**Lab Task 1 – 2-Means Clustering \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

In this task, use the provided dataset. Write the code which performs clustering of the dataset into 2 clusters. The pseudocode for the clustering algorithm is provided as follows:

**specify K number of centroids**

**randomly initialize K number of centroids u**

**for j = 1:epochs**

**for i = 1:m**

**c(i) = index of closest cluster to training example**

**for k = 1:K**

**u(k) = mean of all training examples indexed to k**

**plot of x1 and x2 clusters**

To determine the index c(i), you will need to write a function that calculates the Euclidean distance between the points in the feature space. This function will be used to find the closest centroid from each training example. After determining the indexes, the cluster centroids themselves are updated by taking the average of the x values. For k-th cluster, the training examples with index k will be averaged. This completes one iteration of clustering after which a scatter plot is made. The iterations are repeated until interesting groups are obtained in the plots.

Due to the initial randomization of cluster centroids, you may have to repeat the clustering a few times. Also, ensure the random centroids are from within the domain of the feature space.

Your code must generate scatter plots showing the clusters at each iteration. The input values must be colored and marked according to the cluster to which they belong at each iteration. The cluster centroids must also be shown Provide all of the codes and screenshots of the final output. You must include plots from at least 3 different iterations showing the progress of your clustering algorithm.

**CODE:**

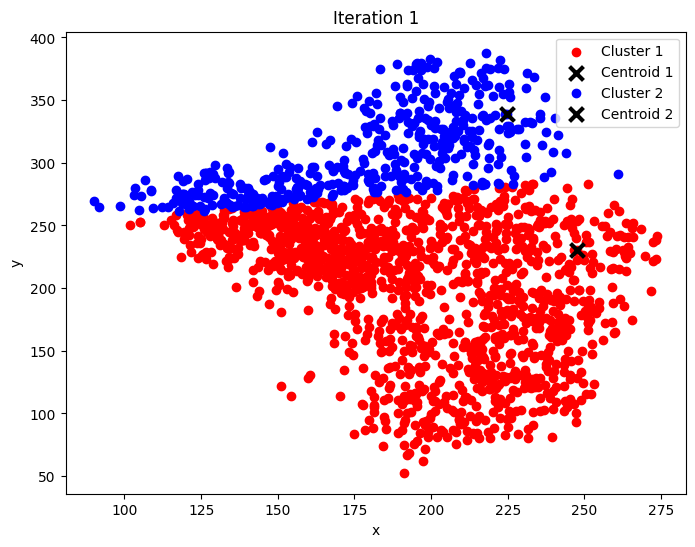
|  |
| --- |
| #Task01  import pandas as pd  df = pd.read\_csv("hyperplane.csv")  df.head()  import numpy as np  import matplotlib.pyplot as plt  X = df [['x', 'y']] .values  K = 2 #Setting clustoroids as 2  epochs = 5 # and setting iterations through the dataset as 5  #now selecting centroids as randomly from the dataset  centroids = X[np.random.choice(range(len(X)), K, replace = False)]  for epoch in range(epochs) :  # Step 2.1: Assign each point to the closest centroid  clusters = [[]for \_ in range(K)]  for point in X :  distances = [np.sqrt(np.sum((point - centroid) \* \*2)) for centroid in centroids]  closest\_centroid = np.argmin(distances)  clusters[closest\_centroid].append(point)  # Step 2.2: Calculate new centroids as the mean of each cluster  new\_centroids = []  for cluster in clusters :  new\_centroid = np.mean(cluster, axis = 0) if cluster else centroids[clusters.index(cluster)]  new\_centroids.append(new\_centroid)  new\_centroids = np.array(new\_centroids)  # Step 2.3: Plot clusters and centroids  plt.figure(figsize = (8, 6))  colors = ['r', 'b']  for i, cluster in enumerate(clusters) :  cluster = np.array(cluster)  if len(cluster) > 0:  plt.scatter(cluster[:, 0], cluster[:, 1], color = colors[i], label = f'Cluster {i+1}')  plt.scatter(centroids[i][0], centroids[i][1], color = 'black', marker = 'x', s = 100, linewidths = 3, label = f'Centroid {i+1}')  plt.title(f"Iteration {epoch + 1}")  plt.xlabel("x")  plt.ylabel("y")  plt.legend()  plt.show()  # Step 2.4: Update centroids for the next iteration  centroids = new\_centroids |

A red and blue dotted diagram

Description automatically generatedA red and blue dots

Description automatically generatedA red and blue dots

Description automatically generatedA red and blue dots

Description automatically generated**OUTPUT:**

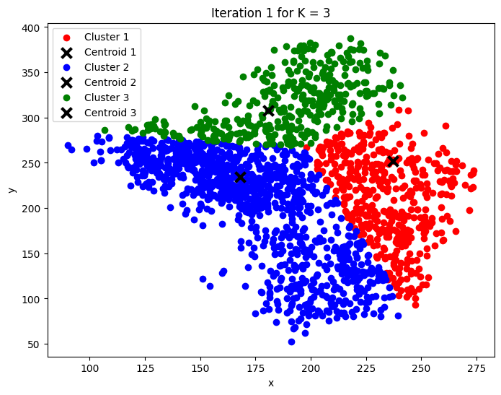
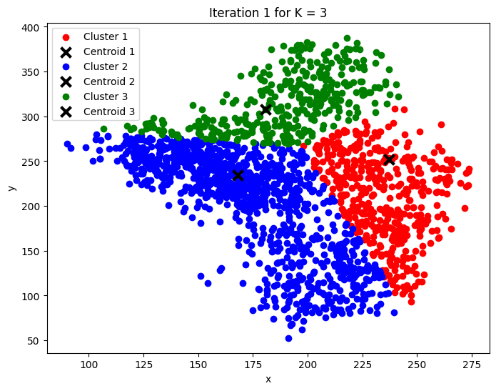
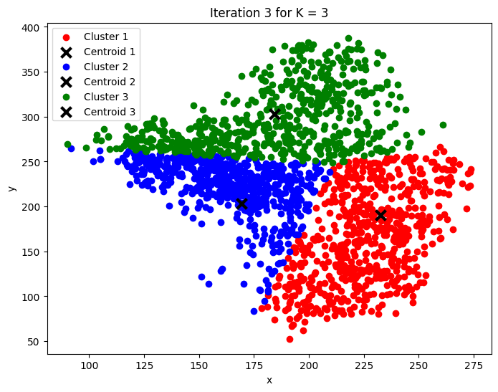
**Lab Task 2 – K-Means Clustering \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Repeat task 1, however, set value of K = 3, 4 and 5. For all three K values, generate at least three plots.

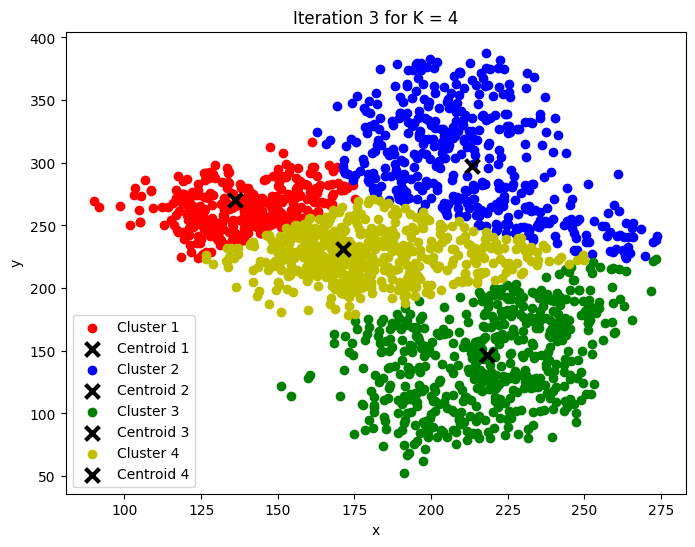
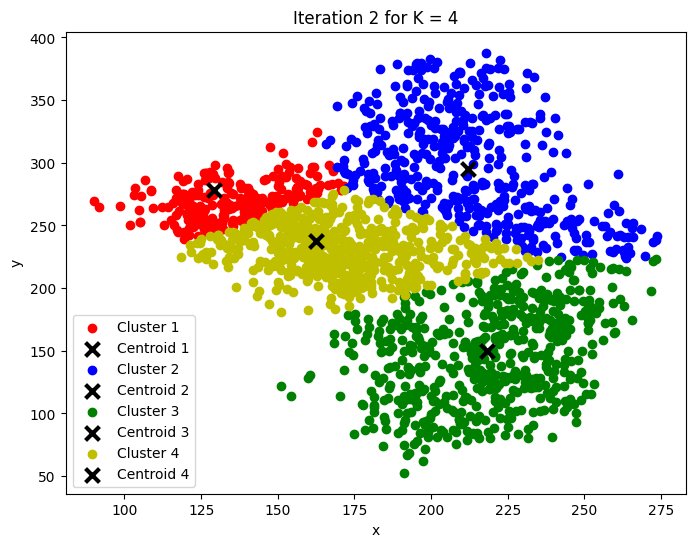
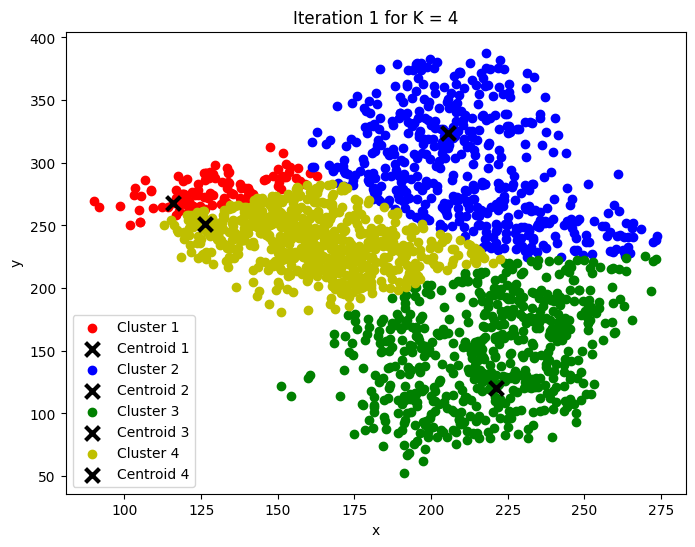
|  |
| --- |
| # Load the data  import pandas as pd  df = pd.read\_csv("hyperplane.csv")  import numpy as np  import matplotlib.pyplot as plt  # Extract the 'x' and 'y' columns to use as our features  X = df [['x', 'y']] .values  # Set the number of epochs(iterations) to 5 for each value of K  epochs = 3  # Loop over each K value we want to test  for K in[3, 4, 5]:  print(f"Clustering with K = {K}")  # Step 1: Randomly pick K data points from X as our initial centroids  centroids = X[np.random.choice(range(len(X)), K, replace = False)]  # Step 2: Start clustering, repeating for the specified number of epochs  for epoch in range(epochs) :  # Create a list of clusters, one for each centroid  clusters = [[]for \_ in range(K)]  # Step 2.1: Assign each point to the closest centroid  # For each data point, calculate its distance to each centroid, then assign it to the nearest one  for point in X :  distances = [np.sqrt(np.sum((point - centroid) \* \*2)) for centroid in centroids]  closest\_centroid = np.argmin(distances) # Index of the nearest centroid  clusters[closest\_centroid].append(point)  # Step 2.2: Recalculate centroids by averaging the points in each cluster  new\_centroids = []  for cluster in clusters :  # Calculate the mean position of each cluster(i.e., the new centroid)  new\_centroid = np.mean(cluster, axis = 0) if cluster else centroids[clusters.index(cluster)]  new\_centroids.append(new\_centroid)  new\_centroids = np.array(new\_centroids)  # Step 2.3: Plot the clusters and centroids to visualize each iteration  plt.figure(figsize = (8, 6))  colors = ['r', 'b', 'g', 'y', 'c'] # Colors for different clusters  for i, cluster in enumerate(clusters) :  cluster = np.array(cluster)  if len(cluster) > 0: # Plot only if the cluster has points  plt.scatter(cluster[:, 0], cluster[:, 1], color = colors[i % len(colors)], label = f'Cluster {i+1}')  plt.scatter(centroids[i][0], centroids[i][1], color = 'black', marker = 'x', s = 100, linewidths = 3, label = f'Centroid {i+1}')  plt.title(f"Iteration {epoch + 1} for K = {K}")  plt.xlabel("x")  plt.ylabel("y")  plt.legend()  plt.show()  # Update centroids for the next iteration  centroids = new\_centroids |

**OUTPUT:**

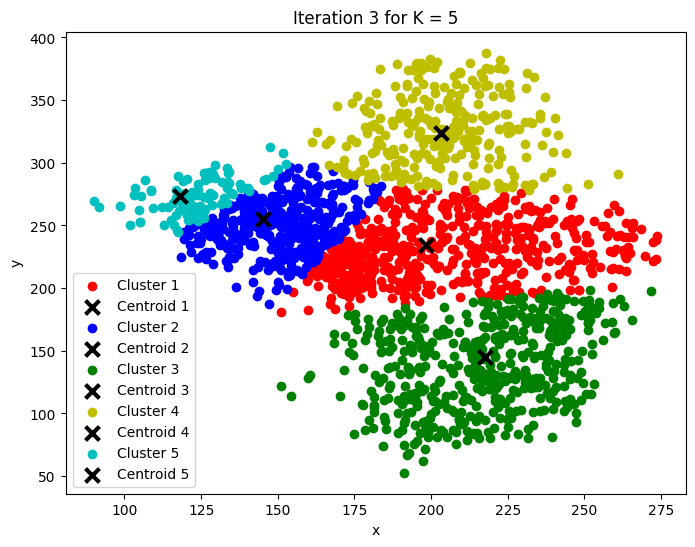
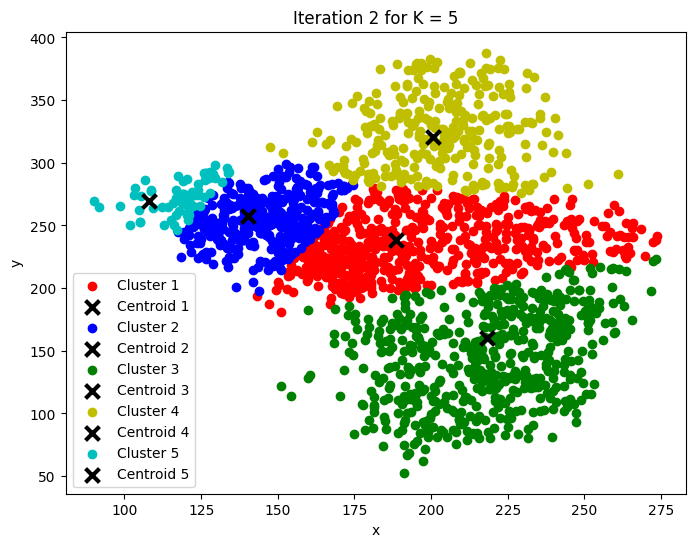
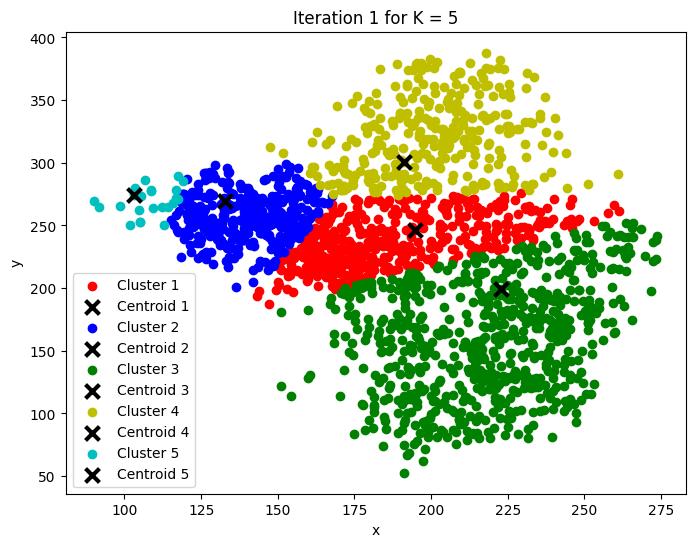
**For K=3**



**Now for K=4**



**And for K = 5**



**Lab Task 3 – Cost Function \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Load the given dataset into the python program for this task. In this task, you will modify your code so that it performs clustering from

K = 2, 3, 4… 10.

For each K value, perform about 20 iterations (epochs) of centroid update before moving to the next value of K. Additionally, at the last iteration of each K-value, determine the cost for that K-value:

**for K = 1:10**

**randomly initialize K number of centroids u**

**for j = 1:epochs**

**compute cluster centroids u(k)**

**plot of x1 and x2 clusters**

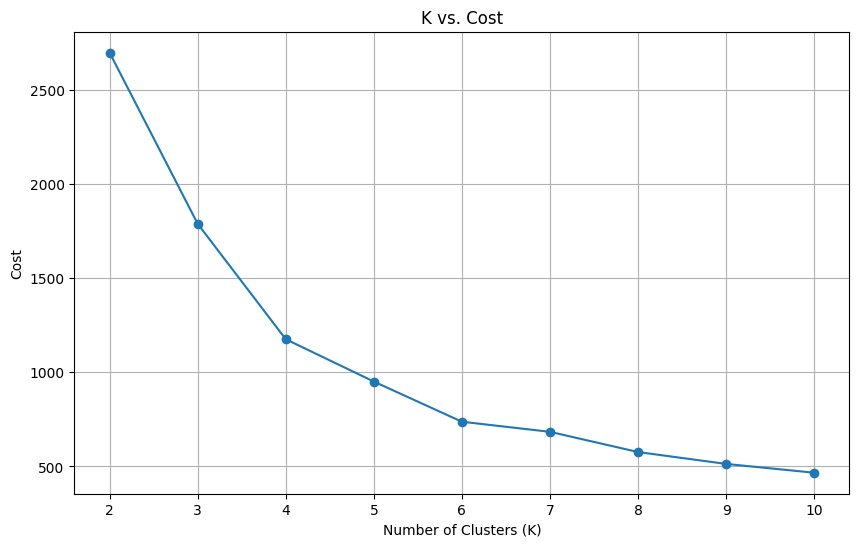
**compute cost for current K value**

**plot of cost and K**

Store the costs for each K value in a list. After the last iteration of the last cluster, make a plot of K vs. cost.

**CODE:**

|  |
| --- |
| import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  # Load the data  df = pd.read\_csv("hyperplane.csv")  X = df [['x', 'y']] .values  # Define the maximum number of clusters and number of epochs  max\_K = 10  epochs = 20  # Initialize an empty list to store the cost for each K  costs = []  # Loop over each K value from 2 to 10  for K in range(2, max\_K + 1) :  print(f"Running K-Means for K = {K}")  # Step 1: Randomly initialize K centroids  centroids = X[np.random.choice(range(len(X)), K, replace = False)]  # Step 2: Run the K - Means algorithm for the specified number of epochs  for epoch in range(epochs) :  # Create a list to hold the clusters  clusters = [[]for \_ in range(K)]  # Assign each data point to the nearest centroid  for point in X :  distances = [np.sqrt(np.sum((point - centroid) \* \*2)) for centroid in centroids]  closest\_centroid = np.argmin(distances)  clusters[closest\_centroid].append(point)  # Update the centroids to the mean of their assigned points  new\_centroids = []  for cluster in clusters :  new\_centroid = np.mean(cluster, axis = 0) if cluster else centroids[clusters.index(cluster)]  new\_centroids.append(new\_centroid)  centroids = np.array(new\_centroids)  # Calculate the cost for this K after the last iteration  cost = 0  m = len(X)  for i, cluster in enumerate(clusters) :  for point in cluster :  cost += np.sum((point - centroids[i]) \* \*2)  cost /= m  costs.append(cost)  # Plot the clusters at the last iteration for this K  plt.figure(figsize = (8, 6))  colors = ['r', 'b', 'g', 'y', 'c', 'm', 'k', 'orange', 'purple', 'brown'] # Add more colors if needed  for i, cluster in enumerate(clusters) :  cluster = np.array(cluster)  if len(cluster) > 0:  plt.scatter(cluster[:, 0], cluster[:, 1], color = colors[i % len(colors)], label = f'Cluster {i+1}')  plt.scatter(centroids[i][0], centroids[i][1], color = 'black', marker = 'x', s = 100, linewidths = 3, label = f'Centroid {i+1}')  plt.title(f"Clusters for K = {K}")  plt.xlabel("x")  plt.ylabel("y")  plt.legend()  plt.show()  # Plot the cost as a function of K  plt.figure(figsize = (10, 6))  plt.plot(range(2, max\_K + 1), costs, marker = 'o')  plt.title("K vs. Cost")  plt.xlabel("Number of Clusters (K)")  plt.ylabel("Cost")  plt.xticks(range(2, max\_K + 1))  plt.grid(True)  plt.show() |

**OUTPUT:**

**Lab Task 4 – Your Own Dataset \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Download your own CSV dataset from the internet (e.g. Kaggle). Your dataset must have at least 500 rows and at least 4 feature columns. Perform clustering of your dataset and showcase the plots (the cost function is optional in this task).

**CODE:**

|  |
| --- |
| #Task04  #using wine quality dataset downloaded from kaggle.com named WineQT  import pandas as pd  df = pd.read\_csv("WineQT.csv")  df.head()  import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  X = df [['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar']] .values  # Define the maximum number of clusters and number of epochs  max\_K = 10  epochs = 20  # Initialize an empty list to store the cost for each K(optional)  costs = []  # Loop over each K value from 2 to 10  for K in range(2, max\_K + 1) :  print(f"Running K-Means for K = {K}")  # Step 1: Randomly initialize K centroids  centroids = X[np.random.choice(range(len(X)), K, replace = False)]  # Step 2: Run the K - Means algorithm for the specified number of epochs  for epoch in range(epochs) :  # Create a list to hold the clusters  clusters = [[]for \_ in range(K)]  # Assign each data point to the nearest centroid  for point in X :  distances = [np.sqrt(np.sum((point - centroid) \* \*2)) for centroid in centroids]  closest\_centroid = np.argmin(distances)  clusters[closest\_centroid].append(point)  # Update the centroids to the mean of their assigned points  new\_centroids = []  for cluster in clusters :  new\_centroid = np.mean(cluster, axis = 0) if cluster else centroids[clusters.index(cluster)]  new\_centroids.append(new\_centroid)  centroids = np.array(new\_centroids)  # Calculate the cost for this K after the last iteration(optional)  cost = 0  m = len(X)  for i, cluster in enumerate(clusters) :  for point in cluster :  cost += np.sum((point - centroids[i]) \* \*2)  cost /= m  costs.append(cost)  # Plot the clusters at the last iteration for this K  plt.figure(figsize = (8, 6))  colors = plt.cm.get\_cmap('viridis', K)  for i, cluster in enumerate(clusters) :  cluster = np.array(cluster)  if len(cluster) > 0:  plt.scatter(cluster[:, 0], cluster[:, 1], color = colors(i), label = f'Cluster {i+1}')  plt.scatter(centroids[i][0], centroids[i][1], color = 'black', marker = 'x', s = 100, linewidths = 3, label = f'Centroid {i+1}')  plt.title(f"Clusters for K = {K}")  plt.xlabel("Fixed Acidity")  plt.ylabel("Volatile Acidity")  plt.legend()  plt.show()  # Plot the cost as a function of K  plt.figure(figsize = (10, 6))  plt.plot(range(2, max\_K + 1), costs, marker = 'o')  plt.title("K vs. Cost")  plt.xlabel("Number of Clusters (K)")  plt.ylabel("Cost")  plt.xticks(range(2, max\_K + 1))  plt.grid(True)  plt.show() |

A diagram of a cluster of dots

Description automatically generated with medium confidenceA diagram of a cluster of colored dots

Description automatically generatedA yellow and purple dots

Description automatically generated **OUTPUT:**

A graph with a line

Description automatically generated