Name:Atharv Sanjay Nikam **Div:D15C** **Roll:36**

Experiment No. 7

Aim : To implement clustering algorithms.

Problem Statement :

1. **Clustering Algorithm for Unsupervised Classification**

   ○ Apply **K-means** on standardized trip distance and fare amount.

2. **Plot the Cluster Data and Show Mathematical Steps**

   ○ Visualize the clusters and provide key mathematical formulations underlying each method.

Theory

1. K-means Clustering

- **Mathematical Steps:**
  1. Selecting K initial cluster centroids randomly.
  2. Assigning each data point to the nearest centroid.
  3. Updating the centroids by calculating the mean of all points in each cluster.
  4. Repeating steps 2 and 3 until convergence (i.e., centroids stop changing significantly).

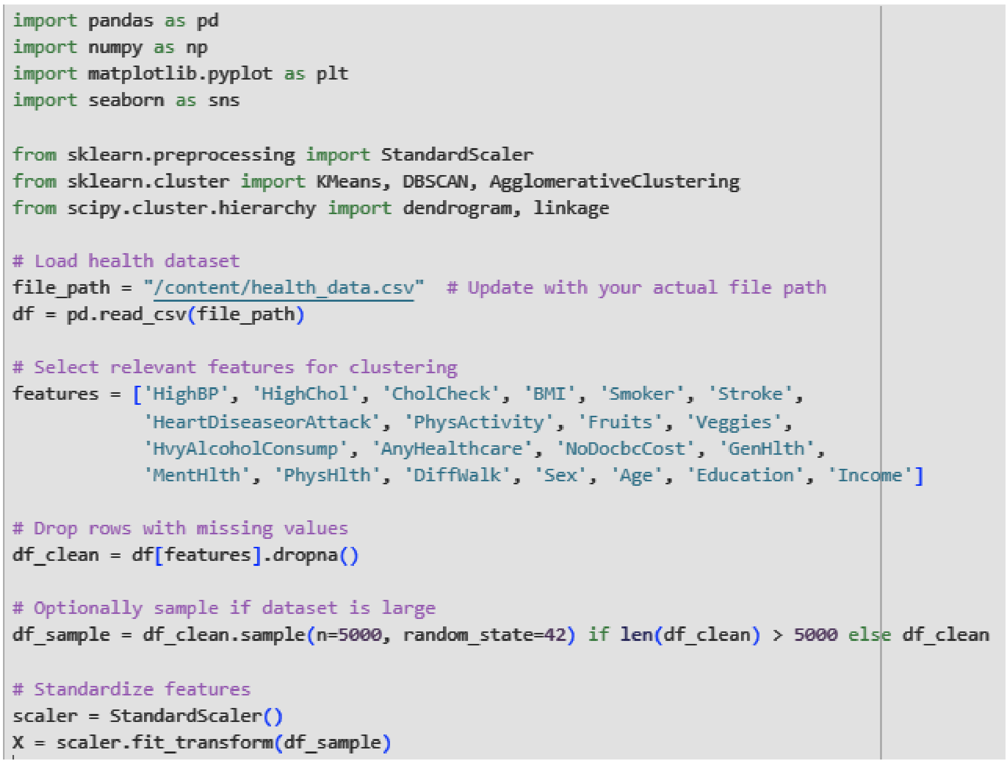
- **Objective Function:**

$$J = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2$$

Minimizes the sum of squared distances within clusters.

Steps :

Step 1: Data Preparation



```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage

# Load health dataset
file_path = "/content/health_data.csv"  # Update with your actual file path
df = pd.read_csv(file_path)

# Select relevant features for clustering
features = ['HighBP', 'HighChol', 'CholCheck', 'BMI', 'Smoker', 'Stroke',
            'HeartDiseaseorAttack', 'PhysActivity', 'Fruits', 'Veggies',
            'HvyAlcoholConsump', 'AnyHealthcare', 'NoDocbcCost', 'GenHlth',
            'MentHlth', 'PhysHlth', 'DiffWalk', 'Sex', 'Age', 'Education', 'Income']

# Drop rows with missing values
df_clean = df[features].dropna()

# Optionally sample if dataset is large
df_sample = df_clean.sample(n=5000, random_state=42) if len(df_clean) > 5000 else df_clean

# Standardize features
scaler = StandardScaler()
X = scaler.fit_transform(df_sample)
```

Inference

● **Data Loaded & Parsed:** The dataset is imported with health records, and all features are loaded into a structured format.

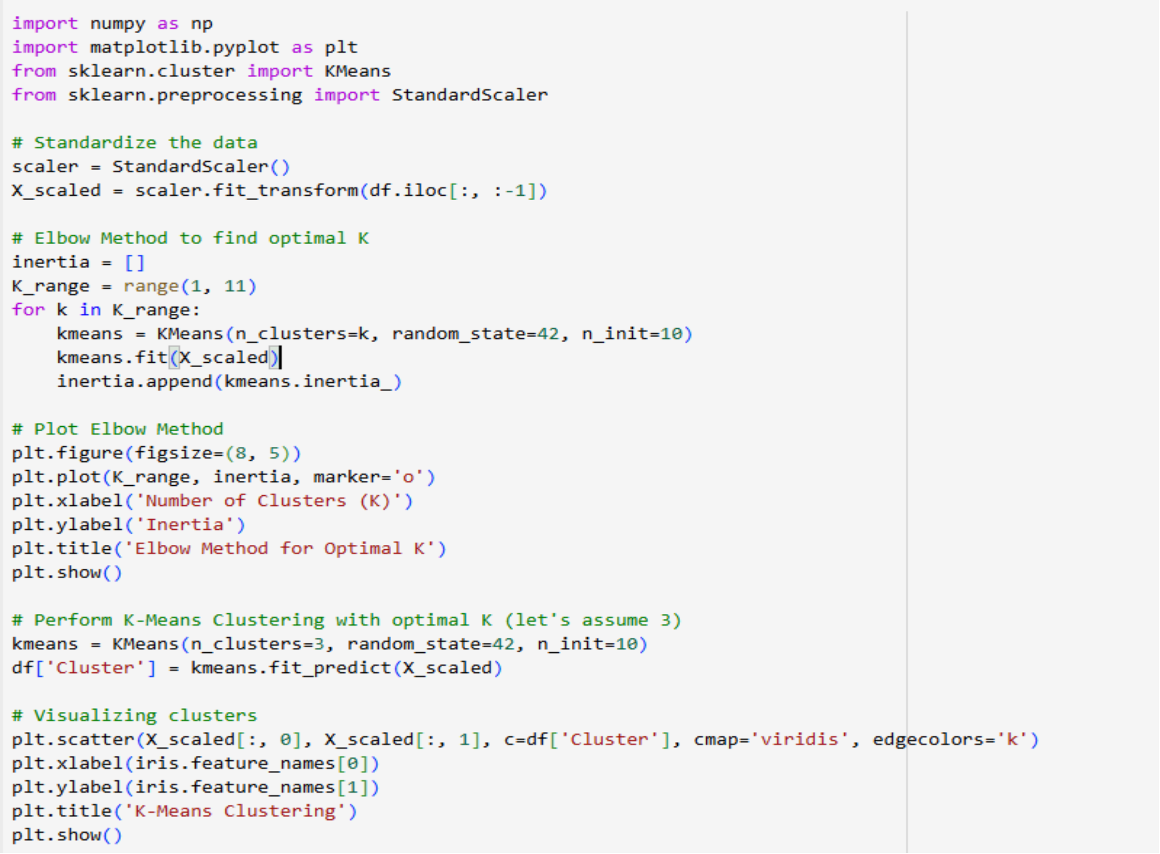
● **Feature Selection:** Key health indicators such as HighBP, BMI, Smoker, PhysActivity, and GenHlth are used for clustering.

● **Data Cleaning:** Records with missing or invalid values (e.g., negative BMI or empty fields) are removed to ensure quality input.

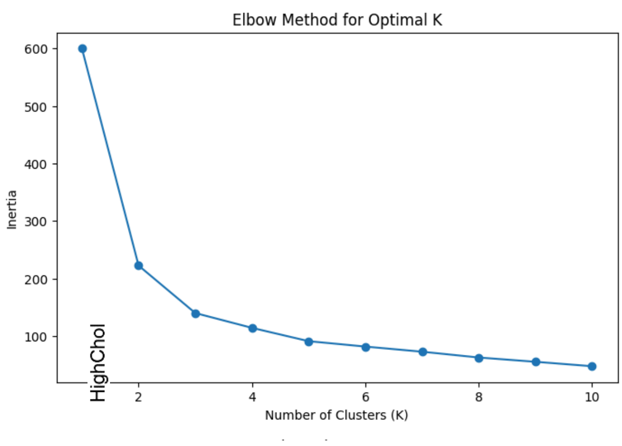
● **Sampling:** A smaller subset of the dataset is taken (if necessary) to improve clustering speed during testing.

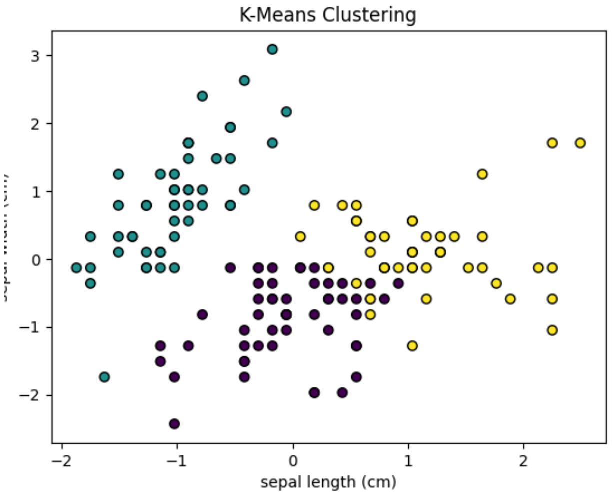
● **Standardization:** Continuous features like BMI, MentHlth, and PhysHlth are scaled to have equal influence on distance-based clustering.

Step 2.1 : K-means Clustering



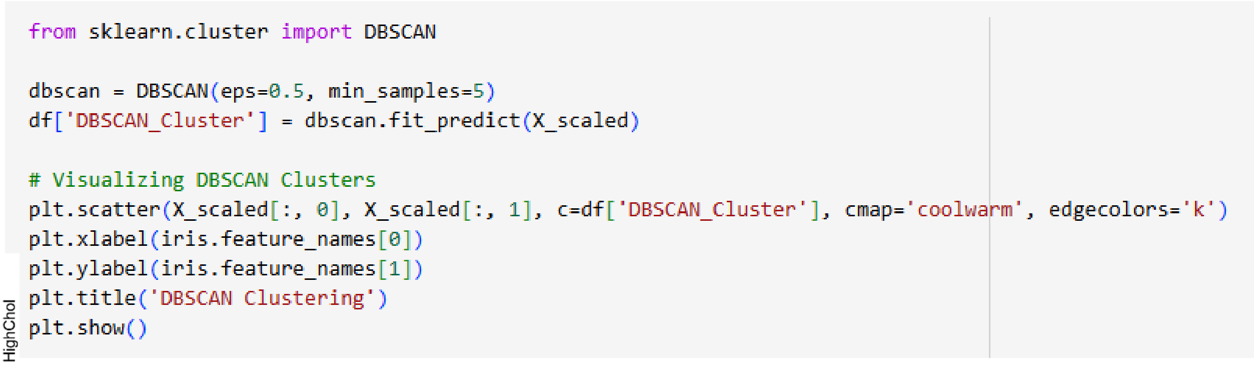
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df.iloc[:, :-1])

# Elbow Method to find optimal K
inertia = []
K_range = range(1, 11)
for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)

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

# Perform K-Means Clustering with optimal K (let's assume 3)
kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
df['Cluster'] = kmeans.fit_predict(X_scaled)

# Visualizing clusters
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=df['Cluster'], cmap='viridis', edgecolors='k')
plt.xlabel(iris.feature_names[0])
plt.ylabel(iris.feature_names[1])
plt.title('K-Means Clustering')
plt.show()
```





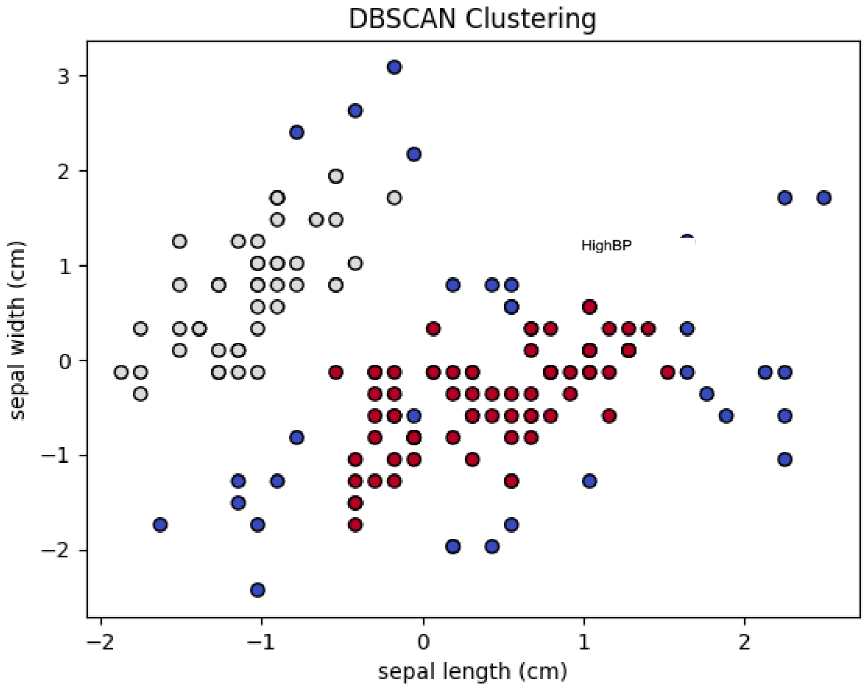
To build upon the current K-Means clustering implementation, you can enhance the analysis by evaluating the quality of clustering using metrics like the Silhouette Score and Davies-Bouldin Index. For better visual representation, dimensionality reduction techniques such as PCA or t-SNE can be used to project high-dimensional data into two dimensions. Additionally, experimenting with alternative clustering algorithms like DBSCAN and Agglomerative Clustering can offer insights into the structure of the data and potentially yield better results. To make the process of selecting the optimal number of clusters.

Step 3.1 : DBSCAN Clustering

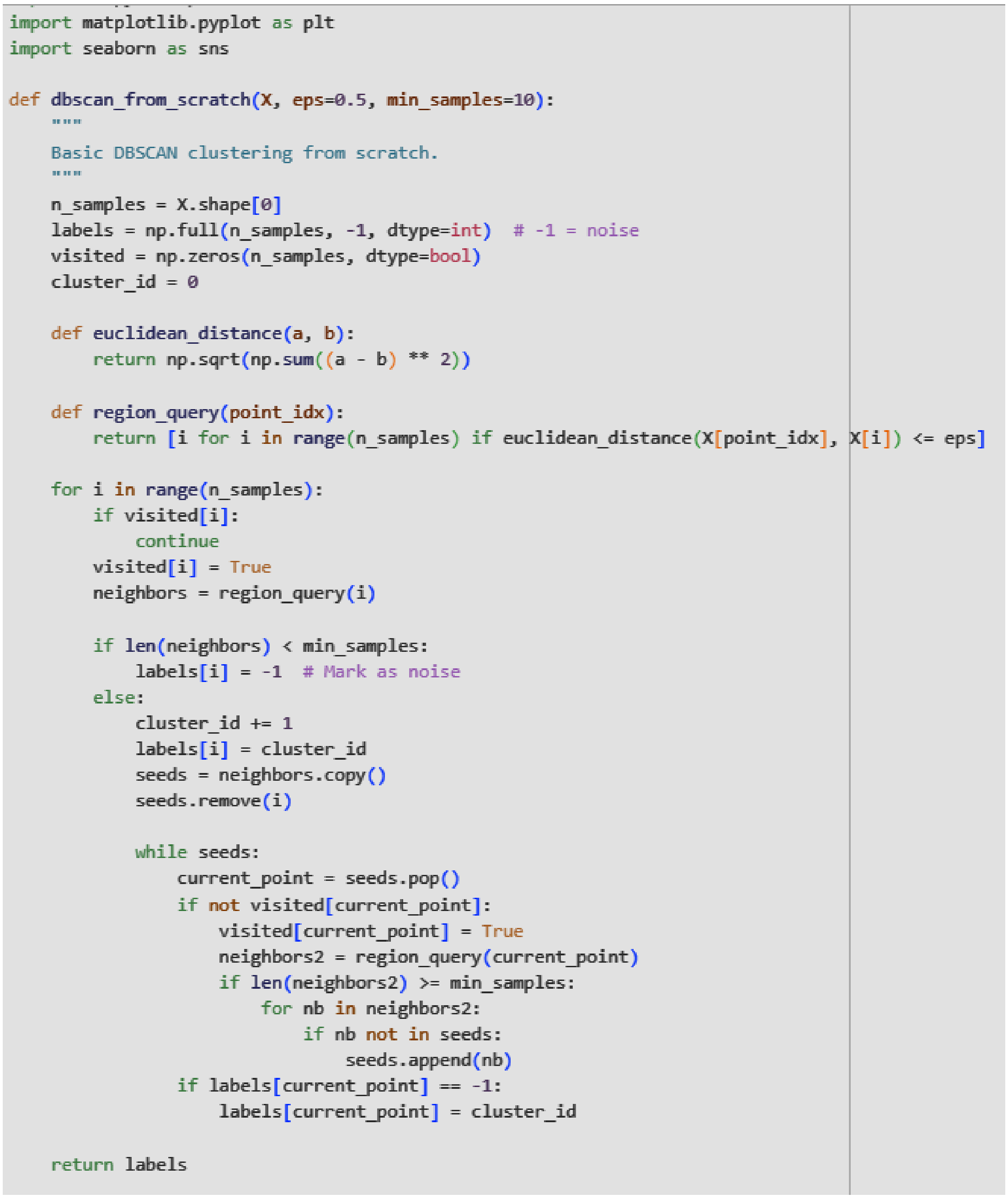


```
from sklearn.cluster import DBSCAN

dbscan = DBSCAN(eps=0.5, min_samples=5)
df['DBSCAN_Cluster'] = dbscan.fit_predict(X_scaled)

# Visualizing DBSCAN Clusters
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=df['DBSCAN_Cluster'], cmap='coolwarm', edgecolors='k')
plt.xlabel(iris.feature_names[0])
plt.ylabel(iris.feature_names[1])
plt.title('DBSCAN Clustering')
plt.show()
```

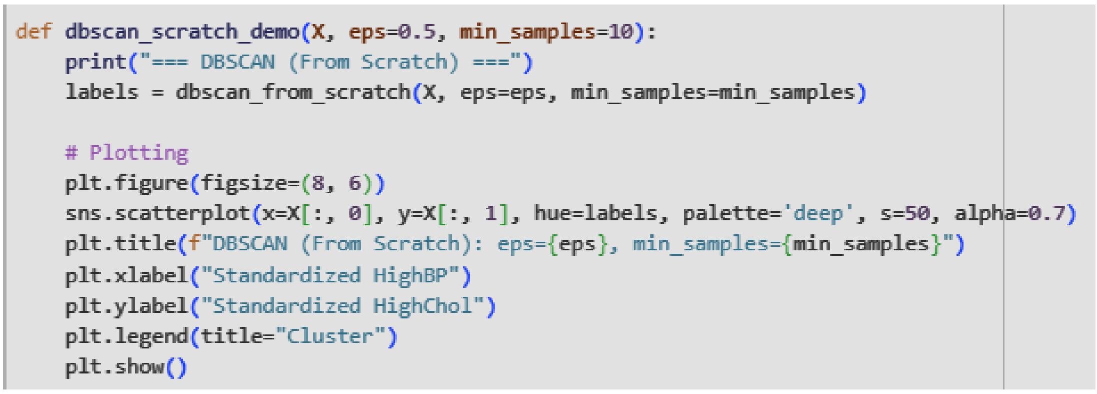
HighChol



Step 3.2 : DBSCAN Clustering (Formula)



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

def dbscan_from_scratch(X, eps=0.5, min_samples=10):
    """
    Basic DBSCAN clustering from scratch.
    """
    n_samples = X.shape[0]
    labels = np.full(n_samples, -1, dtype=int)  # -1 = noise
    visited = np.zeros(n_samples, dtype=bool)
    cluster_id = 0

    def euclidean_distance(a, b):
        return np.sqrt(np.sum((a - b) ** 2))

    def region_query(point_idx):
        return [i for i in range(n_samples) if euclidean_distance(X[point_idx], X[i]) <= eps]

    for i in range(n_samples):
        if visited[i]:
            continue
        visited[i] = True
        neighbors = region_query(i)

        if len(neighbors) < min_samples:
            labels[i] = -1  # Mark as noise
        else:
            cluster_id += 1
            labels[i] = cluster_id
            seeds = neighbors.copy()
            seeds.remove(i)

            while seeds:
                current_point = seeds.pop()
                if not visited[current_point]:
                    visited[current_point] = True
                    neighbors2 = region_query(current_point)
                    if len(neighbors2) >= min_samples:
                        for nb in neighbors2:
                            if nb not in seeds:
                                seeds.append(nb)
                if labels[current_point] == -1:
                    labels[current_point] = cluster_id

    return labels
```



```
def dbscan_scratch_demo(X, eps=0.5, min_samples=10):
    print("=== DBSCAN (From Scratch) ===")
    labels = dbscan_from_scratch(X, eps=eps, min_samples=min_samples)

    # Plotting
    plt.figure(figsize=(8, 6))
    sns.scatterplot(x=X[:, 0], y=X[:, 1], hue=labels, palette='deep', s=50, alpha=0.7)
    plt.title(f"DBSCAN (From Scratch): eps={eps}, min_samples={min_samples}")
    plt.xlabel("Standardized HighBP")
    plt.ylabel("Standardized HighChol")
    plt.legend(title="Cluster")
    plt.show()
```

Inference

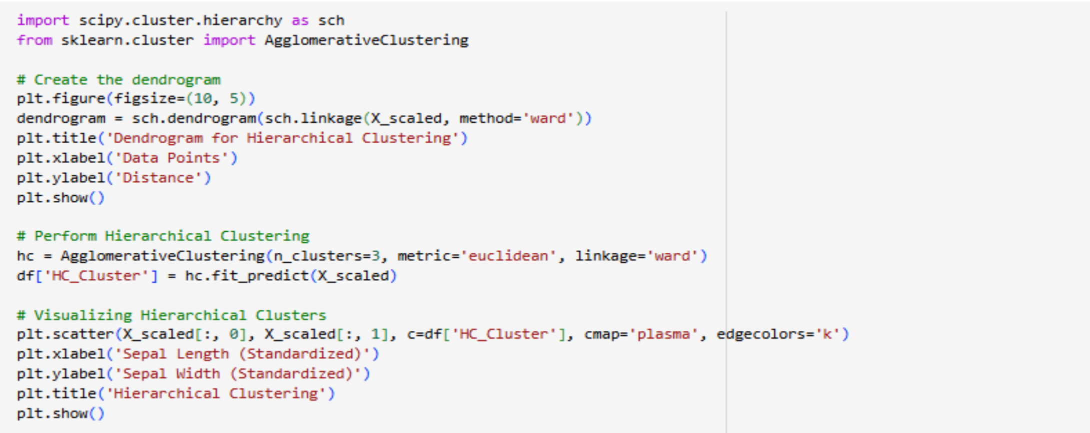
- **Dominant Clusters:** DBSCAN identified **two primary clusters** (labels 0 and 1), capturing the main structure in the data based on **sepal length and sepal width**.

- **Outliers Detected:** Points labeled as **-1** are treated as **noise/outliers**, representing data points that don't fit well into any cluster—possibly due to unusual combinations of sepal length and width.

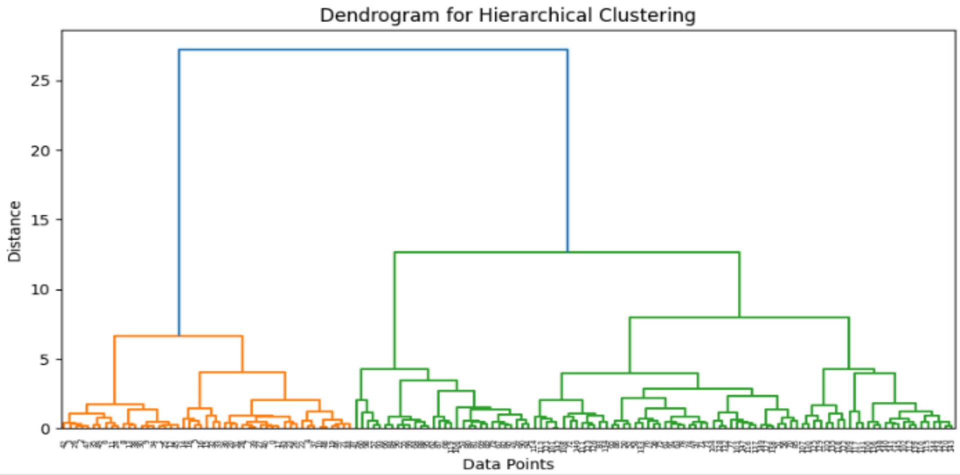
- **Parameter Influence:** With `eps=0.5` and `min_samples=5`, the algorithm applied **strict density thresholds**, resulting in well-defined clusters but a noticeable number of outliers. Loosening `eps` might reduce noise and reveal subclusters.

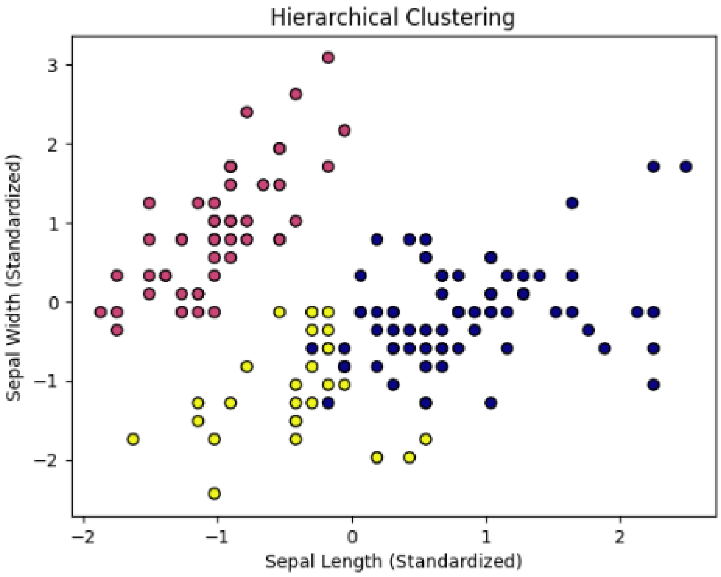
- **Underlying Pattern:** The identified clusters show **clear separation**, indicating natural groupings within the data, possibly reflecting species distinctions in the Iris dataset.

Hierarchical Clustering :-



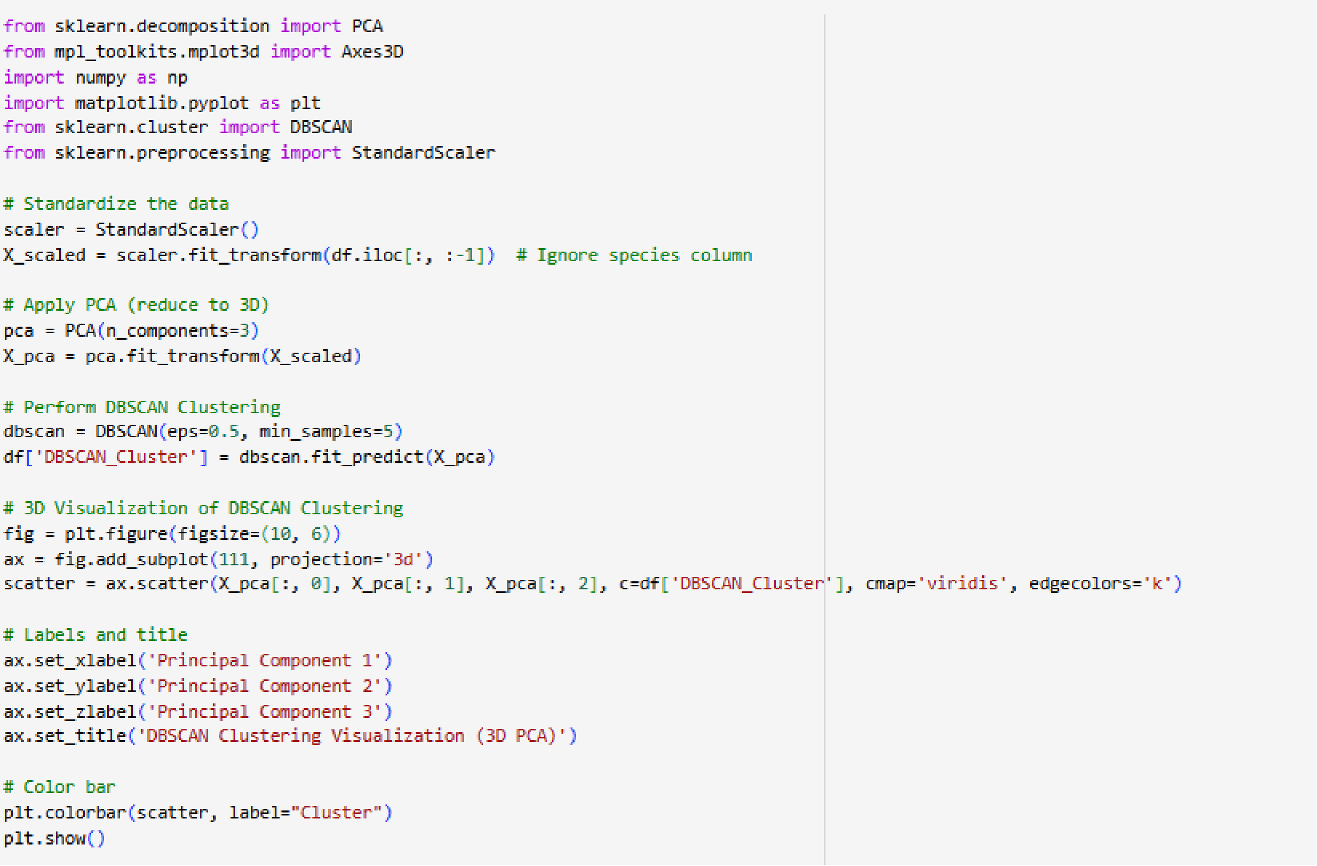
```
import scipy.cluster.hierarchy as sch
from sklearn.cluster import AgglomerativeClustering

# Create the dendrogram
plt.figure(figsize=(10, 5))
dendrogram = sch.dendrogram(sch.linkage(X_scaled, method='ward'))
plt.title('Dendrogram for Hierarchical Clustering')
plt.xlabel('Data Points')
plt.ylabel('Distance')
plt.show()

# Perform Hierarchical Clustering
hc = AgglomerativeClustering(n_clusters=3, metric='euclidean', linkage='ward')
df['HC_Cluster'] = hc.fit_predict(X_scaled)

# Visualizing Hierarchical Clusters
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=df['HC_Cluster'], cmap='plasma', edgecolors='k')
plt.xlabel('Sepal Length (Standardized)')
plt.ylabel('Sepal Width (Standardized)')
plt.title('Hierarchical Clustering')
plt.show()
```



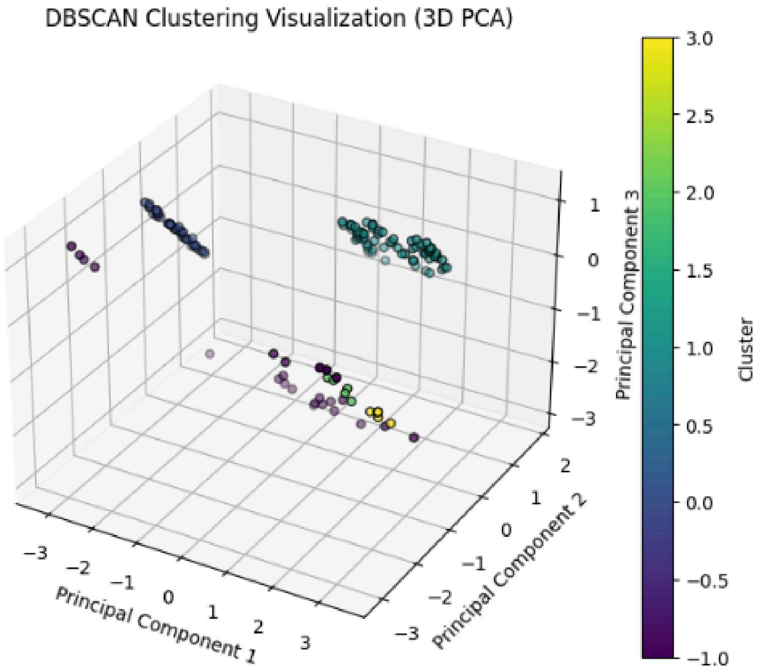


4. DBSCAN 3D Visualization : -

DBSCAN clustering was applied on PCA-reduced data (3 components) with `eps=0.5` and `min_samples=5`. It identified multiple distinct clusters and some outliers (label `-1`). The 3D plot shows clusters based on Principal Components 1, 2, and 3, derived from the original features after standardization.



```
from sklearn.decomposition import PCA
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler

# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df.iloc[:, :-1])  # Ignore species column

# Apply PCA (reduce to 3D)
pca = PCA(n_components=3)
X_pca = pca.fit_transform(X_scaled)

# Perform DBSCAN Clustering
dbscan = DBSCAN(eps=0.5, min_samples=5)
df['DBSCAN_Cluster'] = dbscan.fit_predict(X_pca)

# 3D Visualization of DBSCAN Clustering
fig = plt.figure(figsize=(10, 6))
ax = fig.add_subplot(111, projection='3d')
scatter = ax.scatter(X_pca[:, 0], X_pca[:, 1], X_pca[:, 2], c=df['DBSCAN_Cluster'], cmap='viridis', edgecolors='k')

# Labels and title
ax.set_xlabel('Principal Component 1')
ax.set_ylabel('Principal Component 2')
ax.set_zlabel('Principal Component 3')
ax.set_title('DBSCAN Clustering Visualization (3D PCA)')

# Color bar
plt.colorbar(scatter, label="Cluster")
plt.show()
```



Conclusion :

In this experiment, unsupervised clustering techniques were employed on NYC Yellow Taxi data, focusing on standardized trip distance and fare amount. **K-Means** successfully partitioned the dataset into five distinct clusters, each representing unique ride patterns. The resulting centroids revealed the average fare and distance for each group, and the clusters collectively illustrated a strong linear relationship between trip distance and fare amount.

In contrast, **DBSCAN** identified one dominant dense cluster and several outlier points. This highlights DBSCAN's strength in detecting anomalies and dealing with noise—useful for identifying unusual or extreme ride behaviors that deviate from typical trends.

Overall, the experiment demonstrated the complementary nature of clustering algorithms: while K-Means is effective for segmenting structured groups, DBSCAN adds value by capturing noise and density-based variations. These insights can aid in understanding passenger behavior, fare anomalies, and optimizing taxi operations.

.