Lab Assignments on Clustering and Proximity Analysis

Assignment 1: Metric and Non-Metric Proximity

Dataset: *Iris Dataset (UCI Machine Learning Repository)*

1. Compute the pairwise metric proximity (Euclidean, Manhattan) for the dataset.
2. Compute non-metric proximities (Jaccard similarity for binary-encoded attributes, Cosine similarity for feature vectors).
3. Visualize the proximity matrices using a heatmap.

Assignment 2: Density Estimation

Dataset: *Wine Quality Dataset (UCI Machine Learning Repository)*

1. Implement the **Parzen Window** method to estimate the density for the "alcohol" feature. Experiment with different window sizes (h).
2. Use the **Nearest Neighbor** density estimation method with k = 5, 10, and 20.
3. Visualize the density curves for both methods.

Assignment 3: Hierarchical Clustering

Dataset: *Mall Customers Dataset (Kaggle)*

1. Perform **Agglomerative Hierarchical Clustering** on "Annual Income" and "Spending Score".
   - Use different linkage methods (single, complete, average).

- Visualize the dendrogram and identify clusters.

2. Implement **Divisive Clustering** with the ISODATA approach.
    - Experiment with different thresholds for cluster merging/splitting.

Assignment 4: Fuzzy C-Means Clustering

Dataset: *Human Activity Recognition Dataset (UCI)*

1. Implement **Fuzzy C-Means Clustering** to group activity types based on accelerometer data.
2. Experiment with different numbers of clusters (e.g., 3, 5, 7).
3. Visualize membership values for each data point using a scatter plot.
4. Compare Fuzzy C-Means with traditional K-Means in terms of clustering accuracy and boundary overlap.

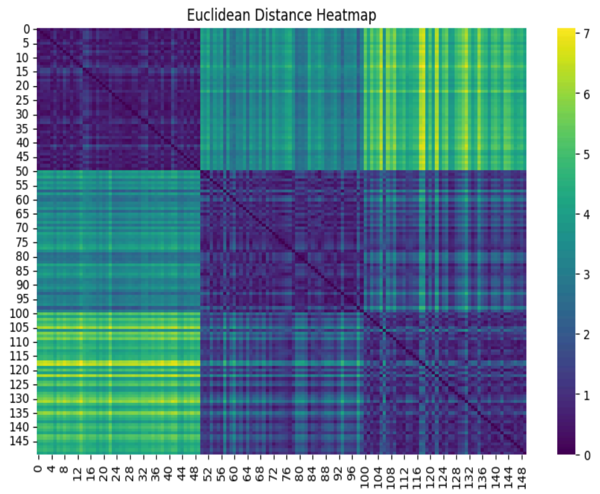
Assignment 11

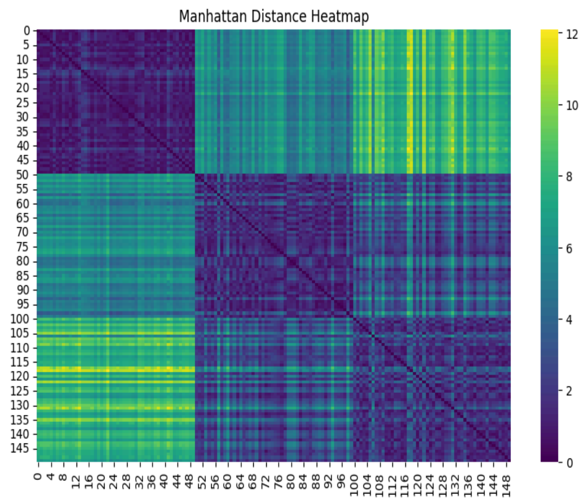
Clustering and Proximity Analysis

Assignment 1: Metric and Non-Metric Proximity
Dataset: Iris Dataset (UCI Machine Learning Repository)
1. Compute the pairwise metric proximity (Euclidean, Manhattan) for the dataset. Visualize the proximity matrices using a heatmap.

```
import numpy as np
import pandas as pd
from scipy.spatial.distance import pdist, squareform
import seaborn as sns
import matplotlib.pyplot as plt
# Load Iris dataset
from sklearn.datasets import load_iris
iris = load_iris()
data = iris.data
# Compute pairwise Euclidean and Manhattan distances
euclidean_dist = squareform(pdist(data, metric='euclidean'))
manhattan_dist = squareform(pdist(data, metric='cityblock'))
# Visualization of Euclidean
plt.figure(figsize=(10, 6))
sns.heatmap(euclidean_dist, cmap='viridis', cbar=True)
plt.title('Euclidean Distance Heatmap')
plt.show()
# Visualization of Manhattan
plt.figure(figsize=(10, 6))
sns.heatmap(manhattan_dist, cmap='viridis', cbar=True)
plt.title('Manhattan Distance Heatmap')
plt.show()
```

Output:

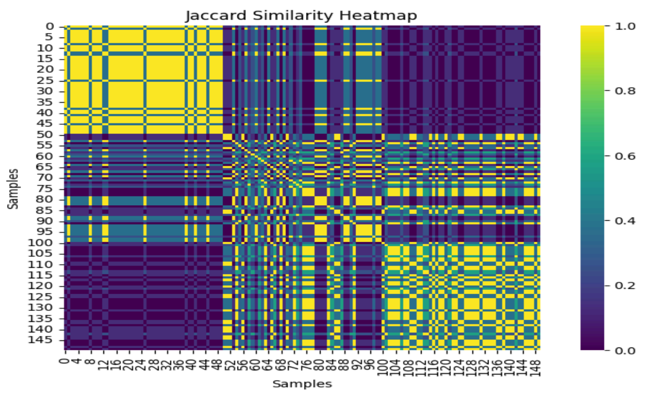


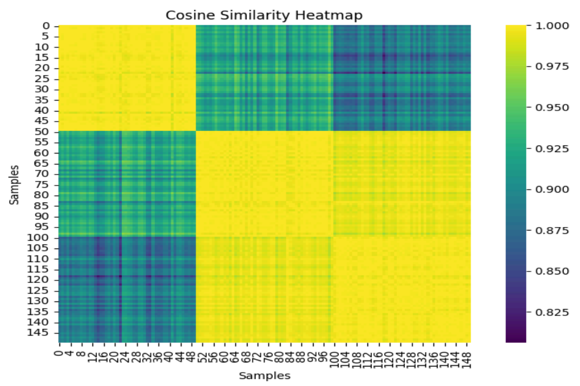


2. Compute non-metric proximities (Jaccard similarity for binary-encoded attributes, Cosine similarity for feature vectors). Visualize the proximity matrices using a heatmap.

```
from sklearn.metrics import jaccard_score
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
# Load dataset (example: Iris Dataset for numeric data)
from sklearn.datasets import load_iris
data = load_iris().data  # Replace with your dataset
# Convert data to binary for Jaccard
binary_data = np.where(data > np.median(data, axis=0), 1, 0)
# Jaccard similarity (pairwise comparison across all rows)
n_samples = binary_data.shape[0]
jaccard_sim = np.zeros((n_samples, n_samples))
for i in range(n_samples):
    for j in range(n_samples):
        jaccard_sim[i, j] = jaccard_score(binary_data[i], binary_data[j],
average='macro')
cosine_sim = cosine_similarity(data)
# Visualization of Jaccard Similarity
plt.figure(figsize=(10, 6))
sns.heatmap(jaccard_sim, cmap='viridis', cbar=True, square=True)
plt.title('Jaccard Similarity Heatmap')
plt.xlabel('Samples')
plt.ylabel('Samples')
plt.show()
# Visualization of Cosine Similarity
plt.figure(figsize=(10, 6))
sns.heatmap(cosine_sim, cmap='viridis', cbar=True, square=True)
plt.title('Cosine Similarity Heatmap')
plt.xlabel('Samples')
plt.ylabel('Samples')
plt.show()
```

Output:





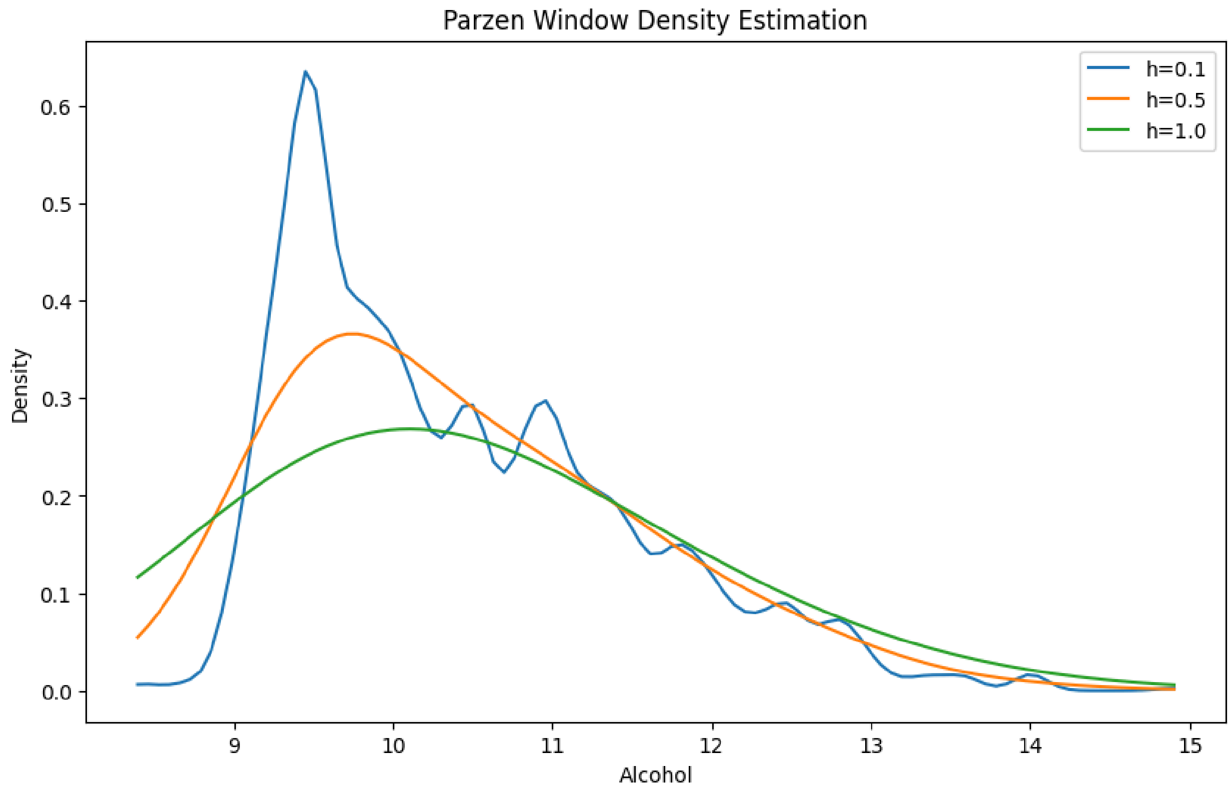
Assignment 2: Density Estimation
Dataset: Wine Quality Dataset (UCI Machine Learning Repository)
1. Implement the Parzen Window method to estimate the density for the "alcohol" feature. Experiment with different window sizes (h).

```
from scipy.stats import gaussian_kde

# Load Wine Quality dataset
wine_data =
pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/win
e-quality/winequality-red.csv', sep=';')
alcohol = wine_data['alcohol']

# Parzen Window method with different h values
plt.figure(figsize=(10, 6))
for h in [0.1, 0.5, 1.0]:
    kde = gaussian_kde(alcohol, bw_method=h)
    x = np.linspace(min(alcohol), max(alcohol), 100)
    plt.plot(x, kde(x), label=f'h={h}')
plt.title('Parzen Window Density Estimation')
plt.xlabel('Alcohol')
plt.ylabel('Density')
plt.legend()
plt.show()
```

Output:

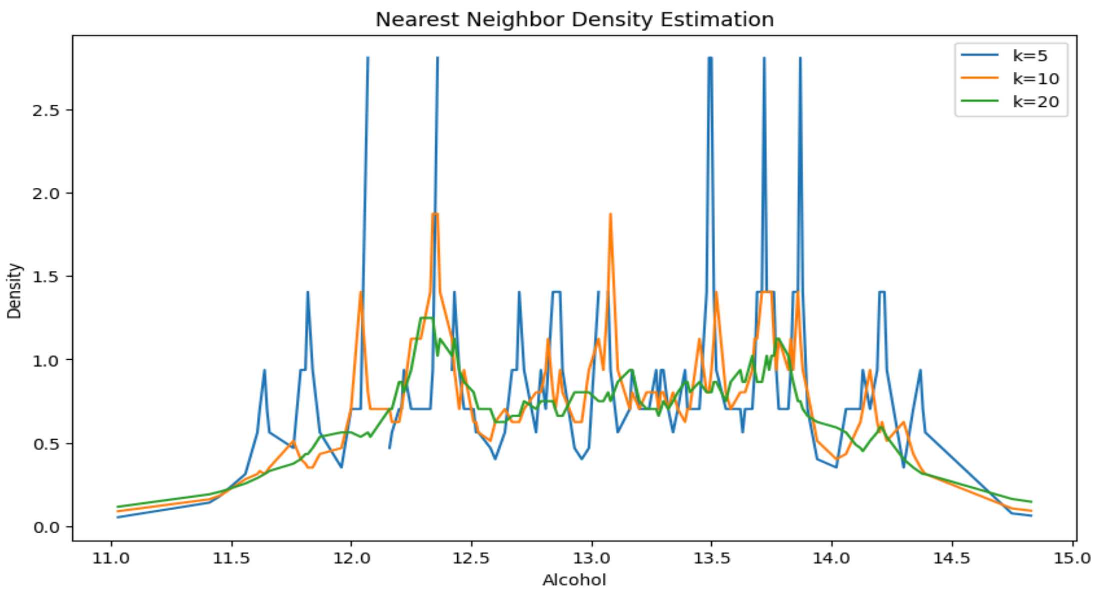


2. Use the Nearest Neighbor density estimation method with k = 5, 10, and 20.

```
from sklearn.neighbors import NearestNeighbors
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Example dataset (replace with Wine Quality Dataset)
from sklearn.datasets import load_wine
wine_data = load_wine(as_frame=True)
alcohol = wine_data.data["alcohol"]
alcohol = alcohol.to_numpy()
# Nearest Neighbor with k = 5, 10, 20
k_values = [5, 10, 20]
plt.figure(figsize=(10, 6))
for k in k_values:
    nbrs = NearestNeighbors(n_neighbors=k).fit(alcohol.reshape(-1, 1))   #
Reshape for compatibility
    distances, _ = nbrs.kneighbors(alcohol.reshape(-1, 1))
    densities = k / (len(alcohol) * distances[:, -1])  # Compute densities
    plt.plot(np.sort(alcohol), densities[np.argsort(alcohol)],
label=f'k={k}')
# Plot settings
plt.title('Nearest Neighbor Density Estimation')
plt.xlabel('Alcohol')
plt.ylabel('Density')
plt.legend()
plt.show()
```

Output:

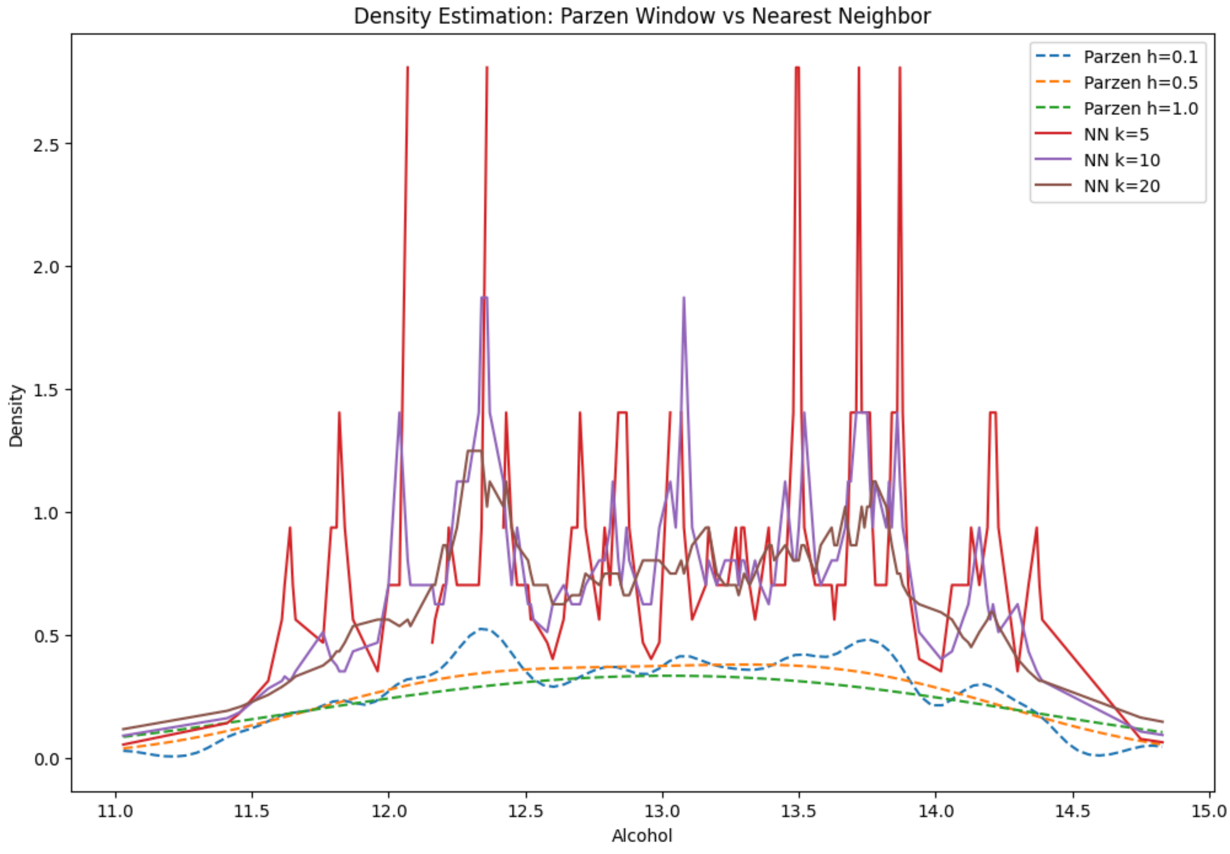


3. Visualize the density curves for both methods.

```
# Combine both Parzen Window and Nearest Neighbor on a single plot
plt.figure(figsize=(12, 8))

# Parzen Window
for h in [0.1, 0.5, 1.0]:
    kde = gaussian_kde(alcohol, bw_method=h)
    x = np.linspace(min(alcohol), max(alcohol), 100)
    plt.plot(x, kde(x), linestyle='dashed', label=f'Parzen h={h}')
# Nearest Neighbor
for k in [5, 10, 20]:
    nbrs = NearestNeighbors(n_neighbors=k).fit(alcohol[:, None])
    distances, _ = nbrs.kneighbors(alcohol[:, None])
    densities = k / (len(alcohol) * distances[:, -1])
    plt.plot(np.sort(alcohol), densities[np.argsort(alcohol)], label=f'NN
k={k}')
plt.title('Density Estimation: Parzen Window vs Nearest Neighbor')
plt.xlabel('Alcohol')
plt.ylabel('Density')
plt.legend()
plt.show()
```

Output:

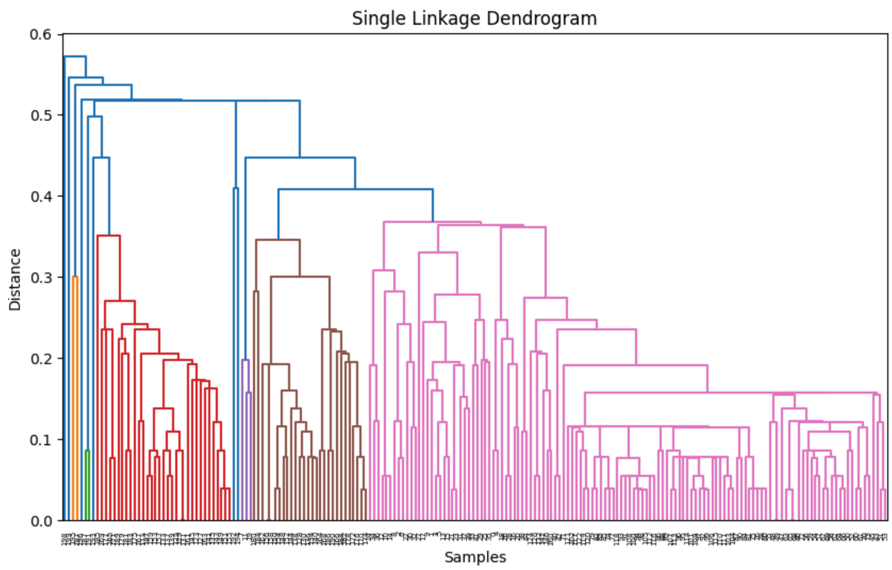


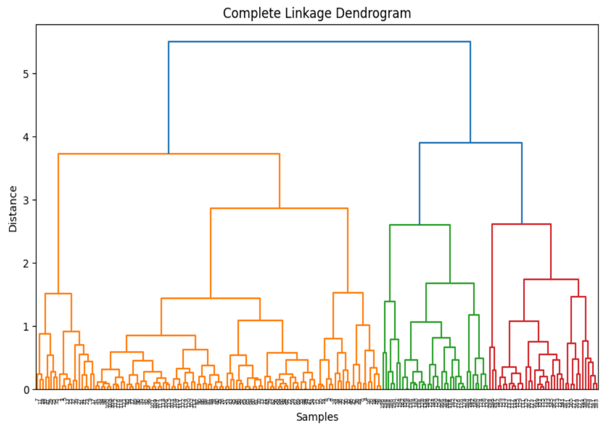
Assignment 3: Hierarchical Clustering
Dataset: Mall Customers Dataset (Kaggle)
1. Perform Agglomerative Hierarchical Clustering on "Annual Income" and "Spending Score".
○ Use different linkage methods (single, complete, average).
○ Visualize the dendrogram and identify clusters.

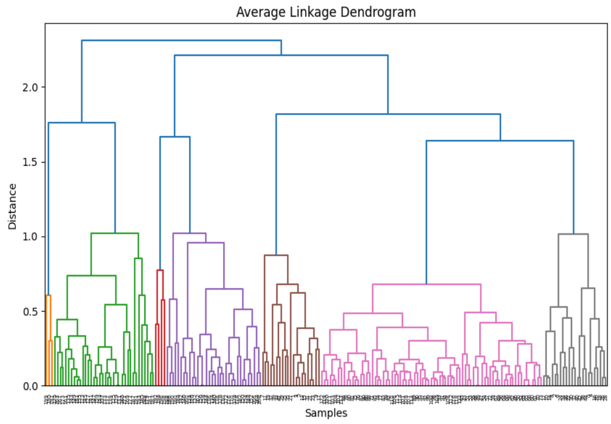
```
import kagglehub
import pandas as pd
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Load the dataset from KaggleHub
path = kagglehub.dataset_download("akram24/mall-customers")
mall_data = pd.read_csv(path + "/Mall_Customers.csv")
# Selecting relevant features
X = mall_data[['Annual Income (k$)', 'Spending Score (1-100)']]
# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Perform hierarchical clustering and visualize dendrograms
linkage_methods = ['single', 'complete', 'average']
for method in linkage_methods:
    plt.figure(figsize=(10, 6))
    Z = linkage(X_scaled, method=method)
    dendrogram(Z)
    plt.title(f'{method.capitalize()} Linkage Dendrogram')
    plt.xlabel('Samples')
    plt.ylabel('Distance')
    plt.show()
```

Output:







2. Implement Divisive Clustering with the ISODATA approach.
○ Experiment with different thresholds for cluster merging/splitting.

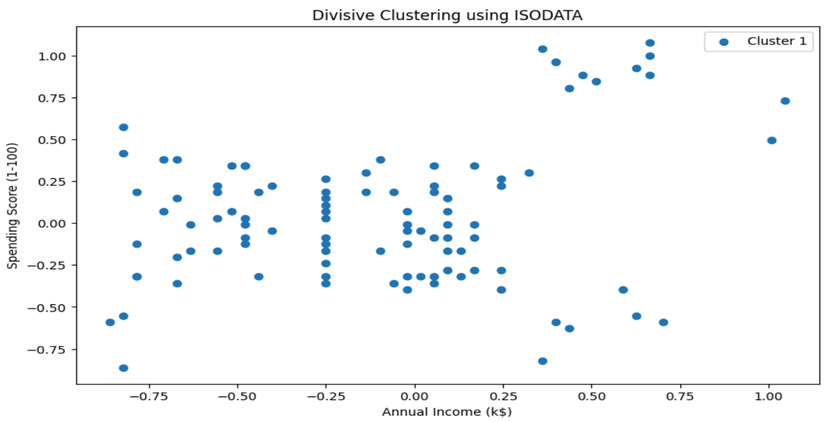
```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import pairwise_distances
import matplotlib.pyplot as plt
# Load the dataset
path = kagglehub.dataset_download("akram24/mall-customers")
mall_data = pd.read_csv(path + "/Mall_Customers.csv")
X = mall_data[['Annual Income (k$)', 'Spending Score (1-100)']]
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
def intra_cluster_variance(cluster, data):
    return np.sum(pairwise_distances(data[cluster], data[cluster])**2)
# Function for ISODATA algorithm
def isodata(X, min_cluster_size=5, max_iter=100, split_threshold=0.5,
merge_threshold=0.5):
    n_samples = X.shape[0]
    # Initially, put all points into one cluster
    clusters = [np.arange(n_samples)]
    cluster_centers = [np.mean(X, axis=0)]
    for iteration in range(max_iter):
        new_clusters = []
        for cluster in clusters:
            variance = intra_cluster_variance(cluster, X)
            if len(cluster) > min_cluster_size and variance >
split_threshold:
                center = np.mean(X[cluster], axis=0)
                distances = np.linalg.norm(X[cluster] - center, axis=1)
                median_distance = np.median(distances)

                cluster1 = cluster[distances <= median_distance]
                cluster2 = cluster[distances > median_distance]
```

```
                new_clusters.append(cluster1)
                new_clusters.append(cluster2)
            else:
                new_clusters.append(cluster)
        # Merging step (if clusters become too small, merge them)
        if len(new_clusters) > 1:
            merged_clusters = []
            for i in range(len(new_clusters) - 1):
                if np.mean(pairwise_distances(X[new_clusters[i]],
X[new_clusters[i+1]])) < merge_threshold:

merged_clusters.append(np.concatenate([new_clusters[i],
new_clusters[i+1]]))
                else:
                    merged_clusters.append(new_clusters[i])
            new_clusters = merged_clusters
        clusters = new_clusters
        if len(clusters) == len(cluster_centers):
            break
    return clusters
# Run ISODATA with different thresholds
clusters = isodata(X_scaled, split_threshold=0.5, merge_threshold=0.5)
# Visualize the clustering results
plt.figure(figsize=(10, 6))
for cluster in clusters:
    plt.scatter(X_scaled[cluster, 0], X_scaled[cluster, 1],
label=f'Cluster {clusters.index(cluster)+1}')
plt.title('Divisive Clustering using ISODATA')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```

Output:



Assignment 4: Fuzzy C-Means Clustering
Dataset: Human Activity Recognition Dataset (UCI)
1. Implement Fuzzy C-Means Clustering to group activity types based on accelerometer data.

```
import pandas as pd
import numpy as np
import skfuzzy as fuzz
from sklearn.preprocessing import StandardScaler

# Load the dataset
X_train = pd.read_csv('X_train.txt', sep='\s+', header=None)
y_train = pd.read_csv('y_train.txt', sep='\s+', header=None)
X = X_train.values
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Implement Fuzzy C-Means with 3 clusters
n_clusters = 3
cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(X_scaled.T, n_clusters,
2, error=0.005, maxiter=1000)
# Print the cluster centers in a readable format
print("Cluster Centers (first 5 features of each center):")
print(cntr[:, :5])
```

Output:

```
Cluster Centers (first 5 features of each center):
[[-0.01548044 -0.03631575 -0.01584798  0.80995411  0.83787358]
 [-0.0159207  -0.03628624 -0.01589484  0.80724991  0.83571476]
 [ 0.02920366  0.02611549  0.01048983 -0.79623855 -0.83913963]]
```

2. Experiment with different numbers of clusters (e.g., 3, 5, 7).

```
import pandas as pd
import numpy as np
import skfuzzy as fuzz
from sklearn.preprocessing import StandardScaler
import warnings

# Suppress FutureWarnings globally
warnings.filterwarnings("ignore", category=FutureWarning)
# Load the dataset with proper separator to avoid warnings
X_train = pd.read_csv('X_train.txt', sep='\s+', header=None)
y_train = pd.read_csv('y_train.txt', sep='\s+', header=None)X =
X_train.values
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Loop through different numbers of clusters (3, 5, 7)
```

```python
for n_clusters in [3, 5, 7]:
    cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(X_scaled.T,
n_clusters, 2, error=0.005, maxiter=1000)
    # Print the cluster centers for each n_clusters
    print(f"\nCluster Centers for {n_clusters} clusters (first 5 features
of each center):")
    print(cntr[:, :5])
```

Output:

```
Cluster Centers for 3 clusters (first 5 features of each center):
[[ 0.02920366  0.02611549  0.01048983 -0.79623855 -0.83913963]
 [-0.01547943 -0.03631582 -0.01584788  0.80996031  0.83787854]
 [-0.01592171 -0.03628617 -0.01589495  0.8072437   0.8357098 ]]

Cluster Centers for 5 clusters (first 5 features of each center):
[[ 0.01334472  0.02801087  0.01138123 -0.64471674 -0.6648726 ]
 [ 0.01329689  0.02803104  0.01137207 -0.64413113 -0.66413695]
 [ 0.00819613 -0.04906536 -0.02146874  1.00942766  1.0162944 ]
 [ 0.01336114  0.02800392  0.01138439 -0.64491785 -0.66512532]
 [ 0.00822851 -0.04906082 -0.02146334  1.00955483  1.0163839 ]]

Cluster Centers for 7 clusters (first 5 features of each center):
[[-0.00951237 -0.04335399 -0.01920433  0.9051567   0.92793799]
 [-0.00981754 -0.04337477 -0.01925232  0.90371959  0.92686793]
 [ 0.02487227  0.02833853  0.01118738 -0.76212997 -0.79741323]
 [-0.00932641 -0.04334128 -0.01917518  0.90603147  0.92858925]
 [-0.00927106 -0.0433375  -0.01916651  0.90629173  0.92878299]
 [ 0.02487209  0.02833938  0.01118686 -0.76212427 -0.79740368]
 [ 0.02487226  0.02833858  0.01118735 -0.76212964 -0.79741267]]
```

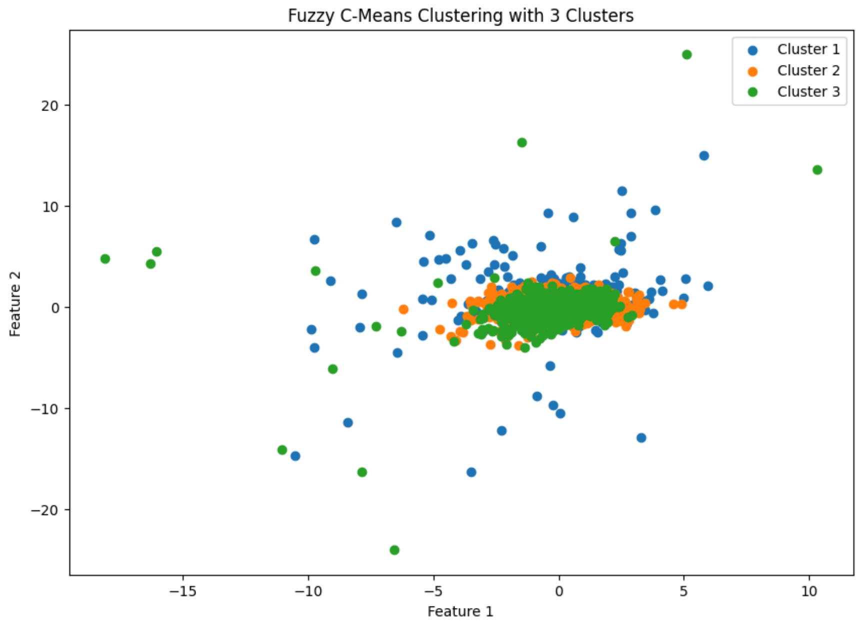
3. Visualize membership values for each data point using a scatter plot.

```python
import pandas as pd
import numpy as np
import skfuzzy as fuzz
from sklearn.preprocessing import StandardScaler
import warnings
import matplotlib.pyplot as plt

# Suppress FutureWarnings globally
warnings.filterwarnings("ignore", category=FutureWarning)
# Load the dataset with proper separator to avoid warnings
X_train = pd.read_csv('X_train.txt', sep='\s+', header=None)
y_train = pd.read_csv('y_train.txt', sep='\s+', header=None)
X = X_train.values
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Implement Fuzzy C-Means
```

```
n_clusters = 3
cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(X_scaled.T, n_clusters,
2, error=0.005, maxiter=1000)
cluster_membership = np.argmax(u, axis=0)
plt.figure(figsize=(10, 7))
for j in range(n_clusters):
    plt.scatter(X_scaled[cluster_membership == j, 0],
X_scaled[cluster_membership == j, 1], label=f'Cluster {j+1}')
plt.title(f'Fuzzy C-Means Clustering with {n_clusters} Clusters')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show()
```

Output:



4. Compare Fuzzy C-Means with traditional K-Means in terms of clustering accuracy and boundary overlap.

```
import pandas as pd
import numpy as np
import skfuzzy as fuzz
from sklearn.preprocessing import StandardScaler
import warnings
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt

# Suppress FutureWarnings globally
warnings.filterwarnings("ignore", category=FutureWarning)
# Load the dataset with proper separator to avoid warnings
```

```python
X_train = pd.read_csv('X_train.txt', sep='\s+', header=None)
y_train = pd.read_csv('y_train.txt', sep='\s+', header=None)
X = X_train.values
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Fuzzy C-Means clustering
n_clusters = 3
cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(X_scaled.T, n_clusters,
2, error=0.005, maxiter=1000)
fcm_labels = np.argmax(u, axis=0)
# Implement K-Means for comparison
kmeans = KMeans(n_clusters=n_clusters, random_state=42)
kmeans.fit(X_scaled)
kmeans_labels = kmeans.labels_
# Compare clustering accuracy using silhouette score
kmeans_silhouette = silhouette_score(X_scaled, kmeans_labels)
fcm_silhouette = silhouette_score(X_scaled, fcm_labels)
# Print the silhouette scores for comparison
print(f"K-Means Silhouette Score: {kmeans_silhouette:.3f}")
print(f"Fuzzy C-Means Silhouette Score: {fcm_silhouette:.3f}")
plt.figure(figsize=(10, 7))
# K-Means clustering results
plt.subplot(1, 2, 1)
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=kmeans_labels,
cmap='viridis')
plt.title("K-Means Clustering")
# Fuzzy C-Means clustering results
plt.subplot(1, 2, 2)
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=fcm_labels, cmap='viridis')
plt.title("Fuzzy C-Means Clustering")
plt.show()
```

Output: