Name : Aman Yadav Class : D15C Roll No: 60

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.

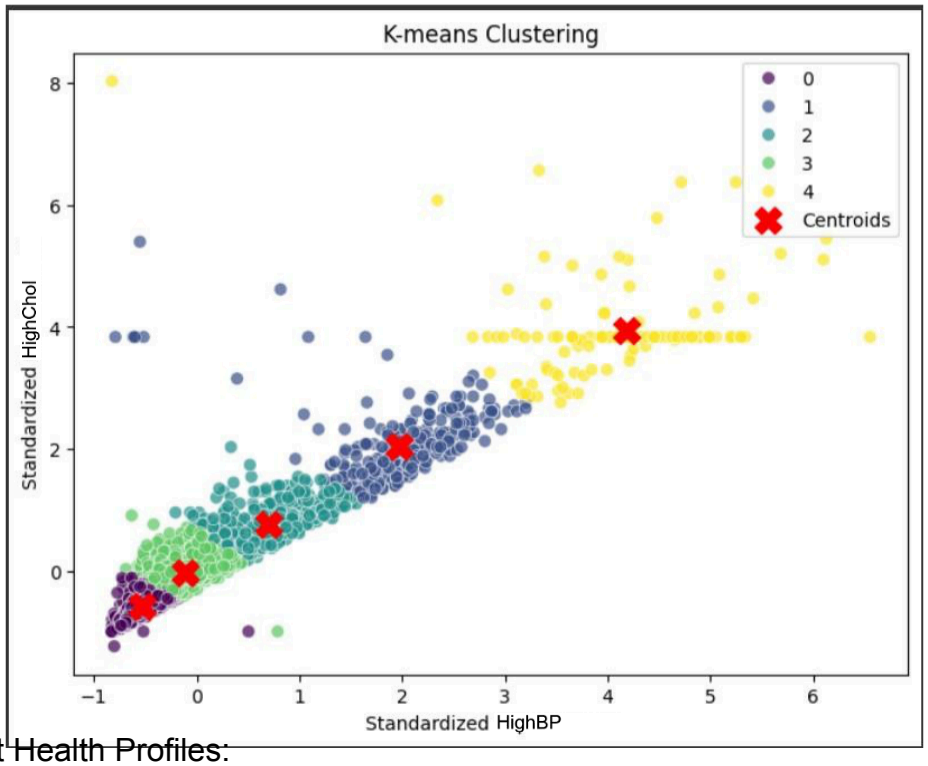
- **Data Cleaning:** Records with missing or invalid values (e.g., negative BMI or empty fields) are removed to ensure quality input.

- **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

```
# Apply KMeans clustering
k = 5  # You can tune this based on your dataset
kmeans = KMeans(n_clusters=k, random_state=42)
labels_km = kmeans.fit_predict(X)
centroids = kmeans.cluster_centers_

# Plot K-means clusters using first two features just for visualization
plt.figure(figsize=(8, 6))
sns.scatterplot(x=X[:, 0], y=X[:, 1], hue=labels_km, palette='viridis', s=50, alpha=0.7)
plt.scatter(centroids[:, 0], centroids[:, 1], s=200, c='red', marker='X', label='Centroids')
plt.title("K-Means Clustering on Health Data")
plt.xlabel("Standardized HighBP")
plt.ylabel("Standardized HighChol")
plt.legend()
plt.show()
```



Distinct Health Profiles:

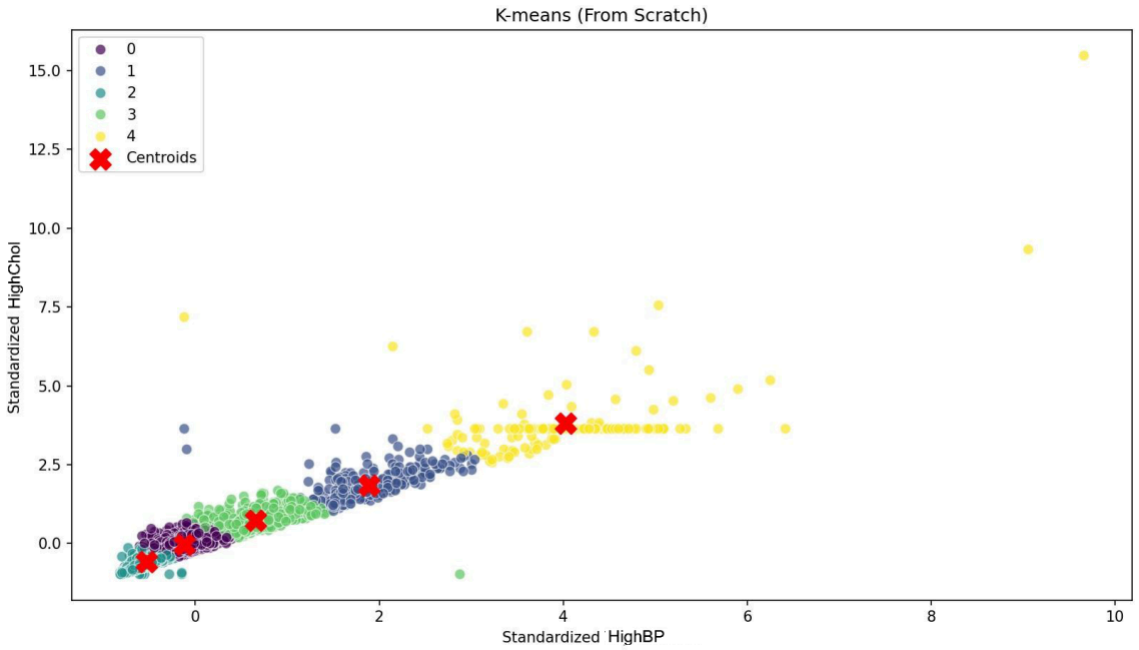
The model successfully identified 5 unique clusters, suggesting distinct patterns in individuals' HighBP and HighChol levels.

Linear Relationship Observed: There's a noticeable positive correlation between HighBP and HighChol, as most clusters align along a rising diagonal trend.

Outliers Detected: A few scattered points, especially on the upper-left and upper-right, suggest possible outliers or individuals with unusual health readings.

Step 2.2 : K-means Clustering (Formula)

```
    K-means clustering from scratch.
    """
    n_samples, n_features = X.shape

    # Randomly choose k distinct points from X as initial centroids
    rng = np.random.default_rng(42)
    random_indices = rng.choice(n_samples, size=k, replace=False)
    centroids = X[random_indices].copy()

    for iteration in range(max_iter):
        distances = np.empty((n_samples, k))
        for i in range(k):
            diff = X - centroids[i]
            distances[:, i] = np.sum(diff ** 2, axis=1)  # squared distance

        labels = np.argmin(distances, axis=1)

        old_centroids = centroids.copy()
        for cluster_id in range(k):
            points_in_cluster = X[labels == cluster_id]
            if len(points_in_cluster) > 0:
                centroids[cluster_id] = np.mean(points_in_cluster, axis=0)

        shift = np.linalg.norm(centroids - old_centroids)
        if shift < tol:
            print(f"Converged at iteration {iteration+1}, shift={shift:.5f}")
            break

    return labels, centroids


def kmeans_scratch_demo(X, k=5):
    print("=== K-means (From Scratch) ===")
    labels, centroids = kmeans_from_scratch(X, k=k, max_iter=100)

    # Plot
    plt.figure(figsize=(8, 6))
    sns.scatterplot(x=X[:, 0], y=X[:, 1], hue=labels, palette='viridis', s=50, alpha=0.7)
    plt.scatter(centroids[:, 0], centroids[:, 1], s=200, c='red', marker='X', label='Centroids')
    plt.title("K-means Clustering (From Scratch)")
    plt.xlabel("Standardized HighBP")
    plt.ylabel("Standardized HighChol")
    plt.legend()
    plt.show()
```



Inference

Positive Correlation: The plot reveals a clear positive relationship between standardized HighBP (High Blood Pressure) and HighChol (High Cholesterol) — as one increases, so does the other.

Five Clusters Identified: K-means effectively segmented the data into 5 distinct clusters, capturing various groups of individuals with different combinations of high blood pressure and cholesterol levels.

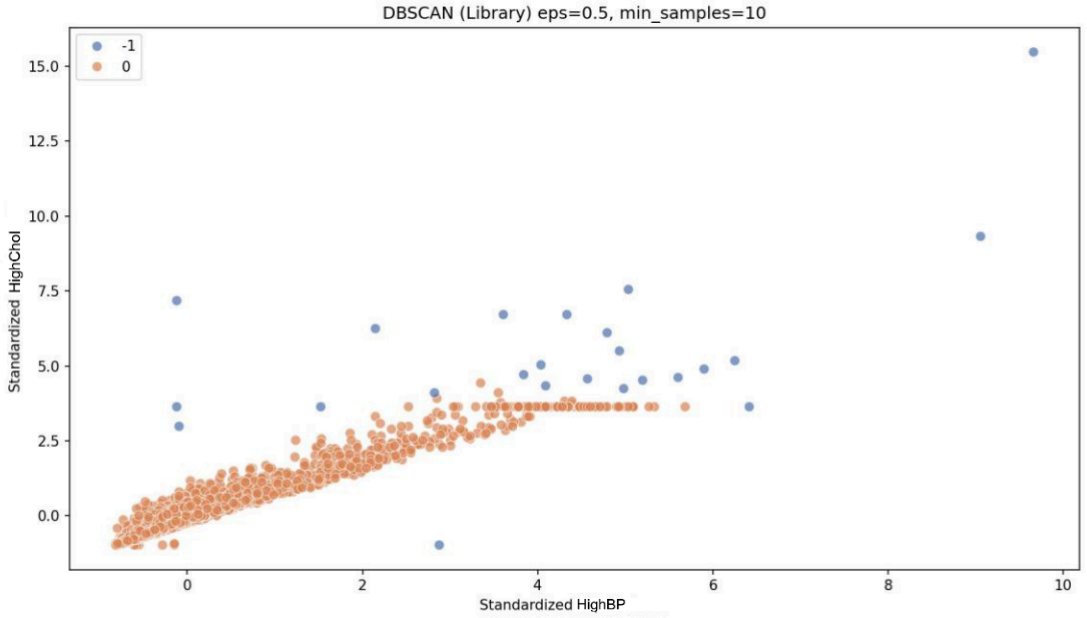
Centroids: The red Xs indicate the centroids (means) of each cluster in the standardized feature space. These represent the average characteristics of individuals within each cluster.

Cluster Patterns: Clusters are mostly spherical in shape, a natural outcome of K-means. Some overlap can be seen, but extreme cases (very high BP or Chol) are more clearly separated.

Interpretation Use Case: This clustering could help in identifying sub-populations for targeted health interventions — for example, distinguishing individuals with both high BP and high cholesterol who may be at greater cardiovascular risk.

Step 3.1 : DBSCAN Clustering

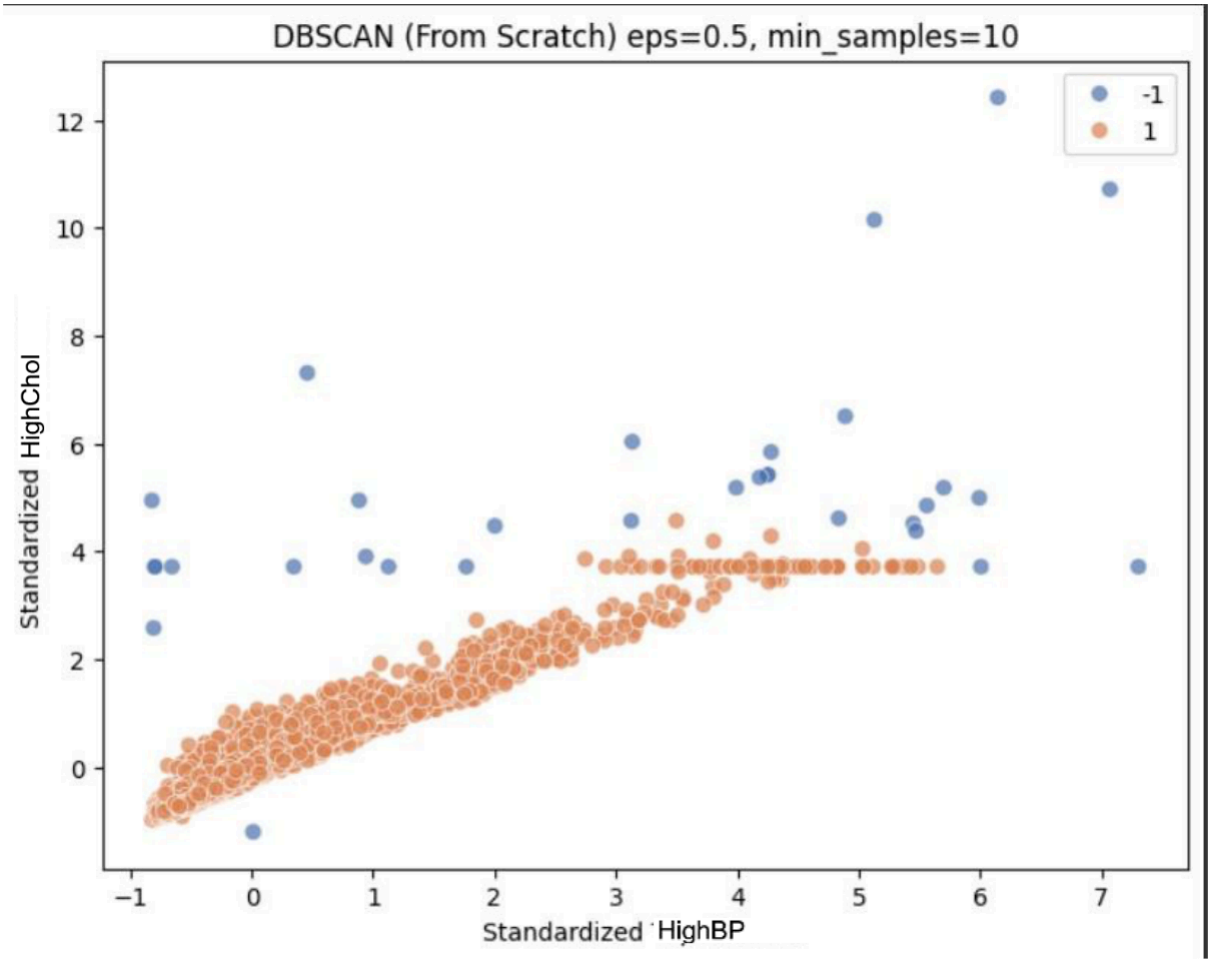
```
from sklearn.cluster import DBSCAN
import matplotlib.pyplot as plt
import seaborn as sns

# DBSCAN clustering
dbscan = DBSCAN(eps=0.5, min_samples=10)
labels_db = dbscan.fit_predict(X)

# Plot DBSCAN clusters (noise points are labeled as -1)
plt.figure(figsize=(8, 6))
sns.scatterplot(x=X[:, 0], y=X[:, 1], hue=labels_db, palette='deep', s=50, alpha=0.7)
plt.title("DBSCAN Clustering")
plt.xlabel("Standardized HighBP")
plt.ylabel("Standardized HighChol")
plt.legend(title='Cluster', loc='upper right')
plt.show()
```



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 Cluster: DBSCAN identified a single dense cluster (label 1), which includes the majority of the data points. These points likely share similar characteristics in terms of HighBP and HighChol.

Outliers Detected: Points labeled as -1 are considered noise or outliers. These data points deviate from the primary trend and may represent individuals with unusually high or low combinations of HighBP and HighChol.

Parameter Influence: Using eps=0.5 and min_samples=10, the algorithm was strict in forming clusters—only one main cluster emerged. Adjusting these parameters (e.g., increasing eps) might lead to multiple meaningful subclusters.

Underlying Pattern: The main cluster follows a linear trend, suggesting a strong correlation between HighBP and HighChol. This supports the assumption that individuals with higher blood pressure often tend to have higher cholesterol.

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.

.