To **recluster based on the mean**, we use the recalculated centroids (means of each cluster) to reassign data points to the closest centroid. Let's break this down step by step with an **example**:

Example

Dataset (2D points):

We have the following points in a 2D space:

$$P = \{(1,1),(2,1),(4,3),(5,4)\}$$

Assume we want to form **2 clusters** ($k = 2$).

Step 1: Initialize Centroids

Choose two random initial centroids:

$$C_1 = (1,1), \quad C_2 = (5,4)$$

Step 2: Assign Points to Closest Centroid

Calculate the **Euclidean distance** of each point to both centroids and assign each point to the closest centroid.

Distance Calculation:

1. Distance formula:
$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$
2. Calculate distances:
   - For point $(1,1)$:
     - To C_1: $d = \sqrt{(1-1)^2 + (1-1)^2} = 0$
     - To C_2: $d = \sqrt{(1-5)^2 + (1-4)^2} = \sqrt{16+9} = 5$
       Assign to C_1.
   - For point $(2,1)$:
     - To C_1: $d = \sqrt{(2-1)^2 + (1-1)^2} = 1$
     - To C_2: $d = \sqrt{(2-5)^2 + (1-4)^2} = \sqrt{9+9} = \sqrt{18} \approx 4.24$
       Assign to C_1.
   - For point $(4,3)$:
     - To C_1: $d = \sqrt{(4-1)^2 + (3-1)^2} = \sqrt{9+4} = \sqrt{13} \approx 3.61$
     - To C_2: $d = \sqrt{(4-5)^2 + (3-4)^2} = \sqrt{1+1} = \sqrt{2} \approx 1.41$
       Assign to C_2.
   - For point $(5,4)$:
     - To C_1: $d = \sqrt{(5-1)^2 + (4-1)^2} = \sqrt{16+9} = \sqrt{25} = 5$

- To C_2: $d = \sqrt{(5-5)^2 + (4-4)^2} = 0$
  Assign to C_2.

Cluster Assignments:

- Cluster 1 (C_1): $(1, 1), (2, 1)$
- Cluster 2 (C_2): $(4, 3), (5, 4)$

Step 3: Recalculate Centroids (Cluster Means)

Compute the mean of each cluster:

For C_1:

Points: $(1, 1), (2, 1)$

$$C_1 = (\frac{1+2}{2}, \frac{1+1}{2}) = (1.5, 1)$$

For C_2:

Points: $(4, 3), (5, 4)$

$$C_2 = (\frac{4+5}{2}, \frac{3+4}{2}) = (4.5, 3.5)$$

New Centroids:

$$C_1 = (1.5, 1), \quad C_2 = (4.5, 3.5)$$

Step 4: Reassign Points to Closest Centroid

Recompute the distances between all points and the new centroids:

Distance Calculation with Updated Centroids:

- For point $(1, 1)$:
  - To C_1: $d = \sqrt{(1-1.5)^2 + (1-1)^2} = 0.5$
  - To C_2: $d = \sqrt{(1-4.5)^2 + (1-3.5)^2} = \sqrt{12.25 + 6.25} = \sqrt{18.5} \approx 4.3$
    Assign to C_1.
- For point $(2, 1)$:
  - To C_1: $d = \sqrt{(2-1.5)^2 + (1-1)^2} = 0.5$
  - To C_2: $d = \sqrt{(2-4.5)^2 + (1-3.5)^2} = \sqrt{6.25 + 6.25} = \sqrt{12.5} \approx 3.54$
    Assign to C_1.
- For point $(4, 3)$:
  - To C_1: $d = \sqrt{(4-1.5)^2 + (3-1)^2} = \sqrt{6.25 + 4} = \sqrt{10.25} \approx 3.2$
  - To C_2: $d = \sqrt{(4-4.5)^2 + (3-3.5)^2} = \sqrt{0.25 + 0.25} = \sqrt{0.5} \approx 0.71$
    Assign to C_2.
- For point $(5, 4)$:
  - To C_1: $d = \sqrt{(5-1.5)^2 + (4-1)^2} = \sqrt{12.25 + 9} = \sqrt{21.25} \approx 4.61$

- To C_2: $d = \sqrt{(5-4.5)^2 + (4-3.5)^2} = \sqrt{0.25 + 0.25} = \sqrt{0.5} \approx 0.71$
  Assign to C_2.

Updated Cluster Assignments:

- Cluster 1 (C_1): $(1, 1), (2, 1)$
- Cluster 2 (C_2): $(4, 3), (5, 4)$

Repeat Until Convergence

Continue recalculating centroids and reassigning points until the centroids stop changing positions or the changes are below a predefined threshold.

This iterative process ensures that the clusters are formed such that intra-cluster variance is minimized!

how to calculate total variation within each cluster and how to came to know that variation is zero or minimum by using above data points and how to visualize variation of each clusters ?

Understanding Total Variation Within Each Cluster

Total variation within a cluster is a measure of how spread out the data points are from the cluster's centroid. In other words, it quantifies the **variance** or **dispersion** of the data points in each cluster. To calculate total variation, we typically calculate the **sum of squared distances** between each data point and the centroid of its assigned cluster.

The formula for the total variation (within-cluster sum of squares, WCSS) for a cluster is:

$$WCSS = \sum_{i=1}^{n} (d_i - C)^2$$

Where:

- d_i is the i-th data point in the cluster.
- C is the centroid of the cluster.
- n is the number of data points in the cluster.

To calculate the total variation across all clusters, you sum the WCSS for each cluster:

$$\text{Total Variation} = \sum_{k=1}^{K} WCSS_k$$

Where K is the total number of clusters.

Steps to Calculate and Visualize Total Variation (WCSS)

Let's use the same example dataset to calculate and visualize the total variation for each cluster.

Step 1: Calculate WCSS for Each Cluster

Using the previous cluster assignments, we have:

- **Cluster 1** ($C_1 = (1.5, 1)$) with points: $(1, 1)$, $(2, 1)$
- **Cluster 2** ($C_2 = (4.5, 3.5)$) with points: $(4, 3)$, $(5, 4)$

Cluster 1:

1. Point $(1, 1)$:
   - Distance from centroid $C_1 = (1.5, 1)$:

$$\text{Distance} = \sqrt{(1 - 1.5)^2 + (1 - 1)^2} = 0.5$$

   - Squared distance: $0.5^2 = 0.25$
2. Point $(2, 1)$:
   - Distance from centroid $C_1 = (1.5, 1)$:

$$\text{Distance} = \sqrt{(2 - 1.5)^2 + (1 - 1)^2} = 0.5$$

   - Squared distance: $0.5^2 = 0.25$

WCSS for Cluster 1:

$$WCSS_1 = 0.25 + 0.25 = 0.5$$

Cluster 2:

1. Point $(4, 3)$:
   - Distance from centroid $C_2 = (4.5, 3.5)$:

$$\text{Distance} = \sqrt{(4 - 4.5)^2 + (3 - 3.5)^2} = 0.707$$

   - Squared distance: $0.707^2 = 0.5$
2. Point $(5, 4)$:
   - Distance from centroid $C_2 = (4.5, 3.5)$:

$$\text{Distance} = \sqrt{(5 - 4.5)^2 + (4 - 3.5)^2} = 0.707$$

   - Squared distance: $0.707^2 = 0.5$

WCSS for Cluster 2:

$$WCSS_2 = 0.5 + 0.5 = 1.0$$

Step 2: Calculate Total Variation (WCSS across all clusters)

Now, let's compute the total variation:

$$\text{Total Variation} = WCSS_1 + WCSS_2 = 0.5 + 1.0 = 1.5$$

Step 3: Check if Variation is Minimum (Convergence)

- The total variation will **reduce** as the centroids converge towards the optimal positions.
- If the variation doesn't change after a few iterations, it indicates that the clustering process has converged, and further recalculations won't improve the results.

To determine whether the variation is zero or minimized:

- **Zero variation** would imply that all points in a cluster are **identical**, which is extremely rare.

- The **minimum variation** means the points are as close as possible to the centroid, minimizing the distance within each cluster.

Step 4: Visualize the Variation of Each Cluster

You can visualize the variation within each cluster using a scatter plot and by plotting the **sum of squared distances**.

python

Copy code

```
import numpy as np import matplotlib.pyplot as plt # Define the points and centroids cluster1_points = np.array([[1, 1], [2, 1]]) cluster2_points = np.array([[4, 3], [5, 4]]) C1 = np.array([1.5, 1]) C2 = np.array([4.5, 3.5]) # Calculate the squared distances for each cluster wcss_1 = np.sum(np.square(np.linalg.norm(cluster1_points - C1, axis=1))) wcss_2 = np.sum(np.square(np.linalg.norm(cluster2_points - C2, axis=1))) # Plot the points and centroids plt.figure(figsize=(8, 6)) plt.scatter(cluster1_points[:, 0], cluster1_points[:, 1], color='blue', label='Cluster 1') plt.scatter(cluster2_points[:, 0], cluster2_points[:, 1], color='green', label='Cluster 2') plt.scatter(C1[0], C1[1], color='red', marker='x', label='Centroid 1') plt.scatter(C2[0], C2[1], color='orange', marker='x', label='Centroid 2') # Annotate the plot plt.text(C1[0], C1[1], f'C1 (WCSS: {wcss_1:.2f})', fontsize=12, color='red') plt.text(C2[0], C2[1], f'C2 (WCSS: {wcss_2:.2f})', fontsize=12, color='orange') # Labels and title plt.xlabel('X-axis') plt.ylabel('Y-axis') plt.title('Cluster Visualization with Centroids') plt.legend() plt.show()
```

Visualizing WCSS:

To visualize the **within-cluster sum of squares (WCSS)**, you can plot the WCSS values for each cluster as a bar graph:

python

Copy code

```
# Plot WCSS clusters = ['Cluster 1', 'Cluster 2'] wcss_values = [wcss_1, wcss_2] plt.bar(clusters, wcss_values, color=['blue', 'green']) plt.title('WCSS for Each Cluster') plt.xlabel('Clusters') plt.ylabel('WCSS') plt.show()
```

This will give you a clear visual representation of how much variation exists within each cluster, with lower WCSS indicating tighter, more compact clusters.

Conclusion

- **Zero variation**: Would only occur if all points within a cluster are the same (rare).
- **Minimum variation**: When the total variation doesn't change significantly between iterations, and the clusters are well-separated.
- **WCSS**: A key indicator to track the tightness of your clusters during the K-means algorithm, and it can be visualized for better understanding of cluster dispersion.

can u take practical example of k means clustering ? and do all process with mathmatical process and python coding with visulazation ?

Certainly! Let's go through a practical example of **K-means clustering** using mathematical steps, followed by Python code, and finally visualizations.

Example Problem

We have a small dataset of 2D data points, and we want to apply K-means clustering to partition these points into 2 clusters.

Step-by-step Process

1. **Initialization**: Choose the number of clusters (K). Let's say $K = 2$.
2. **Randomly initialize centroids**: Randomly select 2 points as centroids.
3. **Assign each data point to the nearest centroid** based on the Euclidean distance.
4. **Recalculate the centroids**: Compute the mean of all points assigned to each centroid.
5. **Repeat steps 3 and 4** until the centroids don't change significantly (convergence).

Mathematical Process

1. **Step 1**: Choose K = 2 clusters.
2. **Step 2**: Randomly initialize centroids. Let's say our initial centroids are $C_1 = (2, 3)$ and $C_2 = (6, 8)$.
3. **Step 3**: Compute Euclidean distance for each point to the centroids.
   - For a point (x_1, y_1) and a centroid (x_2, y_2), the Euclidean distance is:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

4. **Step 4**: Recalculate the centroids by averaging the points assigned to each cluster.

$$C_1 = \left(\frac{\sum x_1}{n_1}, \frac{\sum y_1}{n_1}\right)$$

   where n_1 is the number of points in cluster 1, and similarly for C_2.

5. **Step 5**: Repeat steps 3 and 4 until centroids don't change significantly.

Python Code Implementation

Let's define some data points and perform K-means clustering.

Step 1: Import Libraries and Define Dataset

python Copy code

```python
import numpy as np import matplotlib.pyplot as plt from sklearn.cluster import KMeans # Create a dataset with 2D points X = np.array([[1, 2], [2, 3], [3, 3], [6, 5], [7, 6], [8, 7], [9, 8]]) # Plot the data points plt.scatter(X[:, 0], X[:, 1], c='blue', marker='o', label='Data Points') plt.title('Dataset for K-Means Clustering') plt.xlabel('X') plt.ylabel('Y') plt.legend() plt.show()
```

Step 2: Apply K-means Clustering

python

Copy code

```python
# Perform K-means clustering (K=2) kmeans = KMeans(n_clusters=2, init='random', n_init=10, max_iter=300, random_state=42) kmeans.fit(X) # Get the predicted labels (cluster assignment) labels = kmeans.labels_ # Get the centroids of each cluster centroids = kmeans.cluster_centers_ # Visualize the clusters and centroids plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', marker='o', label='Data Points') plt.scatter(centroids[:, 0], centroids[:, 1], color='red', marker='x', s=200, label='Centroids') plt.title('K-Means Clustering with K=2') plt.xlabel('X') plt.ylabel('Y') plt.legend() plt.show() print("Centroids: ", centroids)
```

Explanation of the Python Code

- **Dataset (`X`)**: We define a small set of 2D points that we want to cluster.
- **KMeans**: We use `KMeans` from `sklearn` to perform the clustering. Here, `n_clusters=2` means we are dividing the dataset into 2 clusters.
- **Centroids**: After clustering, we obtain the centroids of the clusters, which are the mean of the points in each cluster.

Step 3: Mathematical Calculation Example (Manually)

1. **Initialization**:
   - Centroids: $C_1 = (2, 3)$, $C_2 = (6, 8)$.
2. **Step 1: Calculate Distances** (Euclidean distance)

   For the first point $(1, 2)$:
   - Distance to $C_1 = (2, 3)$:
   $$d_1 = \sqrt{(1-2)^2 + (2-3)^2} = \sqrt{1+1} = \sqrt{2} \approx 1.414$$
   - Distance to $C_2 = (6, 8)$:
   $$d_2 = \sqrt{(1-6)^2 + (2-8)^2} = \sqrt{25+36} = \sqrt{61} \approx 7.810$$

   The point $(1, 2)$ is closer to C_1.
3. **Step 2: Assign Points to Nearest Centroid**
   - Point $(1, 2)$ goes to cluster 1 (closer to C_1).
   - Similarly, for all points, you calculate the distance to both centroids and assign each point to the nearest centroid.
4. **Step 3: Recalculate Centroids**
   - Calculate the new centroids by taking the mean of all points in each cluster.

   For cluster 1 (points $(1, 2)$, $(2, 3)$):
   - $C_1 = (\frac{1+2}{2}, \frac{2+3}{2}) = (1.5, 2.5)$

   For cluster 2 (points $(6, 5)$, $(7, 6)$, $(8, 7)$, $(9, 8)$):
   - $C_2 = (\frac{6+7+8+9}{4}, \frac{5+6+7+8}{4}) = (7.5, 6.5)$

Step 4: Repeat Until Convergence

Repeat the above steps (distance calculation, assignment, and centroid recalculation) until the centroids no longer change significantly. In this case, after a few iterations, the centroids will stabilize, and the clustering process will stop.

Visualizing the Results

The following plot will show the data points with different colors representing different clusters, along with the centroids marked by red "X":

- Points closer to $C_1 = (1.5, 2.5)$ will be assigned to Cluster 1.
- Points closer to $C_2 = (7.5, 6.5)$ will be assigned to Cluster 2.

python Copy code

```python
# Visualize the clusters with the final centroids plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', marker='o', label='Data Points') plt.scatter(centroids[:, 0], centroids[:, 1], color='red', marker='x', s=200, label='Centroids') plt.title('Final K-Means Clustering') plt.xlabel('X') plt.ylabel('Y') plt.legend() plt.show()
```

Conclusion

- **Centroids**: The centroids represent the "center" of each cluster, calculated by averaging the points assigned to each cluster.
- **Convergence**: K-means stops when the centroids no longer change after an iteration (or a maximum number of iterations is reached).
- **Visualization**: You can visualize the data points, the clusters, and the centroids to understand how K-means is dividing the data.

This example demonstrates the entire K-means clustering process, from initialization to convergence, with both mathematical calculations and Python code implementation.

i am talking about data set of sales or health , having few features and relate with k means clustering?

Ah, I see! You're looking for an example where K-means clustering is applied to a real-world dataset such as sales or health data. Let's walk through the process of performing K-means clustering on a dataset, focusing on sales or health-related features.

For this, I will use a **sales dataset** (or similar structured dataset) to demonstrate how to perform K-means clustering. Here's a practical example with a dataset containing sales data, where we want to segment customers into clusters based on features like **age, annual income**, and **spending score**.

Steps to Perform K-means Clustering on a Sales/Health Dataset

We'll follow the general steps of K-means clustering as described earlier, but this time, we'll work with a more complex dataset containing features that relate to sales or health metrics.

Let's assume the dataset has the following columns:

- **Age**: The age of the customer.
- **Annual Income**: The annual income of the customer.
- **Spending Score**: A score given to the customer based on their purchasing behavior.

We want to segment customers into different clusters (e.g., high spenders, low spenders) using K-means clustering.

Step-by-step Explanation with Code and Visualization

Step 1: Dataset Creation or Import

For this example, we will assume we have a sales dataset like this:

CustomerID	Age	Annual_Income	Spending_Score
1	25	40000	50
2	30	50000	60
3	35	60000	70
4	40	70000	80
5	45	80000	90
6	50	90000	95
7	60	100000	100
8	65	110000	110
9	70	120000	120

You can imagine this as part of a sales dataset where we want to understand customer spending behavior.

Step 2: Preprocessing

Before applying K-means, it's essential to:

- Handle missing values (if any).
- Normalize the features (because K-means is sensitive to the scale of features).
- Choose the number of clusters (K).

Step 3: Apply K-means Clustering

We'll perform K-means clustering on this dataset. Let's take $K = 3$ (which means we're segmenting the data into 3 clusters).

Step 4: Code Implementation

```python
import pandas as pd import numpy as np import matplotlib.pyplot as plt from
sklearn.cluster import KMeans from sklearn.preprocessing import StandardScaler # Step 1:
Create the dataset data = { 'Age': [25, 30, 35, 40, 45, 50, 60, 65, 70],
'Annual_Income': [40000, 50000, 60000, 70000, 80000, 90000, 100000, 110000, 120000],
'Spending_Score': [50, 60, 70, 80, 90, 95, 100, 110, 120] } df = pd.DataFrame(data) #
Step 2: Preprocessing - Standardize the data scaler = StandardScaler() scaled_data =
scaler.fit_transform(df[['Age', 'Annual_Income', 'Spending_Score']]) # Step 3: Apply K-
means clustering kmeans = KMeans(n_clusters=3, random_state=42) df['Cluster'] =
kmeans.fit_predict(scaled_data) # Step 4: Get centroids centroids =
kmeans.cluster_centers_ # Step 5: Visualize the clusters and centroids
plt.figure(figsize=(8, 6)) # Plotting each cluster plt.scatter(df['Age'],
df['Spending_Score'], c=df['Cluster'], cmap='viridis', s=100, label='Data Points')
plt.scatter(centroids[:, 0], centroids[:, 2], color='red', marker='X', s=200,
label='Centroids') plt.title('K-Means Clustering of Customers') plt.xlabel('Age')
plt.ylabel('Spending Score') plt.legend() plt.show() print("Centroids: ", centroids)
```

Step 5: Visualize the Results

The **scatter plot** will show the following:

- Data points colored by the cluster they belong to.
- The **red "X" markers** represent the centroids of the clusters.

The visualization will help us see how customers are grouped based on their age and spending score. For example, the clusters might indicate:

- One group of young customers with low spending scores.
- Another group of middle-aged customers with moderate spending scores.
- A third group of older customers with high spending scores.

Mathematical Steps

1. **Initialization**: Select initial centroids, say C_1, C_2, and C_3 randomly or based on the data.
2. **Calculate distances**: For each point, calculate the Euclidean distance to each centroid.

$$d = \sqrt{(Age_i - C_{1,Age})^2 + (SpendingScore_i - C_{1,SpendingScore})^2}$$

3. **Assign clusters**: Assign each data point to the nearest centroid.
4. **Recalculate centroids**: After assigning points to clusters, recalculate the centroids of each cluster by taking the mean of the data points in each cluster.
   - For each cluster k, the centroid is recalculated as:

$$C_k = \left(\frac{\sum x}{n_k}, \frac{\sum y}{n_k}\right)$$

   where x and y are the feature values (e.g., Age, Spending Score), and n_k is the number of points in cluster k.
5. **Repeat**: Repeat steps 2 and 3 until the centroids no longer change.

Understanding the Clusters

After performing K-means clustering, the `df['Cluster']` column will indicate which cluster each customer belongs to. Based on the **centroid values**, you can analyze the characteristics of each cluster, such as:

- **Cluster 1**: Young customers with low income and low spending scores.
- **Cluster 2**: Middle-aged customers with moderate income and spending.
- **Cluster 3**: Older customers with high income and high spending.

How to Interpret Results

- The **distance between centroids** helps us understand the spread of each cluster.
- The **size** of the cluster can indicate how many customers fall into that category.
- You can refine the number of clusters K by trying different values (using **elbow method** or **silhouette score**).

Conclusion

In this example, we applied K-means clustering to segment customers based on their **age, annual income**, and **spending score**. The steps involved preprocessing the data (scaling), applying K-means, and visualizing the results. By clustering, you can gain insights into customer behavior, segment them into distinct groups, and tailor marketing strategies accordingly.

This approach can be applied to various other datasets such as health data, where features like age, health conditions, and spending on healthcare can be clustered to identify patient profiles.