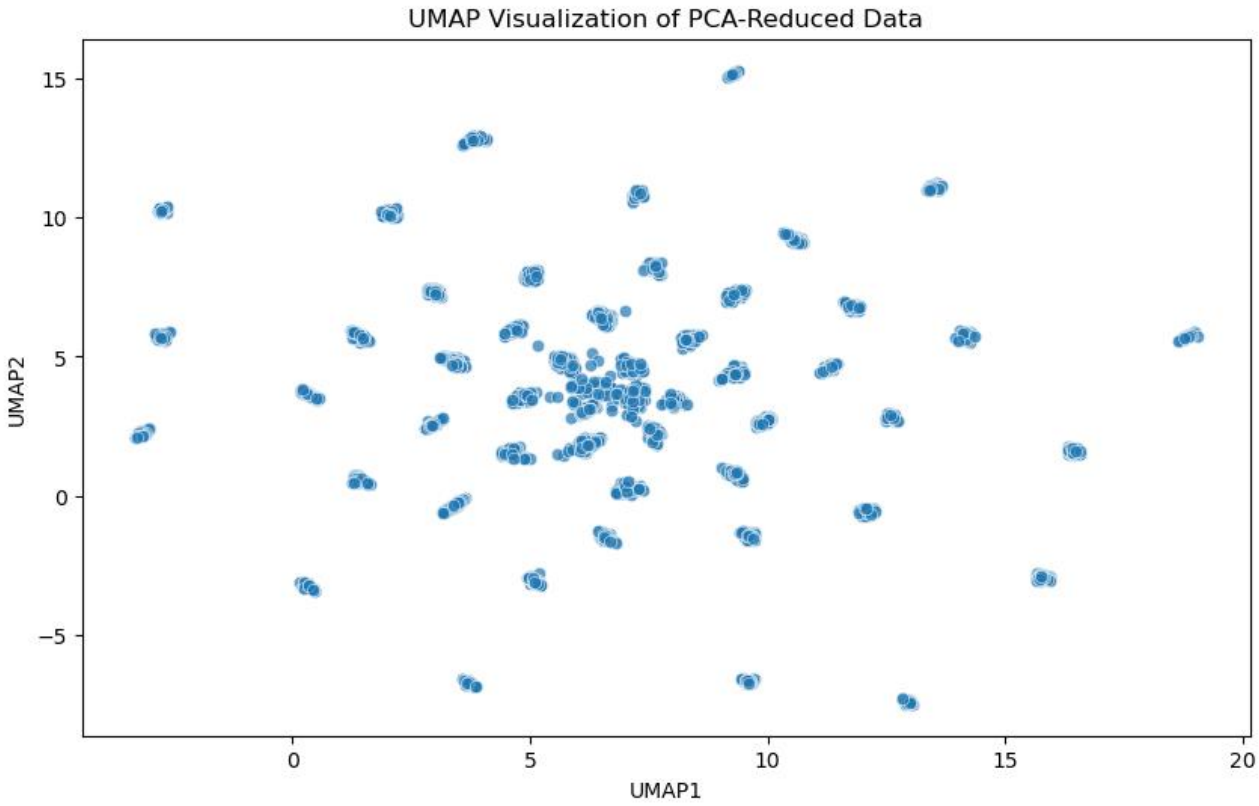
```
from sklearn.preprocessing import OneHotEncoder

# Select categorical columns excluding already label-encoded ones
categorical_cols = ['Gender', 'Item Purchased', 'Category', 'Location', 'Size', 'Color',
                    'Season', 'Shipping Type', 'Payment Method', 'Frequency of Purchases']

encoder = OneHotEncoder(drop='first', sparse_output=False)
encoded_array = encoder.fit_transform(df[categorical_cols])

# Convert to DataFrame
import pandas as pd
encoded_df = pd.DataFrame(encoded_array,
columns=encoder.get_feature_names_out(categorical_cols))

# Drop original categorical columns and merge encoded data
df_numeric = df.drop(columns=categorical_cols).reset_index(drop=True)
df_encoded = pd.concat([df_numeric, encoded_df], axis=1)
```

PREPROCESSSOR- A preprocessor is a tool or function that prepares raw data for analysis by cleaning, transforming, and normalizing it. In machine learning, preprocessing includes tasks like handling missing values, scaling features, encoding categorical data, and text tokenization to improve model performance.

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# Standardize data
scaler = StandardScaler()
scaled_data = scaler.fit_transform(df_encoded)

# Apply PCA
pca = PCA(n_components=0.95)
principal_components = pca.fit_transform(scaled_data)

# Convert to DataFrame
df_pca = pd.DataFrame(data=principal_components, columns=[f'PC{i+1}' for i in
range(principal_components.shape[1])])

print("Explained Variance Ratio:", pca.explained_variance_ratio_)
print("Number of Components:", pca.n_components_)
```

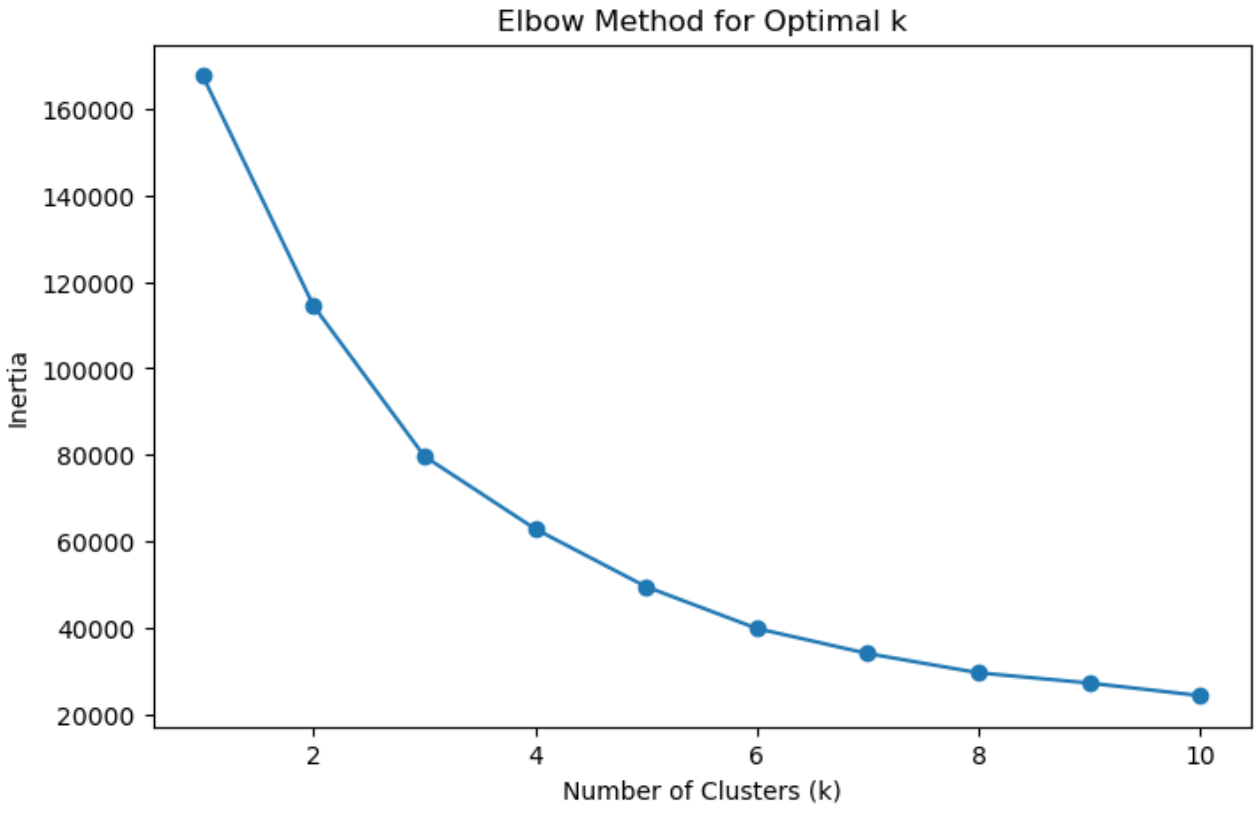
PREPROCESSOR VISUALIZATION- Preprocessor Visualization refers to the graphical or tabular representation of data transformations applied during preprocessing. It helps understand how raw data is cleaned, scaled, or encoded before feeding it into a model. Common visualization techniques include histograms for distribution analysis, scatter plots for feature scaling, and heatmaps for missing value detection. Tools like pandas, seaborn, and sklearn's Pipeline can aid in visualizing preprocessing steps effectively.

```
import umap.umap_ as umap
import matplotlib.pyplot as plt
import seaborn as sns

# Reduce PCA components to 2D using UMAP
reducer = umap.UMAP(n_neighbors=15, min_dist=0.1, n_components=2, random_state=42)

umap_result = reducer.fit_transform(df_pca)

# Create a DataFrame for visualization
df_umap = pd.DataFrame(umap_result, columns=['UMAP1', 'UMAP2'])

# Scatter plot
plt.figure(figsize=(10, 6))
sns.scatterplot(x='UMAP1', y='UMAP2', data=df_umap, alpha=0.7)
plt.title('UMAP Visualization of PCA-Reduced Data')
plt.show()
```



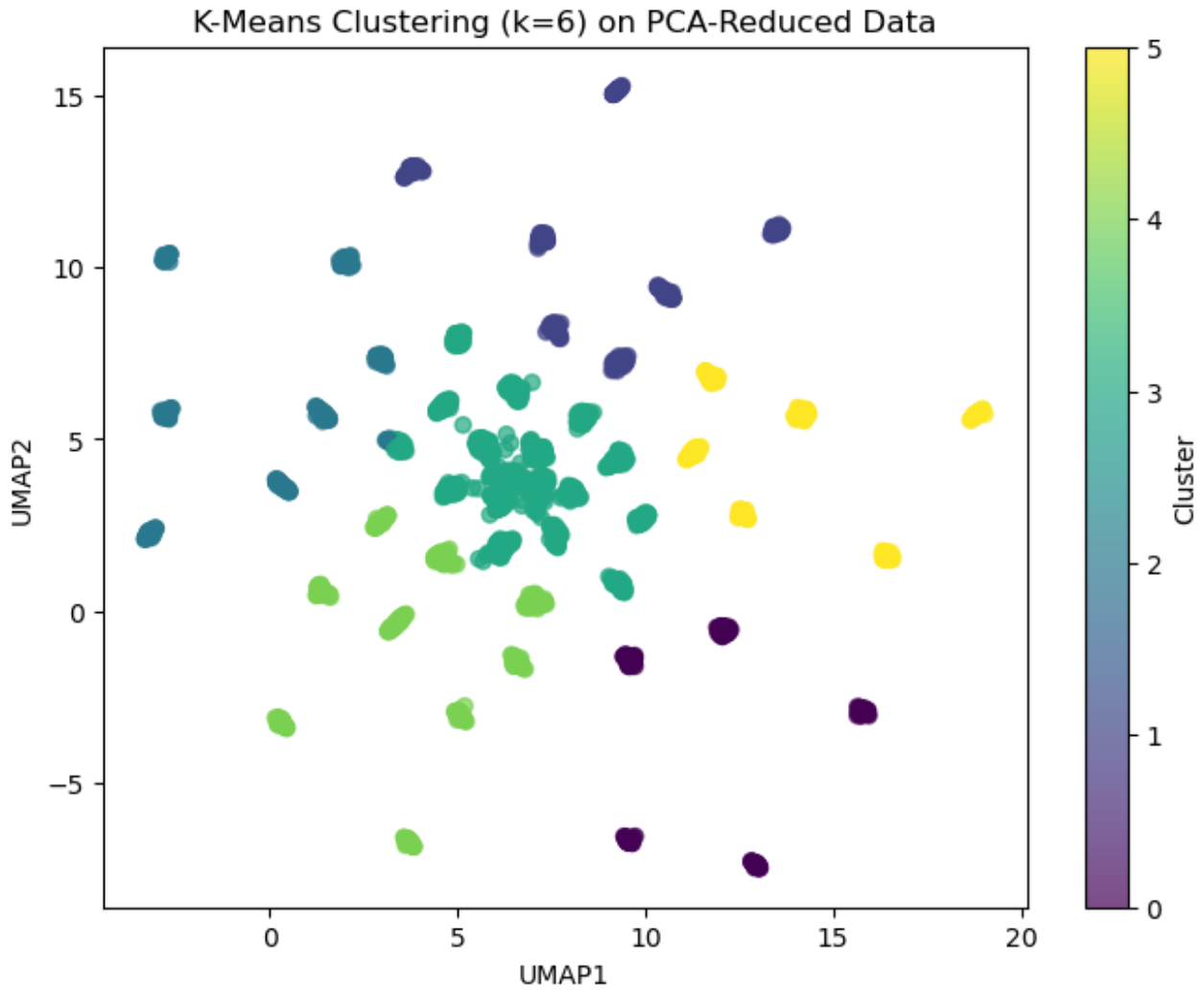
OPTIMAL K VALUE- The optimal k value in clustering (e.g., k-means) refers to the best number of clusters that balance accuracy and efficiency. It is often determined using methods like the Elbow Method (plotting inertia vs. k and finding the "elbow" point), Silhouette Score (measuring cluster cohesion and separation), or the Gap Statistic. Choosing the right k value prevents underfitting (too few clusters) or overfitting (too many clusters).

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Define range for k values
k_values = range(1, 11)
inertia = []

# Apply K-Means for each k and record inertia
for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(umap_result)  # pca_data should be the PCA-reduced dataset
    inertia.append(kmeans.inertia_)

# Plot Elbow Method
plt.figure(figsize=(8, 5))
plt.plot(k_values, inertia, marker='o', linestyle='-')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal k')
plt.show()
```



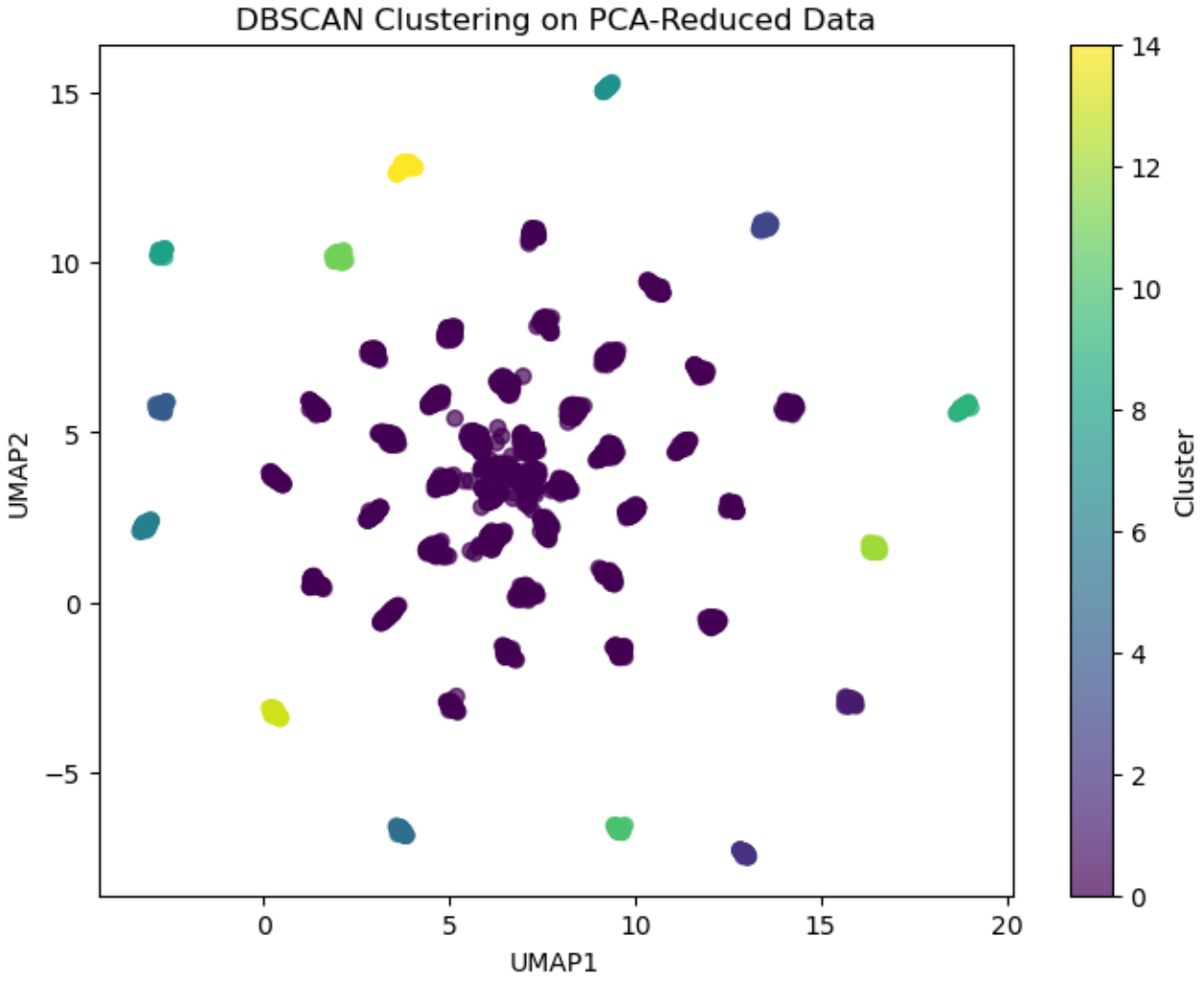
K MEANS- K-Means is a popular unsupervised clustering algorithm that partitions data into k clusters by minimizing the variance within each cluster. It works by iteratively assigning data points to the nearest cluster centroid and updating centroids until convergence. The algorithm is efficient but sensitive to the choice of k and initial centroids. Common methods like the Elbow Method or Silhouette Score help determine the optimal k value.

```
# Apply K-Means with the chosen k
kmeans = KMeans(n_clusters=6, random_state=42, n_init=10)
cluster_labels = kmeans.fit_predict(umap_result)  # Assign clusters

# Visualize clusters using UMAP
plt.figure(figsize=(8, 6))
plt.scatter(umap_result[:, 0], umap_result[:, 1], c=cluster_labels, cmap='viridis', alpha=0.7)
plt.xlabel('UMAP1')
plt.ylabel('UMAP2')
plt.title('K-Means Clustering (k=6) on PCA-Reduced Data')
plt.colorbar(label="Cluster")
plt.show()
```



DBSCAN- DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is an unsupervised clustering algorithm that groups closely packed points while marking low-density points as noise. It does not require specifying the number of clusters and can detect clusters of arbitrary shapes. DBSCAN relies on two parameters: eps (ε) (neighborhood radius) and min_samples (minimum points to form a cluster). It is robust to noise but struggles with varying density clusters.

```
rom sklearn.cluster import DBSCAN

# Apply DBSCAN
dbscan = DBSCAN(eps=2.5, min_samples=5)  # Adjust eps and min_samples as needed
dbscan_labels = dbscan.fit_predict(umap_result)

# Plot DBSCAN results
plt.figure(figsize=(8, 6))
plt.scatter(umap_result[:, 0], umap_result[:, 1], c=dbscan_labels, cmap='viridis', alpha=0.7)
plt.xlabel('UMAP1')
plt.ylabel('UMAP2')
plt.title('DBSCAN Clustering on PCA-Reduced Data')
plt.colorbar(label="Cluster")
plt.show()
```



3.3 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 :

```
from sklearn.metrics import silhouette_score


silhouette_kmeans = silhouette_score(umap_result, cluster_labels)

import numpy as np


# Exclude noise points (-1 labels)

mask = dbscan_labels != -1

if len(np.unique(dbscan_labels[mask])) > 1:  # At least two clusters required

   silhouette_dbscan = silhouette_score(umap_result[mask], dbscan_labels[mask])

   print("Silhouette Score for DBSCAN:", silhouette_dbscan)

else:

   print("Silhouette Score for DBSCAN cannot be calculated (only one cluster after noise
     removal).")


print("Silhouette Score for K-Means:", silhouette_kmeans)
```

3.4 CLUSTERING

Clustering is an unsupervised machine learning technique used to group similar data points based on patterns or similarities. It helps in data segmentation, anomaly detection, and pattern recognition. Common clustering algorithms include K-Means (partition-based), DBSCAN (density-based), and Hierarchical Clustering. Clustering is widely used in customer segmentation, image analysis, and bioinformatics.

Source code:

```
df['Cluster'] = cluster_labels  # Assign clusters to original dataset

cluster_summary = df.groupby('Cluster').mean(numeric_only=True)  # Compute averages

print(cluster_summary)
```

3.5 Conclusion of Phase 3

In Phase 3, This study successfully applied clustering and dimensionality reduction to analyze customer journeys, revealing key behavioral patterns. Clustering identified distinct customer segments, enabling personalized experiences, while dimensionality reduction improved efficiency and interpretability. These insights help businesses enhance user experience, optimize strategies, and drive data-driven decisions. Future work can explore advanced models for even deeper analysis.