Step 1: Feature Selection

Note: We will only use unsupervised feature selection techniques.

**1.1 Variance Threshold**

python

Copy code

from sklearn.feature\_selection import VarianceThreshold

def variance\_threshold\_selector(data, threshold=0.8):

selector = VarianceThreshold(threshold=(threshold \* (1 - threshold)))

return selector.fit\_transform(data)

Z\_var = variance\_threshold\_selector(Z)

**1.2 Correlation Matrix**

python

Copy code

import pandas as pd

import numpy as np

def correlation\_threshold\_selector(data, threshold=0.95):

correlation\_matrix = pd.DataFrame(data).corr().abs()

upper = correlation\_matrix.where(np.triu(np.ones(correlation\_matrix.shape), k=1).astype(bool))

to\_drop = [column for column in upper.columns if any(upper[column] > threshold)]

return pd.DataFrame(data).drop(columns=to\_drop).values

Z\_corr = correlation\_threshold\_selector(Z)

**Step 2: Dimensionality Reduction**

**2.1 PCA**

python

Copy code

from sklearn.decomposition import PCA

def apply\_pca(data, n\_components=2):

pca = PCA(n\_components=n\_components)

return pca.fit\_transform(data)

Z\_pca = apply\_pca(Z)

**2.2 MDS**

python

Copy code

from sklearn.manifold import MDS

def apply\_mds(data, n\_components=2):

mds = MDS(n\_components=n\_components, random\_state=42)

return mds.fit\_transform(data)

Z\_mds = apply\_mds(Z)

**2.3 LLE**

python

Copy code

from sklearn.manifold import LocallyLinearEmbedding

def apply\_lle(data, n\_components=2):

lle = LocallyLinearEmbedding(n\_components=n\_components, random\_state=42)

return lle.fit\_transform(data)

Z\_lle = apply\_lle(Z)

**2.4 MCA**

python

Copy code

from prince import MCA

def apply\_mca(data, n\_components=2):

mca = MCA(n\_components=n\_components)

return mca.fit\_transform(data).values

Z\_mca = apply\_mca(Z)

**2.5 t-SNE**

python

Copy code

from sklearn.manifold import TSNE

def apply\_tsne(data, n\_components=2):

tsne = TSNE(n\_components=n\_components, random\_state=42)

return tsne.fit\_transform(data)

Z\_tsne = apply\_tsne(Z)

**2.6 UMAP**

python

Copy code

import umap

def apply\_umap(data, n\_components=2):

umap\_model = umap.UMAP(n\_components=n\_components, random\_state=42)

return umap\_model.fit\_transform(data)

Z\_umap = apply\_umap(Z)

**Step 3: Clustering Techniques**

**3.1 K-Means Clustering**

python

Copy code

from sklearn.cluster import KMeans

from sklearn.metrics import silhouette\_score, davies\_bouldin\_score

def kmeans\_clustering(data, n\_clusters=3):

kmeans = KMeans(n\_clusters=n\_clusters, random\_state=42)

labels = kmeans.fit\_predict(data)

silhouette\_avg = silhouette\_score(data, labels)

davies\_bouldin\_avg = davies\_bouldin\_score(data, labels)

return labels, silhouette\_avg, davies\_bouldin\_avg

labels\_kmeans, silhouette\_kmeans, davies\_bouldin\_kmeans = kmeans\_clustering(Z\_pca)

print(f'K-Means: Silhouette Score = {silhouette\_kmeans}, Davies-Bouldin Index = {davies\_bouldin\_kmeans}')

**3.2 Hierarchical Clustering (Agglomerative)**

python

Copy code

from sklearn.cluster import AgglomerativeClustering

def hierarchical\_clustering(data, n\_clusters=3):

hierarchical = AgglomerativeClustering(n\_clusters=n\_clusters)

labels = hierarchical.fit\_predict(data)

silhouette\_avg = silhouette\_score(data, labels)

davies\_bouldin\_avg = davies\_bouldin\_score(data, labels)

return labels, silhouette\_avg, davies\_bouldin\_avg

labels\_hierarchical, silhouette\_hierarchical, davies\_bouldin\_hierarchical = hierarchical\_clustering(Z\_pca)

print(f'Hierarchical Clustering: Silhouette Score = {silhouette\_hierarchical}, Davies-Bouldin Index = {davies\_bouldin\_hierarchical}')

**3.3 Gaussian Mixture Models (GMM)**

python

Copy code

from sklearn.mixture import GaussianMixture

def gmm\_clustering(data, n\_components=3):

gmm = GaussianMixture(n\_components=n\_components, random\_state=42)

labels = gmm.fit\_predict(data)

silhouette\_avg = silhouette\_score(data, labels)

davies\_bouldin\_avg = davies\_bouldin\_score(data, labels)

return labels, silhouette\_avg, davies\_bouldin\_avg

labels\_gmm, silhouette\_gmm, davies\_bouldin\_gmm = gmm\_clustering(Z\_pca)

print(f'GMM: Silhouette Score = {silhouette\_gmm}, Davies-Bouldin Index = {davies\_bouldin\_gmm}')

**3.4 DBSCAN**

python

Copy code

from sklearn.cluster import DBSCAN

def dbscan\_clustering(data, eps=0.5, min\_samples=5):

dbscan = DBSCAN(eps=eps, min\_samples=min\_samples)

labels = dbscan.fit\_predict(data)

silhouette\_avg = silhouette\_score(data, labels)

davies\_bouldin\_avg = davies\_bouldin\_score(data, labels)

return labels, silhouette\_avg, davies\_bouldin\_avg

labels\_dbscan, silhouette\_dbscan, davies\_bouldin\_dbscan = dbscan\_clustering(Z\_pca)

print(f'DBSCAN: Silhouette Score = {silhouette\_dbscan}, Davies-Bouldin Index = {davies\_bouldin\_dbscan}')

**3.5 Mean-Shift Clustering**

python

Copy code

from sklearn.cluster import MeanShift

def mean\_shift\_clustering(data):

mean\_shift = MeanShift()

labels = mean\_shift.fit\_predict(data)

silhouette\_avg = silhouette\_score(data, labels)

davies\_bouldin\_avg = davies\_bouldin\_score(data, labels)

return labels, silhouette\_avg, davies\_bouldin\_avg

labels\_mean\_shift, silhouette\_mean\_shift, davies\_bouldin\_mean\_shift = mean\_shift\_clustering(Z\_pca)

print(f'Mean-Shift Clustering: Silhouette Score = {silhouette\_mean\_shift}, Davies-Bouldin Index = {davies\_bouldin\_mean\_shift}')

**3.6 Spectral Clustering**

python

Copy code

from sklearn.cluster import SpectralClustering

def spectral\_clustering(data, n\_clusters=3):

spectral = SpectralClustering(n\_clusters=n\_clusters, random\_state=42)

labels = spectral.fit\_predict(data)

silhouette\_avg = silhouette\_score(data, labels)

davies\_bouldin\_avg = davies\_bouldin\_score(data, labels)

return labels, silhouette\_avg, davies\_bouldin\_avg

labels\_spectral, silhouette\_spectral, davies\_bouldin\_spectral = spectral\_clustering(Z\_pca)

print(f'Spectral Clustering: Silhouette Score = {silhouette\_spectral}, Davies-Bouldin Index = {davies\_bouldin\_spectral}')

**3.7 Birch Clustering**

python

Copy code

from sklearn.cluster import Birch

def birch\_clustering(data, n\_clusters=3):

birch = Birch(n\_clusters=n\_clusters)

labels = birch.fit\_predict(data)

silhouette\_avg = silhouette\_score(data, labels)

davies\_bouldin\_avg = davies\_bouldin\_score(data, labels)

return labels, silhouette\_avg, davies\_bouldin\_avg

labels\_birch, silhouette\_birch, davies\_bouldin\_birch = birch\_clustering(Z\_pca)

print(f'Birch Clustering: Silhouette Score = {silhouette\_birch}, Davies-Bouldin Index = {davies\_bouldin\_birch}')

Step 4: Evaluate Clustering Techniques

**4.1 Silhouette Scores and Elbow Criterion**

For each clustering technique, we have already computed the silhouette scores and Davies-Bouldin index in the above steps.

Step 5: Visualize Clusters

**5.1 t-SNE Visualization**

python

Copy code

import matplotlib.pyplot as plt

import seaborn as sns

def visualize\_clusters\_tsne(data, labels, title):

tsne = TSNE(n\_components=2, random\_state=42)

data\_tsne = tsne.fit\_transform(data)

plt.figure(figsize=(10, 8))

sns.scatterplot(data\_tsne[:, 0], data\_tsne[:, 1], hue=labels, palette="viridis")

plt.title(f"t-SNE Visualization of Clusters ({title})")

plt.show()

visualize\_clusters\_tsne(Z, labels\_kmeans, "K-Means")

visualize\_clusters\_tsne(Z, labels\_hierarchical, "Hierarchical Clustering")

visualize\_clusters\_tsne(Z, labels\_gmm, "GMM")

visualize\_clusters\_tsne(Z, labels\_dbscan, "DBSCAN")

visualize\_clusters\_tsne(Z, labels\_mean\_shift, "Mean-Shift")

visualize\_clusters\_tsne(Z, labels\_spectral, "Spectral Clustering")

visualize\_clusters\_tsne(Z, labels\_birch, "Birch Clustering")

**5.2 UMAP Visualization**

python

Copy code

def visualize\_clusters\_umap(data, labels, title):

umap\_model = umap.UMAP(n\_components=2, random\_state=42)

data\_umap = umap\_model.fit\_transform(data)

plt.figure(figsize=(10, 8))

sns.scatterplot(data\_umap[:, 0], data\_umap[:, 1], hue=labels, palette="viridis")

plt.title(f"UMAP Visualization of Clusters ({title})")

plt.show()

visualize\_clusters\_umap(Z, labels\_kmeans, "K-Means")

visualize\_clusters\_umap(Z, labels\_hierarchical, "Hierarchical Clustering")

visualize\_clusters\_umap(Z, labels\_gmm, "GMM")

visualize\_clusters\_umap(Z, labels\_dbscan, "DBSCAN")

visualize\_clusters\_umap(Z, labels\_mean\_shift, "Mean-Shift")

visualize\_clusters\_umap(Z, labels\_spectral, "Spectral Clustering")

visualize\_clusters\_umap(Z, labels\_birch, "Birch Clustering")

**Step 6: Cluster Profiling**

python

Copy code

def cluster\_profiling(data, labels):

df = pd.DataFrame(data)

df['Cluster'] = labels

# Calculate cluster means

cluster\_means = df.groupby('Cluster').mean()

return cluster\_means

profile\_kmeans = cluster\_profiling(Z, labels\_kmeans)

profile\_hierarchical = cluster\_profiling(Z, labels\_hierarchical)

profile\_gmm = cluster\_profiling(Z, labels\_gmm)

profile\_dbscan = cluster\_profiling(Z, labels\_dbscan)

profile\_mean\_shift = cluster\_profiling(Z, labels\_mean\_shift)

profile\_spectral = cluster\_profiling(Z, labels\_spectral)

profile\_birch = cluster\_profiling(Z, labels\_birch)

print("K-Means Cluster Profiling:\n", profile\_kmeans)

print("Hierarchical Clustering Cluster Profiling:\n", profile\_hierarchical)

print("GMM Cluster Profiling:\n", profile\_gmm)

print("DBSCAN Cluster Profiling:\n", profile\_dbscan)

print("Mean-Shift Clustering Cluster Profiling:\n", profile\_mean\_shift)

print("Spectral Clustering Cluster Profiling:\n", profile\_spectral)

print("Birch Clustering Cluster Profiling:\n", profile\_birch)

This comprehensive workflow will help you in performing feature selection, dimensionality reduction, and clustering on your dataset, as well as evaluating and visualizing the results. You can now implement these steps sequentially and refer back for specific issues or enhancements.