Name: Krish Nisar
Class: TY03 B Batch
Roll No. : 41
Subject: Data Warehouse and Mining
Experiment 4: Implementation of Clustering Algorithm (K-means / Agglomerative) using Python

Procedure:

1. **Data Collection & Preprocessing**
    a. **Load the dataset using `pandas`.**
    b. **Handle missing values and scale numerical features using StandardScaler or MinMaxScaler.**
    c. **Choose relevant features for clustering.**

2. **Choosing the Number of Clusters (K)**
    a. **Use the Elbow Method (plot inertia vs. K values) to find the optimal number of clusters.**
    b. **Use the Silhouette Score to measure cluster quality.**

3. **Applying K-Means Algorithm**
    a. **Import `KMeans` from `sklearn.cluster`.**
    b. **Initialize the model with the chosen `n_clusters` and `random_state`.**
    c. **Fit the model using the `fit()` method on the dataset.**

4. **Assigning Clusters & Evaluating Results**
    a. **Use `predict()` to assign cluster labels to data points.**
    b. **Visualize clusters using scatter plots (if data is 2D or reduced using PCA).**

5. **Interpreting and Refining Clusters**
    a. **Analyze cluster characteristics to understand patterns.**
    b. **Fine-tune parameters or pre-process data again if needed.**

Program Codes: (lang: Java)

```
import java.util.*;

public class KMeans1D {

   public static void kMeans(int[] data, int k) {
```

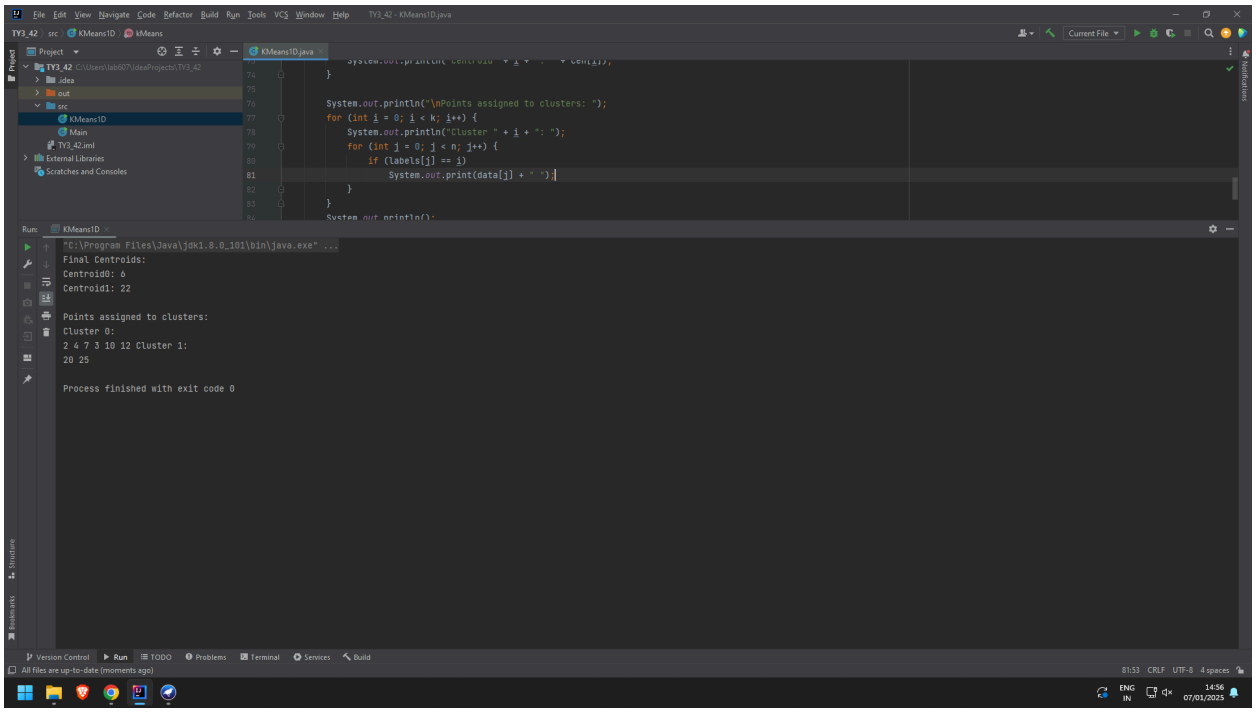
```
int n = data.length;
Random rand = new Random();
int[] cen = new int[k];

for (int i = 0; i < k; i++) {
   int newCent;
   boolean isDup;

   do {
      isDup = false;
      newCent = data[rand.nextInt(n)];
      for (int j = 0; j < i; j++) {
         if (cen[j] == newCent) {
            isDup = true;
            break;
         }
      }
   }while (isDup);
   cen[i] = newCent;
}

boolean converged = false;
int[] labels = new int[n];
int[] prevCent = new int[k];
```

```
while (!converged) {
    for (int i = 0; i < n; i++) {
        int minDist = Integer.MAX_VALUE;
        int closeCent = -1;
        for (int j = 0; j < k; j++) {
            int dist = Math.abs(data[i] - cen[j]);
            if (dist < minDist) {
                minDist = dist;
                closeCent = j;
            }
        }
        labels[i] = closeCent;
    }

    System.arraycopy(cen, 0, prevCent, 0, k);
    for (int i = 0; i < k; i++) {
        int sum = 0;
        int count = 0;
        for (int j = 0; j < n; j++) {
            if (labels[j] == i) {
                sum += data[j];
                count++;
            }
        }
```

```
        if (count > 0) {
            cen[i] = sum/count;
        }
    }

    converged = true;
    for (int i = 0; i < k; i++) {
        if (cen[i] != prevCent[i]) {
            converged = false;
            break;
        }
    }
}

System.out.println("Final Centroids: ");
for (int i = 0; i < k; i++) {
    System.out.println("Centroid" + i + ": " + cen[i]);
}

System.out.println("\nPoints assigned to clusters: ");
for (int i = 0; i < k; i++) {
    System.out.println("Cluster " + i + ": ");
    for (int j = 0; j < n; j++) {
        if (labels[j] == i)
            System.out.print(data[j] + " ");
```

```
            }
        }
        System.out.println();
    }

    public static void main(String[] a) {
        int[] data = {2, 4, 7, 3, 10, 12, 20, 25};
        int k = 2;
        kMeans(data, k);
    }
}
```

Output:



Conclusion: Implementing a Decision Tree Classifier in Python using `sklearn` is straightforward and effective for classification tasks. It provides high interpretability but may suffer from overfitting if not pruned properly. By tuning hyperparameters and performing feature selection, we can improve the model's accuracy and generalization for real-world applications.

Review Questions:

Q1) What is the K-means clustering algorithm, and how does it work?

Ans:

K-Means is an **unsupervised clustering algorithm** that partitions data into **K clusters** based on similarity.

It works iteratively as follows:

1. Select **K random centroids** (initial cluster centers).
2. Assign each data point to the **nearest centroid** (using a distance metric like Euclidean).
3. Compute the **new centroid** for each cluster by averaging the assigned points.
4. Repeat steps 2 & 3 until centroids **no longer change** or a stopping condition is met.

The final clusters contain data points with similar characteristics.

Q2) How do you determine the optimal number of clusters in K-means

Ans:

The **Elbow Method**:

1. Plot **inertia (sum of squared distances from centroids) vs. K**.
2. Choose the **K value** where the inertia graph bends like an **elbow** (optimal trade-off between accuracy & efficiency).

The **Silhouette Score**:

1. Measures how similar a data point is to its **own cluster vs. other clusters**.
2. Values range from **-1 to 1** (higher = better clustering).

The **Gap Statistic**:

1. Compares clustering performance with a **randomly generated dataset**.
2. The optimal K is where the gap is largest.

Q3) What are the common distance metrics used in Agglomerative Clustering?

Ans:

Agglomerative clustering is a hierarchical clustering approach that uses distance metrics to merge data points into clusters. Common metrics include:

1. **Euclidean Distance:**
    a. **The straight-line distance between two points.**
    b. **Commonly used in most clustering algorithms.**

2. **Manhattan Distance:**
    a. **Measures distance along axes at right angles (sum of absolute differences).**
    b. **Useful for grid-based clustering.**

3. **Cosine Similarity:**
    a. **Measures the angle between two vectors instead of direct distance.**
    b. **Common in text and document clustering.**

4. **Mahalanobis Distance:**
    a. **Accounts for correlations between variables and scales data accordingly.**
    b. **Used in high-dimensional datasets.**

5. **Correlation Distance:**
    a. **Measures the dissimilarity based on statistical correlation rather than absolute values.**
    b. **Suitable for time series and financial data.**