

## Experiment No 7

Aim: Implementation of Clustering algorithm (K-means/Kmedoids)

Theory:

K-Means Clustering is an Unsupervised Machine Learning algorithm, which groups the unlabeled dataset into different clusters.

K means Clustering

Unsupervised Machine Learning is the process of teaching a computer to use unlabeled, unclassified data and enabling the algorithm to operate on that data without supervision. Without any previous data training, the machine's job in this case is to organize unsorted data according to parallels, patterns, and variations.

The goal of clustering is to divide the population or set of data points into a number of groups so that the data points within each group are more comparable to one another and different from the data points within the other groups. It is essentially a grouping of things based on how similar and different they are to one another.

We are given a data set of items, with certain features, and values for these features (like a vector). The task is to categorize those items into groups. To achieve this, we will use the K-means algorithm; an unsupervised learning algorithm. 'K' in the name of the algorithm represents the number of groups/clusters we want to classify our items into.

(It will help if you think of items as points in an n-dimensional space). The algorithm will categorize the items into k groups or clusters of similarity. To calculate that similarity, we will use the Euclidean distance as a measurement.

The algorithm works as follows:

1. First, we randomly initialize k points, called means or cluster centroids.
2. We categorize each item to its closest mean and we update the mean's coordinates, which are the averages of the items categorized in that cluster so far.
3. We repeat the process for a given number of iterations and at the end, we have our clusters.

The "points" mentioned above are called means because they are the mean values of the items categorized in them. To initialize these means, we have a lot of options. An intuitive method is to initialize the means at random items in the data set. Another

method is to initialize the means at random values between the boundaries of the data set (if for a feature  $x$ , the items have values in  $[0,3]$ , we will initialize the means with values for  $x$  at  $[0,3]$ ).

The above algorithm in pseudocode is as follows:

Initialize  $k$  means with random values

--> For a given number of iterations:

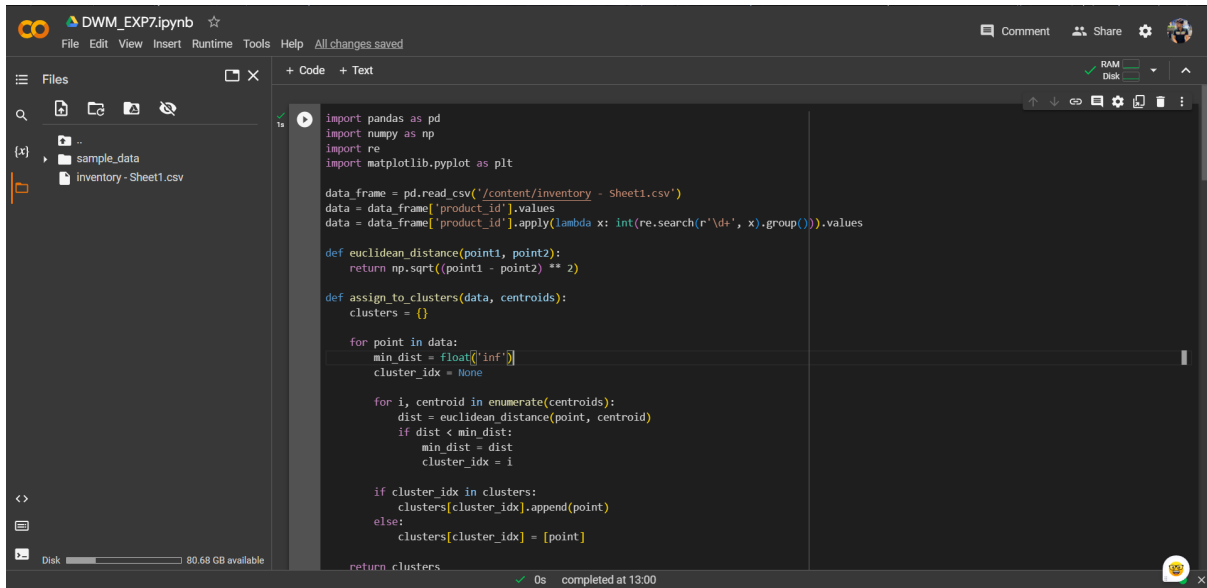
--> Iterate through items:

--> Find the mean closest to the item by calculating the Euclidean distance of the item with each of the means

--> Assign item to mean

--> Update mean by shifting it to the average of the items in that cluster

Output:



```

import pandas as pd
import numpy as np
import re
import matplotlib.pyplot as plt

data_frame = pd.read_csv('/content/inventory - Sheet1.csv')
data = data_frame['product_id'].values
data = data_frame['product_id'].apply(lambda x: int(re.search(r'\d+', x).group())).values

def euclidean_distance(point1, point2):
    return np.sqrt((point1 - point2) ** 2)

def assign_to_clusters(data, centroids):
    clusters = {}

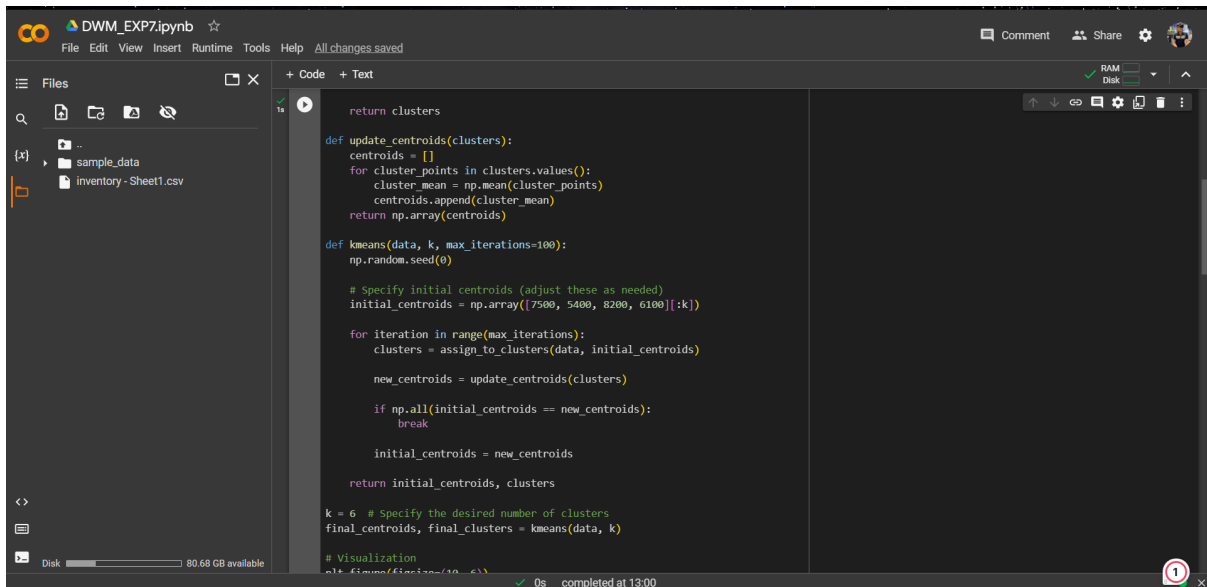
    for point in data:
        min_dist = float('inf')
        cluster_idx = None

        for i, centroid in enumerate(centroids):
            dist = euclidean_distance(point, centroid)
            if dist < min_dist:
                min_dist = dist
                cluster_idx = i

        if cluster_idx in clusters:
            clusters[cluster_idx].append(point)
        else:
            clusters[cluster_idx] = [point]

    return clusters

```



```

return clusters

def update_centroids(clusters):
    centroids = []
    for cluster_points in clusters.values():
        cluster_mean = np.mean(cluster_points)
        centroids.append(cluster_mean)
    return np.array(centroids)

def kmeans(data, k, max_iterations=100):
    np.random.seed(0)

    # Specify initial centroids (adjust these as needed)
    initial_centroids = np.array([7500, 5400, 8200, 6100][:k])

    for iteration in range(max_iterations):
        clusters = assign_to_clusters(data, initial_centroids)

        new_centroids = update_centroids(clusters)

        if np.all(initial_centroids == new_centroids):
            break

        initial_centroids = new_centroids

    return initial_centroids, clusters

k = 6 # Specify the desired number of clusters
final_centroids, final_clusters = kmeans(data, k)

# Visualization
plt.figure(figsize=(10, 6))

```



