

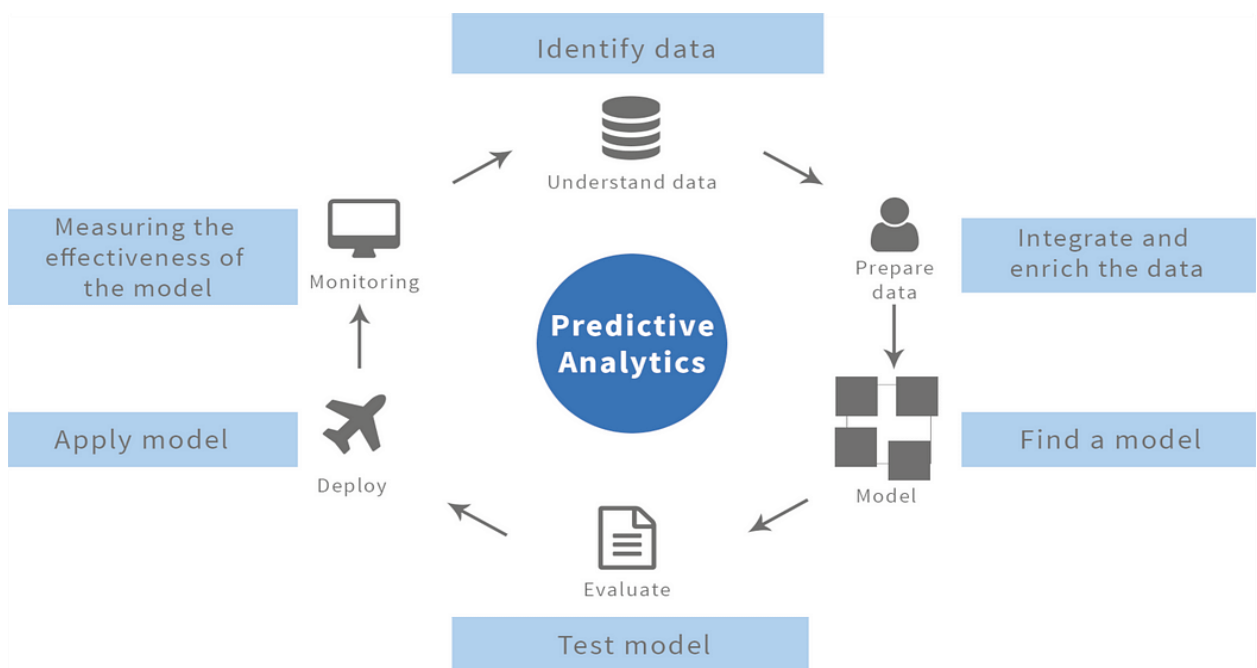
## Lesson 8: Unsupervised methods in Data science

### Clusters methods

#### Introduction

- **Clustering is an unsupervised** machine learning technique with a lot of applications in the areas of pattern recognition, image analysis, customer analytics, market segmentation, social network analysis, and more. A broad range of industries use clustering, from airlines to healthcare and beyond.
- It is a type of unsupervised learning, meaning that we do not need labeled data for clustering algorithms; this is one of the biggest advantages of clustering over other supervised learning like Classification.
- Clustering is the process of arranging a group of objects in such a manner that the objects in the same group (which is referred to as a cluster) are more similar to each other than to the objects in any other group. Data professionals often use clustering in the Exploratory Data Analysis phase to discover new information and patterns in the data. As clustering is unsupervised machine learning, it doesn't require a labeled dataset.
- Clustering itself is not one specific algorithm but the general task to be solved. You can achieve this goal using various algorithms that differ significantly in their understanding of what constitutes a cluster and how to find them efficiently.
- Clustering is an unsupervised machine learning technique. It does not require labeled data for training.

#### Predictive analytics in clustering



Predictive analytics in clustering involves using clustering techniques to make predictions or inform decision-making based on the patterns and structures discovered within the data. While clustering is primarily an unsupervised learning technique for grouping similar data points, it can be combined with other methods to perform predictive tasks. Here's how predictive analytics is applied in clustering:

- ✚ **Cluster Profiling:** After clusters are created, you can analyze the characteristics and behavior of the data points within each cluster. This profiling can help make predictions about the behavior of new, unseen data points. For example, if you have clusters of customer segments, you can predict the likely purchasing behavior of new customers based on which cluster they are assigned to.
- ✚ **Anomaly Detection:** Clustering can be used to identify clusters of "normal" data and detect anomalies or outliers. Predictive analytics can then be applied to predict when new data points fall into the outlier cluster, indicating potential issues or anomalies in the data. Anomaly detection is used in fraud detection, network security, and quality control.
- ✚ **Recommendation Systems:** In recommendation systems, clustering can group users or items with similar preferences. Predictive analytics is used to recommend products, services, or content to users based on the preferences of similar users or items in their cluster. This approach is common in e-commerce and content platforms.
- ✚ **Customer Segmentation:** After clustering customers based on their behavior, demographics, or preferences, predictive analytics can be applied to make predictions about future behavior. For example, you can predict which products a customer is likely to buy or how likely a customer is to churn based on their cluster.
- ✚ **Resource Allocation:** In resource management, clustering can be used to group similar resources, such as servers or equipment, and predictive analytics can be applied to predict which resources are likely to fail or require maintenance based on the behavior of their cluster.
- ✚ **Text and Document Clustering:** In text mining and natural language processing, clustering is used to group similar documents or text data.

Predictive analytics can be used to predict the topic or sentiment of new, unlabeled documents based on their cluster.

✚ **Market Basket Analysis:** Clustering can identify groups of frequently co-occurring items in transaction data. Predictive analytics can be applied to suggest which items are likely to be purchased together in future transactions, informing inventory management and marketing strategies.

✚ **Image Clustering and Object Detection:** In computer vision, clustering can group similar objects in images. Predictive analytics can be used to predict the presence or location of similar objects in new images.

### Key Success Criteria for Clustering Analysis

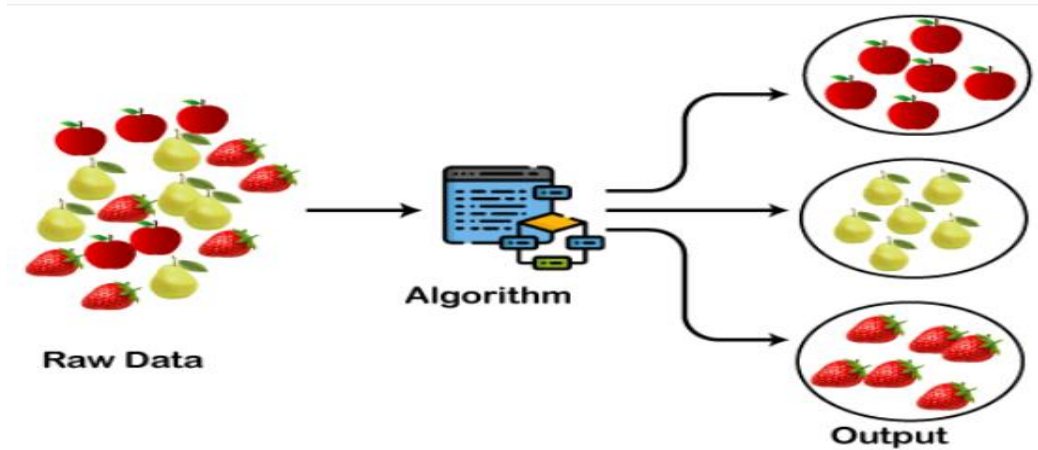
- Clustering, unlike supervised learning use-cases such as classification or regression, cannot be completely automated end-to-end. Instead, it is an iterative process of information discovery that requires domain expertise and human judgment used frequently to adjust the data and the model parameters to achieve the desired result.
- Most importantly, because clustering is unsupervised learning and doesn't use labeled data, we cannot calculate performance metrics like accuracy, AUC Area Under the Receiver Operating Characteristic (ROC) , RMSE (**Root Mean Squared Error**) , etc., to compare different algorithms or data preprocessing techniques. As a result, this makes it really challenging and subjective to assess the performance of clustering models.

### The key success criteria in clustering models revolve around:

- Is it interpretable?
- Is the output of clustering useful for business?
- Have you learned new information or discovered new patterns in the data that you weren't aware of before clustering?

### Building Intuition behind Clustering

- Before diving into algorithmic details, let's just build an intuition behind clustering using a toy example of fruit datasets. Let's say we have a huge collection of image dataset containing three fruits (i) strawberries, (ii) pears, and (iii) apples.
- In the dataset all the images are mixed up and your use-case is to group similar fruits together i.e. create three groups with each one of them containing one type of fruit. This is exactly what a clustering algorithm will do.



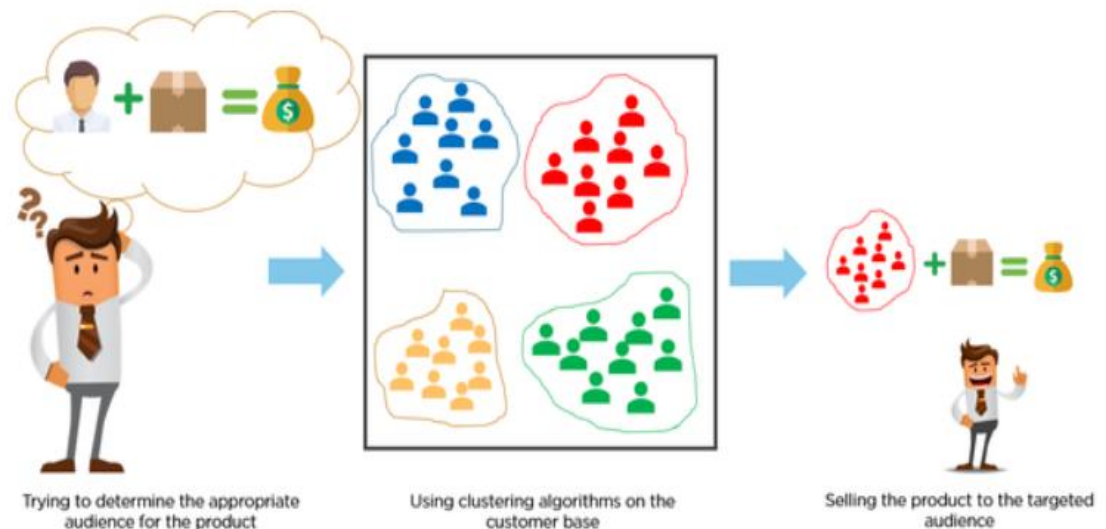
### **Business Applications of Clustering**

Clustering is a very powerful technique and has broad applications in various industries ranging from media to healthcare, manufacturing to service industries, and anywhere you have large amounts of data. Let's take a look at some practical use-cases:

#### **1) Customer Segmentation.**

Customers are categorized by using clustering algorithms according to their purchasing behavior or interests to develop focused marketing campaigns.

Imagine you have 10M customers, and you want to develop customized or focused marketing campaigns. It is unlikely that you will develop 10M marketing campaigns, so what do we do? We could use clustering to group 10M customers into 25 clusters and then design 25 marketing campaigns instead of 10M.



Image

Source: [https://miro.medium.com/max/845/1\\*rFATWK6tWBrDJ1o1rzEZ8w.png](https://miro.medium.com/max/845/1*rFATWK6tWBrDJ1o1rzEZ8w.png)

## 2) Retail Clustering

There are many opportunities for clustering in retail businesses. For example, you can gather data on each store and cluster at store level to generate insights that may tell you which locations are similar to each other based on attributes like foot traffic, average store sales, number of SKUs - Stock Keeping Units., etc.

Another example could be clustering at a category level. In the diagram below, we have eight stores. Different colors represent different clusters. There are four clusters in this example.

Notice that the deodorants category in Store 1 is represented by the red cluster, whereas the deodorants category in Store 2 is represented by the blue cluster. This depicts that Store 1 and Store 2 have completely different target markets for the deodorant's category.



Image source: <https://www.dotactiv.com/hs-fs/hubfs/Category-based%20clustering.png?width=1038&height=557&name=Category-based%20clustering.png>

## 3) Clustering in Clinical Care / Disease Management

Healthcare and Clinical Science is again one of those areas that are full of opportunities for clustering that are indeed very impactful in the field. One such example is research published by Komaru & Yoshida et al. 2020, where they collected demographics and laboratory data for 101 patients and then segmented them into 3 clusters.

Each cluster was represented by different conditions. For example, cluster 1 has patients with Low WBC & CRP. Cluster 2 has patients with High BMP & Serum, and Cluster 3 has patients with Low Serum. Each cluster represents a different survival trajectory given the 1-year mortality after hemodialysis.



# Hierarchical clustering analysis for predicting 1-year mortality after starting hemodialysis

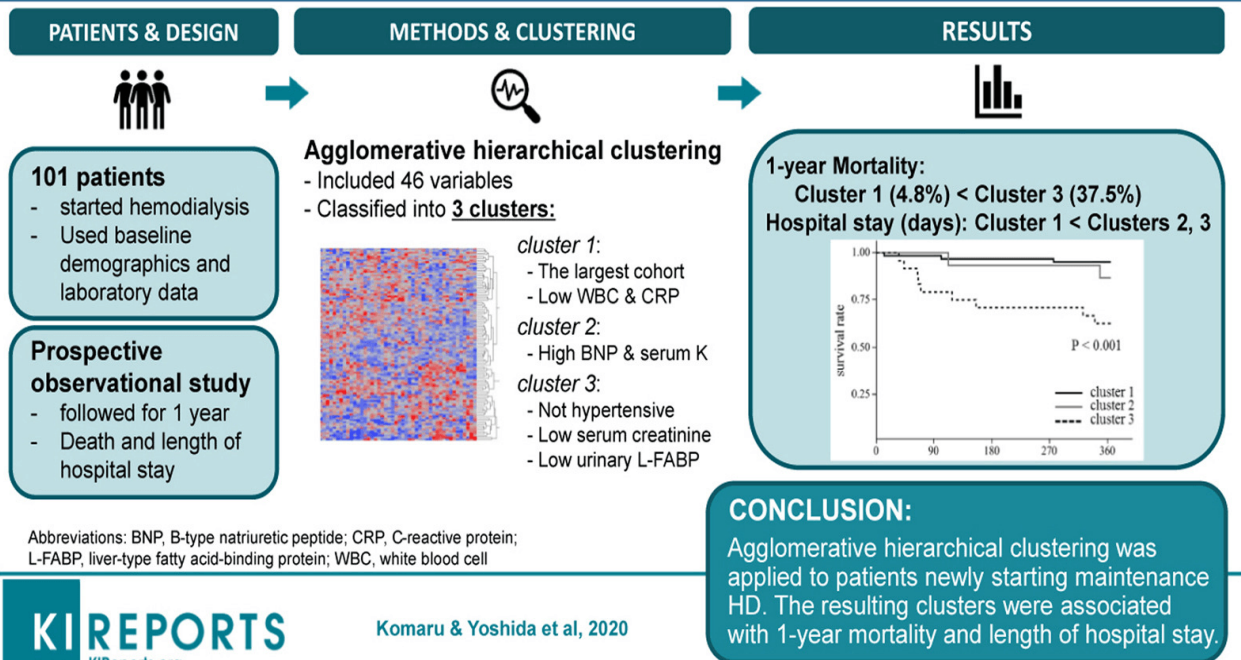
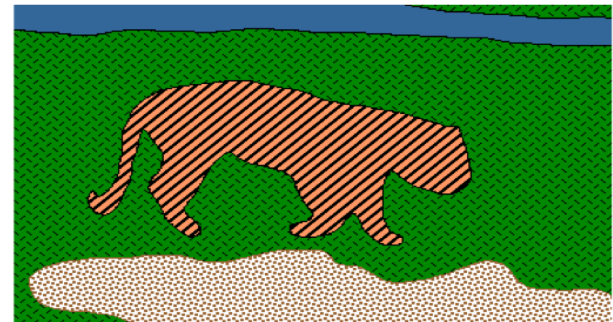


Image source: [https://els-jbs-prod-cdn.jbs.elsevierhealth.com/cms/attachment/da4cb0c9-0a86-4702-8a78-80ffffcf1f9c/fx1\\_lrg.jpg](https://els-jbs-prod-cdn.jbs.elsevierhealth.com/cms/attachment/da4cb0c9-0a86-4702-8a78-80ffffcf1f9c/fx1_lrg.jpg)

## 4) Image Segmentation

Image segmentation is the classification of an image into different groups. Much research has been done in the area of image segmentation using clustering. This type of clustering is useful if you want to isolate objects in an image to analyze each object individually to check what it is.

In the example below, the left-hand side represents the original image, and the right-hand side is the result of the clustering algorithm. You can clearly see there are 4 clusters which are 4 different objects in the image determined based on the pixels (tiger, grass, water, and sand).



## Comparison of Different Clustering Algorithms

There are several unsupervised clustering algorithms implemented in *scikit-learn* - a popular machine learning library in Python. There are fundamental underlying differences in how each algorithm determines and assigns clusters in the dataset.

The underlying differences in the mathematical modality of these algorithms boil down to four aspects on which we can compare and contrast these algorithms:

- ❖ **Parameters required for the model** - This aspect refers to the specific settings or values that you need to provide to a clustering algorithm to create a clustering model. Different clustering algorithms may require different types and numbers of parameters. These parameters often influence how the algorithm identifies and groups data points into clusters.
- ❖ **Scalability** - Scalability in the context of clustering algorithms refers to how well the algorithm can handle larger datasets or a larger number of data points. It's a measure of the algorithm's efficiency and performance as data size increases. Some clustering algorithms are highly scalable and can efficiently process large datasets, while others may become computationally expensive or impractical when faced with substantial amounts of data.
- ❖ **Use-cases** - Use-cases refer to the specific applications or scenarios where a clustering algorithm is most suitable. Different clustering algorithms have strengths and weaknesses that make them more or less appropriate for certain tasks. For example, K-Means is often used for customer segmentation in marketing, while hierarchical clustering is useful for constructing hierarchical taxonomies. Understanding the use-cases helps choose the right algorithm for a given problem.
- ❖ **Geometry, i.e., metric used for calculation of distances** - The geometry aspect is related to the distance metric or measure used to calculate the similarity or dissimilarity between data points. In clustering, the choice of distance metric can significantly affect the clustering results. Common distance metrics include Euclidean distance (measuring straight-line distance), Manhattan distance (sum of absolute differences), and others. The choice of distance metric should match the data's characteristics and the problem domain.

## 5 Essential Clustering Algorithms

### 1) K-Means

K-Means clustering algorithm is easily the most popular and widely used algorithm for clustering tasks. It is primarily because of the intuition and the ease of implementation. It is a centroid-based algorithm where the user must define the required number of clusters it wants to create.

This normally comes from business use-case or by trying different values for the number of clusters and then evaluating the output.

K-Means clustering is an iterative algorithm that creates non-overlapping clusters meaning each instance in your dataset can only belong to one cluster exclusively. The

easiest way to get the intuition of the K-Means algorithm is to understand the steps along with the example diagram below.

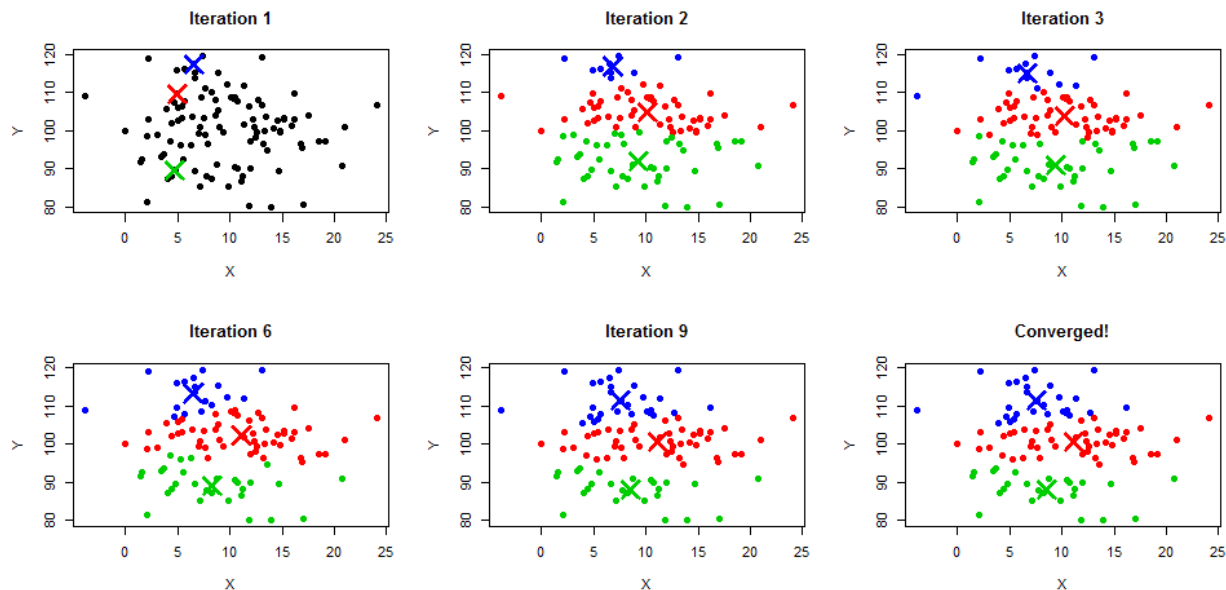


Image Source: <https://www.learnbymarketing.com/wp-content/uploads/2015/01/method-k-means-steps-example.png>

- 1) User specifies the number of clusters.
- 2) Initialize centroids randomly based on the number of clusters. In the diagram below in Iteration 1, notice three centroids are initialized randomly in blue, red, and green colors.
- 3) Calculate the distance between data points and each centroid and assign each data point to the nearest centroids.
- 4) Recalculate the mean of the centroid based on all the assigned data points, and this will change the position of the centroid, as you can see in Iteration 2 - 9, until it finally converges.
- 5) Iteration keeps on going until there is no change to the centroid's mean or a parameter `max_iter` is reached, which is the maximum number of the iterations as defined by the user during training. In scikit-learn, `max_iter` by default is set to 300.

## 2) MeanShift

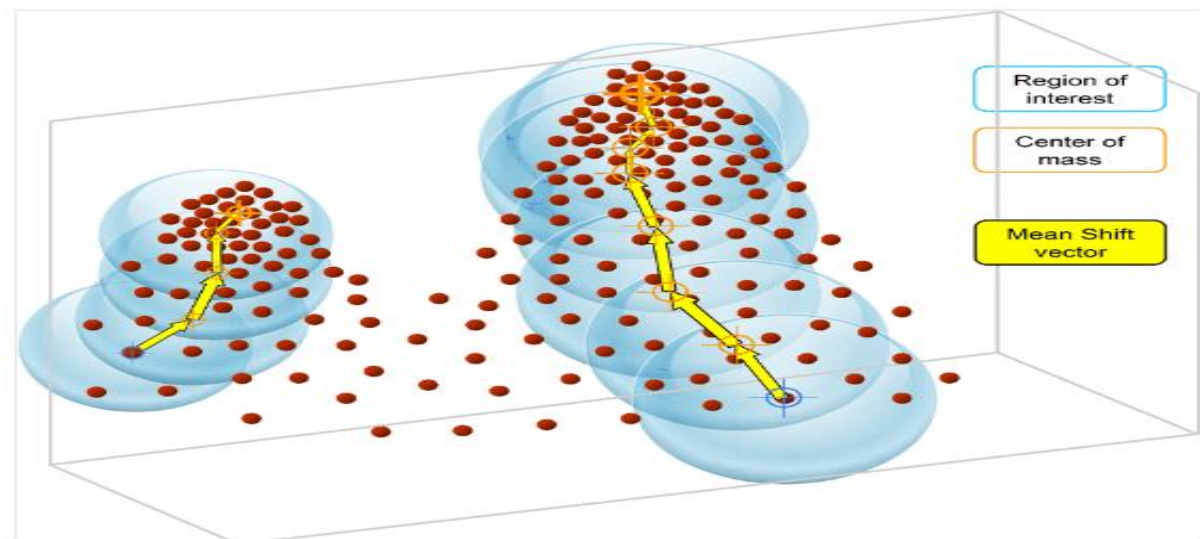
Unlike the K-Means algorithm, MeanShift algorithm does not require specifying the number of clusters. The algorithm itself automatically determines the number of clusters, which is a pretty big advantage over K-Means if you are not sure about patterns in data. MeanShift is also based on centroids and iteratively assigns each data point to clusters. The most common use case for MeanShift clustering is image segmentation tasks. The MeanShift algorithm is based on kernel density estimation. Similar to the K-Means algorithm, MeanShift algorithms iteratively assigns each data point towards the closest cluster centroid which are initialized randomly and each point are iteratively moved in



the space based on where the most points are i.e. the Mode (mode is the highest density of data points in the region, in the context of the MeanShift).

This is why the MeanShift algorithm is also known as the Mode-seeking algorithm. The steps of the MeanShift algorithm are as follows:

- ✚ Pick any random point, and create a window around that random point.
- ✚ Calculate the mean of all the points inside this window.
- ✚ Shift the window by following the direction of mode.
- ✚ Repeat the steps till convergence.



*Image*

Source: <https://www.researchgate.net/publication/326242239/figure/fig3/AS:645578044231681@1530929208053/Intuitive-description-of-the-mean-shift-procedure-find-the-densest-regions-in-the.png>

### **3) DBSCAN**

DBSCAN or **D**ensity-**B**ased **S**patial Clustering of Applications with Noise is an unsupervised clustering algorithm that works on the premise that clusters are dense spaces in the region separated by lower-density regions.

The biggest advantage of this algorithm over K-Means and MeanShift is that it is robust to outliers meaning it will not include outliers data points in any cluster.

*DBSCAN algorithms require only two parameters from the user:*

- ✚ The radius of the circle to be created around each data point, also known as 'epsilon'
- ✚ minPoints which defines the minimum number of data points required inside that circle for that data point to be classified as a Core point.

Every data point is surrounded by a circle with a radius of epsilon, and DBSCAN identifies them as being either a Core point, Border point, or Noise point. A data point is considered to be a Core point if the circle that surrounds it has a minimum number of points specified by minPoints parameter.

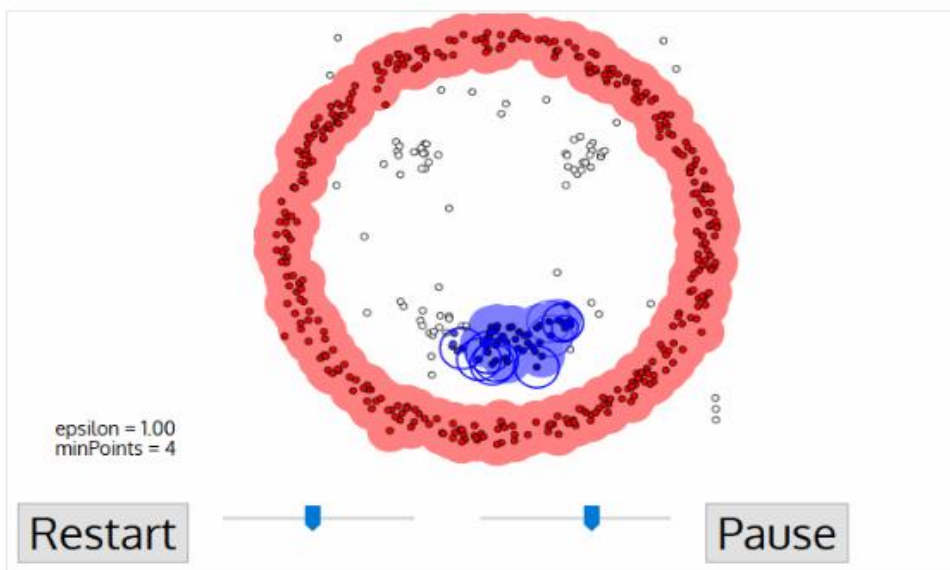
It is considered a Border Point if the number of points is lower than the minimum required, and it is considered Noise if there are no additional data points located within an epsilon radius of any data point. Noise data points are not categorized in any cluster (basically, they are outliers).

*Some of the common use-cases for DBSCAN clustering algorithm are:*

- ✚ It performs great at separating clusters of high density versus low density;
- ✚ It works great on non-linear datasets; and
- ✚ It can be used for anomaly detection as it separates out the noise points and do not assign them to any cluster.

*Comparing DBSCAN with K-Means algorithms, the most common differences are:*

- ✚ K-Means algorithm cluster all the instances in the datasets whereas DBSCAN doesn't assign noise points (outliers) to a valid cluster
- ✚ K-Means has difficulty with non-global clusters whereas DBSCAN can handle that smoothly
- ✚ K-Means algorithm makes assumptions that all data points in the dataset come from a gaussian distribution whereas DBSCAN makes no assumption about the data.



*Image Source: [https://miro.medium.com/proxy/1\\*tc8UF-h0nQqUfLC8-0uInQ.gif](https://miro.medium.com/proxy/1*tc8UF-h0nQqUfLC8-0uInQ.gif)*

#### 4) Hierarchical Clustering

Hierarchical clustering is a method of clustering that builds a hierarchy of clusters. There are two types of this method.

- ✦ **Agglomerative:** This is a bottom-up approach where each observation is treated as its own cluster in the beginning and as we move from bottom to top, each observation is merged into pairs, and pairs are merged into clusters.
- ✦ **Divisive:** This is a "top-down" approach: all observations start in one cluster, and splits are performed recursively as we move from top to bottom.

When it comes to analyzing data from social networks, hierarchical clustering is by far the most common and popular method of clustering. The nodes (branches) in the graph are compared to each other depending on the degree of similarity that exists between them. By linking together smaller groups of nodes that are related to one another, larger groupings may be created.

The biggest advantage of hierarchical clustering is that it is easy to understand and implement. Usually, the output of this clustering method is analyzed in an image such as below. It is called a Dendrogram.

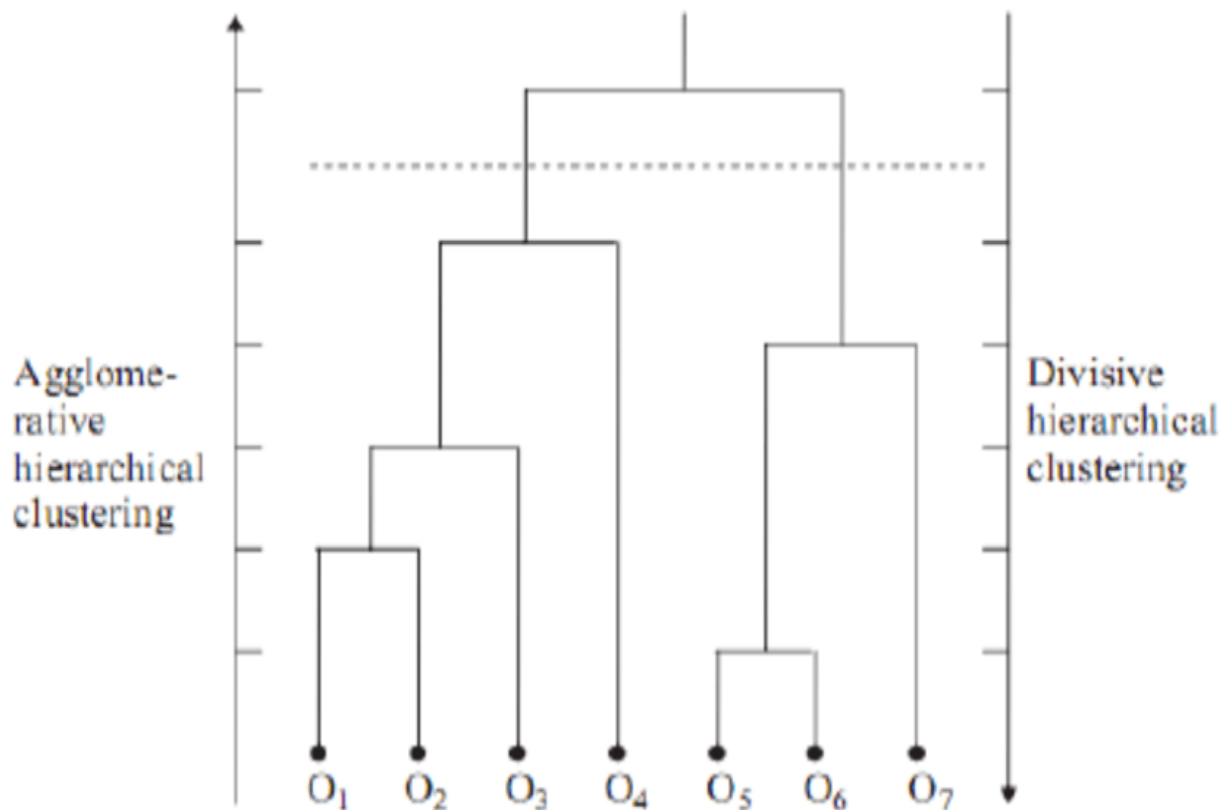


Image Source: <https://www.researchgate.net/profile/Rahmat-Widia-Sembiring/publication/48194320/figure/fig1/AS:307395533262848@1450300214331/Example-of-a-dendrogram-from-hierarchical-clustering.png>

## 5) BIRCH

BIRCH stands for Balanced Iterative Hierarchical Based Clustering. It is used on very large datasets where K-Means cannot practically scale. BIRCH algorithm divides large data into small clusters and tries to retain the maximum amount of information possible. Smaller groups are then clustered for a final output instead of clustering the large datasets directly.

BIRCH is often used to supplement other clustering algorithms by generating a summary of the information that the other clustering algorithms can utilize. Users have to define the number of clusters for training the BIRCH algorithm, similar to how we define it in K-Means.

One of the benefits of using BIRCH is that it can progressively and dynamically cluster multi-dimensional data points. This is done to create the highest quality clusters under a given memory and time constraints. In most cases, BIRCH just needs to do one search across the database, which makes BIRCH scalable.

The most common use-case of BIRCH clustering algorithm is that it is a memory-efficient alternative to KMeans that can be used to cluster large datasets that cannot be handled through KMeans due to memory or compute limitations.

*A Table comparing the clustering algorithms K-Means, MeanShift, DBSCAN, Hierarchical Clustering, and BIRCH based on the four aspects: parameters required for the model, scalability, use-cases, and geometry (distance metric).*

<b>Clustering Algorithm</b>	<b>Parameters required for the model</b>	<b>Scalability</b>	<b>Use-cases</b>	<b>Geometry (distance metric)</b>
<b>K-Means</b>	Number of clusters (K)	Good scalability for large datasets	Segmentation, anomaly detection, customer segmentation	Euclidean distance
<b>Mean-Shift</b>	Bandwidth parameter	Good scalability for large datasets	Image segmentation, anomaly detection, tracking	Euclidean distance
<b>DBSCAN</b>	Epsilon (radius of neighborhood) and MinPts (minimum number of points in a cluster)	Good scalability for large datasets	Anomaly detection, clustering data with non-globular shapes, clustering data with noise	Euclidean distance

<b>Hierarchical Clustering</b>	None	Poor scalability for large datasets	Exploratory data analysis, clustering data with unknown number of clusters	Euclidean distance or other distance metrics
<b>BIRCH</b>	Number of clusters (K) and threshold parameter	Good scalability for very large datasets	Data preprocessing, clustering large datasets, clustering data with unknown number of clusters	Euclidean distance or other distance metrics

### **Summary:**

- ✚ Parameters required for the model: K-Means and BIRCH require the number of clusters (K) to be specified upfront. Mean-Shift requires a bandwidth parameter, and DBSCAN requires two parameters: epsilon (radius of neighborhood) and MinPts (minimum number of points in a cluster). Hierarchical Clustering does not require any parameters to be specified.
- ✚ Scalability: K-Means, Mean-Shift, and DBSCAN all scale well to large datasets. Hierarchical Clustering and BIRCH are the most scalable clustering algorithms, as they can handle very large datasets.
- ✚ Use-cases: K-Means is commonly used for segmentation, anomaly detection, and customer segmentation. Mean-Shift is commonly used for image segmentation, anomaly detection, and tracking. DBSCAN is commonly used for anomaly detection, clustering data with non-globular shapes, and clustering data with noise. Hierarchical Clustering is commonly used for exploratory data analysis and clustering data with unknown number of clusters. BIRCH is commonly used for data preprocessing, clustering large datasets, and clustering data with unknown number of clusters.
- ✚ Geometry (distance metric): All of the clustering algorithms listed above can use the Euclidean distance metric. However, some algorithms, such as DBSCAN and BIRCH, can also use other distance metrics.

### **Recommendation:**

The best clustering algorithm to use depends on the specific data set and use case. However, as a general rule of thumb, K-Means, Mean-Shift, and DBSCAN are good choices for most applications. Hierarchical Clustering and BIRCH are good choices for applications where the number of clusters is unknown or where the data set is very large.



## **IMPLEMENTATION IN GOOGLE COLAB:**

**EXAMPLE 1: A python program to cluster image in google colab using unsupervised learning:**

### **INPUT IMAGE:**



### **SOURCE CODE:**

```
from google.colab import files
import cv2
import matplotlib.pyplot as plt
import numpy as np
from sklearn.cluster import KMeans
import matplotlib.patches as mpatches
#Upload the image
uploaded = files.upload()
image = cv2.imdecode(np.frombuffer(uploaded[next(iter(uploaded))], np.uint8), -1)
# Load and preprocess the image
#image = cv2.imread('WAINAINA.jpg')
#image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
height, width, _ = image.shape
reshaped_image = image.reshape(-1, 3)
# Perform K-Means clustering
k = 5 # Number of clusters
kmeans = KMeans(n_clusters=k, random_state=0).fit(reshaped_image)
labels = kmeans.labels_
centers = kmeans.cluster_centers_
# Create a color-coded segmentation map
segmented_image = centers[labels].reshape(height, width, 3).astype('uint8')
# Create a title for the output
title = f"Image Segmentation with {k} Clusters"
```

```

# Define segment labels based on colors
segment_labels = [f"Segment {i+1}" for i in range(k)]
# Visualize the segmented image with title and a legend for colors
plt.figure(figsize=(10, 6))
# Display the segmented image
plt.imshow(segmented_image)
plt.axis('off')
# Add a title using plt.title
plt.title(title, fontsize=16, fontweight='bold')
# Create a legend for colors
legend_patches = [mpatches.Patch(color=centers[i] / 255, label=segment_labels[i]) for i
in range(k)]
plt.legend(handles=legend_patches, loc='lower right')
plt.show()


```

Line	Code	Explanation
1	from google.colab import files	Import the files module from Google Colab for uploading files.
2	import cv2	Import the OpenCV library for image processing.
3	import matplotlib.pyplot as plt	Import the matplotlib library for data visualization.
4	import numpy as np	Import the NumPy library for numerical operations.
5	from sklearn.cluster import KMeans	Import the K-Means clustering algorithm from scikit-learn.
6	import matplotlib.patches as mpatches	Import patches module from matplotlib for creating legend patches.
8	uploaded = files.upload()	Upload an image file using Google Colab's file upload feature and store it in the uploaded variable.
10	image = cv2.imdecode(np.frombuffer(uploaded [next(iter(uploaded))], np.uint8), -1)	Decode the uploaded image data using OpenCV and store it in the image variable.
13	height, width, _ = image.shape	Get the height and width dimensions of the image.
14	reshaped_image = image.reshape(-1, 3)	Reshape the image to have a single column of RGB values.
18	k = 5	Define the number of clusters for K-Means clustering.
21	kmeans = KMeans(n_clusters=k, random_state=0).fit(reshaped_image)	Create a K-Means clustering model with k clusters and fit it to the reshaped image.
25	labels = kmeans.labels_	Get the cluster labels assigned to each data point.
26	centers = kmeans.cluster_centers_	Get the cluster centers (average colors) for each cluster.
31	segmented_image = centers[labels].reshape(height, width, 3).astype('uint8')	Create a segmented image by assigning each pixel its cluster's center color.
34	title = f"Image Segmentation with {k} Clusters"	Create a title for the output image specifying the number of clusters.

37	<code>segment_labels = [f"Segment {i+1}" for i in range(k)]</code>	Create labels for each segment based on cluster numbers.
40	<code>plt.figure(figsize=(10, 6))</code>	Create a figure for displaying the segmented image.
43	<code>plt.imshow(segmented_image)</code>	Display the segmented image.
44	<code>plt.axis('off')</code>	Turn off the axis to remove the axis labels.
49	<code>plt.title(title, fontsize=16, fontweight='bold')</code>	Add a title to the plot with a specified font size and style.
52	<code>legend_patches = [mpatches.Patch(color=centers[i] / 255, label=segment_labels[i]) for i in range(k)]</code>	Create legend patches for each segment with their respective colors.
55	<code>plt.legend(handles=legend_patches, loc='lower right')</code>	Add a legend to the plot with legend patches and position it in the lower right corner.
56	<code>plt.show()</code>	Display the final plot with the segmented image, title, and legend.

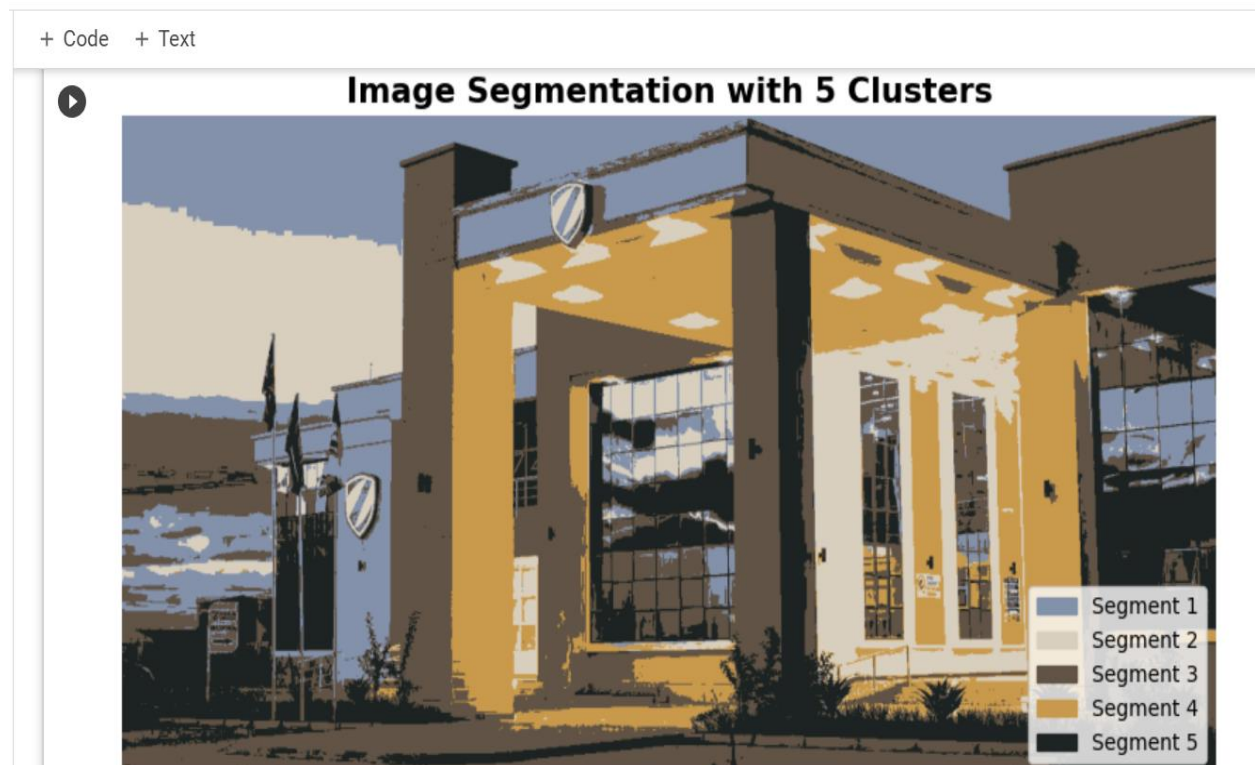
## OUTPUT:

+ Code
+ Text



Choose Files
MANGU.jpg

- MANGU.jpg**(image/jpeg) - 408198 bytes, last modified: 7/9/2023 - 100% done  
Saving MANGU.jpg to MANGU (11).jpg  
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/\_kmeans.py:870:



**EXAMPLE 2: Python code that demonstrates a simple method using K-Means clustering to detect hidden messages within an image.**

**INPUT IMAGES:**



**SOURCE CODE:**

*This code takes an image, performs K-Means clustering to find two dominant colors, and then determines a threshold to decode a hidden message embedded in the image. The decoded message is printed at the end.*

```
import cv2
import numpy as np
# Load the image
image = cv2.imread('MANGU.jpg')
if image is None:
    print("Image not found")
else:
    # Convert the image to grayscale for simplicity
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # Reshape the image into a flat 1D array for clustering
    reshaped_image = gray_image.reshape(-1, 1)
    # Perform K-Means clustering to identify the two dominant colors
    kmeans = KMeans(n_clusters=2, random_state=0)
    kmeans.fit(reshaped_image)
    # Find the two cluster centers
    cluster_centers = kmeans.cluster_centers_.astype(int)
    # Sort cluster centers by color intensity
    sorted_cluster_centers = sorted(cluster_centers, key=lambda x: sum(x))
    # Define a threshold value to distinguish between the two clusters
    threshold = (sorted_cluster_centers[0][0] + sorted_cluster_centers[1][0]) // 2
    # Extract the hidden message from pixel values
    decoded_bits = [1 if pixel > threshold else 0 for pixel in reshaped_image]
    # Reconstruct the message by grouping bits and converting to characters
    message_bits = [decoded_bits[i:i+8] for i in range(0, len(decoded_bits), 8)]
    decoded_message = "".join([chr(int("".join(map(str, bits)), 2)) for bits in message_bits])
    print("Decoded Hidden Message:")
    print(decoded_message)
```



Line	Code	Explanation
1	<code>image = cv2.imread('MANGU.jpg')</code>	Load an image from the file 'MANGU.jpg' using OpenCV's imread function.
2	<code>if image is None:</code>	Check if the image is loaded successfully. If not, print an error message.
3	<code>gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)</code>	Convert the loaded image to grayscale to simplify processing.
4	<code>reshaped_image = gray_image.reshape(-1, 1)</code>	Reshape the grayscale image into a 1D array for clustering.
5	<code>kmeans = KMeans(n_clusters=2, random_state=0)</code>	Create a K-Means clustering object with 2 clusters and a random seed.
6	<code>kmeans.fit(reshaped_image)</code>	Apply K-Means clustering to the reshaped image to identify two dominant clusters.
7	<code>cluster_centers = kmeans.cluster_centers_.astype(int)</code>	Obtain the cluster centers as integer values.
8	<code>sorted_cluster_centers = sorted(cluster_centers, key=lambda x: sum(x))</code>	Sort the cluster centers based on their color intensity.
9	<code>threshold = (sorted_cluster_centers[0][0] + sorted_cluster_centers[1][0]) // 2</code>	Calculate a threshold value to distinguish between the two clusters.
10	<code>decoded_bits = [1 if pixel &gt; threshold else 0 for pixel in reshaped_image]</code>	Iterate over pixel values and determine if they belong to one of the clusters based on the threshold.
11	<code>message_bits = [decoded_bits[i:i+8] for i in range(0, len(decoded_bits), 8)]</code>	Group the decoded bits into 8-bit segments for character conversion.
12	<code>decoded_message = ''.join([chr(int(''.join(map(str, bits)), 2)) for bits in message_bits])</code>	Reconstruct the hidden message by converting grouped bits to characters.
13	<code>print("Decoded Hidden Message:")</code>	Display a message to indicate that the hidden message is being printed.
14	<code>print(decoded_message)</code>	Print the decoded hidden message to the console.

## OUTPUT:

The screenshot shows the output of the code in a Jupyter Notebook. The output is a large grid of binary digits (0s and 1s) arranged in a pattern that resembles a hidden message. The grid is composed of many rows and columns of characters, with some rows containing more 1s than others, creating a visual representation of the decoded data.



**EXAMPLE 3: A python program to show how Clustering can be applied to clinical care and disease management to identify patient groups with similar characteristics.**

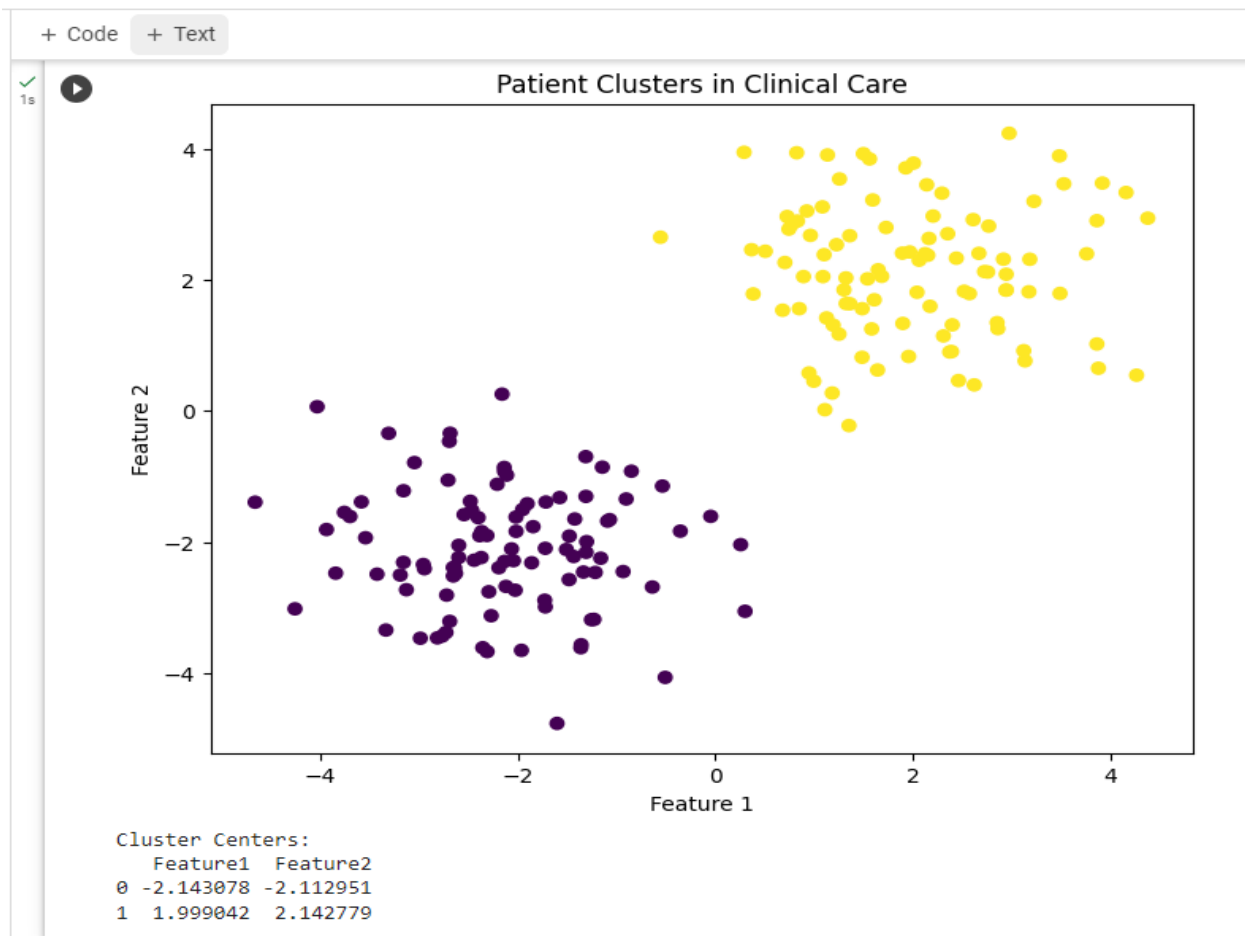
**SOURCE CODE:**

```
import numpy as np
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
# Generate synthetic patient data
np.random.seed(0)
n_samples = 200
n_features = 2
# Create two distinct patient clusters
cluster1 = np.random.randn(n_samples // 2, n_features) + np.array([2, 2])
cluster2 = np.random.randn(n_samples // 2, n_features) + np.array([-2, -2])
patient_data = np.vstack([cluster1, cluster2])
# Create a DataFrame for visualization
data_df = pd.DataFrame(patient_data, columns=['Feature1', 'Feature2'])
# Standardize the data
scaler = StandardScaler()
patient_data_std = scaler.fit_transform(patient_data)
# Apply K-Means clustering
n_clusters = 2
kmeans = KMeans(n_clusters=n_clusters, random_state=0)
data_df['Cluster'] = kmeans.fit_predict(patient_data_std)
# Visualize the patient clusters
plt.figure(figsize=(8, 6))
plt.scatter(data_df['Feature1'], data_df['Feature2'], c=data_df['Cluster'], cmap='viridis')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Patient Clusters in Clinical Care')
plt.show()
# Analyze cluster characteristics (e.g., for disease management)
cluster_centers = scaler.inverse_transform(kmeans.cluster_centers_)
print('Cluster Centers:')
print(pd.DataFrame(cluster_centers, columns=['Feature1', 'Feature2']))
```

Line	Code	Explanation
1	import numpy as np	Imports the NumPy library as 'np' for numerical and array operations.
2	import pandas as pd	Imports the Pandas library as 'pd' for data manipulation and analysis.
3	from sklearn.cluster import KMeans	Imports the KMeans clustering algorithm from the scikit-learn library.

4	<code>from sklearn.preprocessing import StandardScaler</code>	Imports the StandardScaler from scikit-learn for data standardization.
5	<code>import matplotlib.pyplot as plt</code>	Imports the Matplotlib library for data visualization.
7	<code>np.random.seed(0)</code>	Sets a random seed to ensure reproducibility of the random data generation.
8	<code>n_samples = 200</code>	Defines the number of synthetic patient samples to generate.
9	<code>n_features = 2</code>	Defines the number of features for each patient.
10	<code>cluster1 = np.random.randn(n_samples // 2, n_features) + np.array([2, 2])</code>	Generates the first cluster of patient data with normal distribution and shifts it to [2, 2].
11	<code>cluster2 = np.random.randn(n_samples // 2, n_features) + np.array([-2, -2])</code>	Generates the second cluster of patient data with normal distribution and shifts it to [-2, -2].
12	<code>patient_data = np.vstack([cluster1, cluster2])</code>	Stacks the two clusters vertically to create the synthetic patient data.
14	<code>data_df = pd.DataFrame(patient_data, columns=['Feature1', 'Feature2'])</code>	Creates a Pandas DataFrame for the synthetic patient data with column names 'Feature1' and 'Feature2'.
16	<code>scaler = StandardScaler()</code>	Initializes a StandardScaler to standardize the data.
17	<code>patient_data_std = scaler.fit_transform(patient_data)</code>	Standardizes the synthetic patient data.
19	<code>n_clusters = 2</code>	Defines the number of clusters for the K-Means algorithm.
20	<code>kmeans = KMeans(n_clusters=n_clusters, random_state=0)</code>	Initializes a K-Means clustering model with 2 clusters and a random seed of 0.
21	<code>data_df['Cluster'] = kmeans.fit_predict(patient_data_std)</code>	Adds a 'Cluster' column to the DataFrame to store cluster labels assigned by K-Means.
23	<code>plt.figure(figsize=(8, 6))</code>	Sets the figure size for the visualization.
24	<code>plt.scatter(data_df['Feature1'], data_df['Feature2'], c=data_df['Cluster'], cmap='viridis')</code>	Creates a scatter plot of patient data points, coloring them by cluster.
25	<code>plt.xlabel('Feature 1')</code>	Sets the x-axis label.
26	<code>plt.ylabel('Feature 2')</code>	Sets the y-axis label.
27	<code>plt.title('Patient Clusters in Clinical Care')</code>	Sets the title for the plot.
28	<code>plt.show()</code>	Displays the plot.
30	<code>cluster_centers = scaler.inverse_transform(kmeans.cluster_centers_)</code>	Transforms the standardized cluster centers back to the original scale.
31	<code>print('Cluster Centers:')</code>	Prints a message indicating the cluster centers.
32	<code>print(pd.DataFrame(cluster_centers, columns=['Feature1', 'Feature2']))</code>	Displays the cluster centers as a Pandas DataFrame.

## OUTPUT:



*In the provided output:*

- Cluster Centers is the title indicating that the following information represents the cluster centers.
- Feature1 and Feature2 are the feature names or dimensions of the data.
- The table below the column names displays the cluster centers for the two clusters.

*Here's the meaning of the table:*

- Cluster 0 (Row 0) represents one cluster's center. The values in this row correspond to the feature values (Feature1 and Feature2) for the center of the first cluster.
- Cluster 1 (Row 1) represents the center of the second cluster. The values in this row correspond to the feature values (Feature1 and Feature2) for the center of the second cluster.

In this context, the cluster centers indicate the representative points for each cluster. These points are often used for cluster analysis and interpretation. The values of Feature1 and Feature2 represent the characteristics or attributes of the data points in each cluster.

**EXAMPLE 4: A Python program to perform K-Means clustering on a dataset containing information about insurance charges. The goal is to cluster individuals based on their specified features in the insurance.csv file and their corresponding insurance charges.**

<https://www.kaggle.com/datasets/mirichoi0218/insurance>

**SOURCE CODE:**

```
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler, LabelEncoder
import matplotlib.pyplot as plt
from google.colab import files
# Upload your dataset (CSV file)
uploaded = files.upload()
# Read the uploaded CSV file into a DataFrame
data = pd.read_csv("insurance.csv")
# Define the feature you want to cluster against 'charges'
feature_to_cluster = 'bmi' # Replace with the feature of your choice
# Convert 'smoker' and 'sex' columns to numerical values
label_encoder = LabelEncoder()
data['smoker'] = label_encoder.fit_transform(data['smoker'])
data['sex'] = label_encoder.fit_transform(data['sex'])
# Create a feature matrix X
X = data[['feature_to_cluster', 'charges']]
# Standardize the data
scaler = StandardScaler()
X_std = scaler.fit_transform(X)
# Apply K-Means clustering
n_clusters = 3 # You can adjust the number of clusters as needed
kmeans = KMeans(n_clusters=n_clusters, random_state=0)
data['Cluster'] = kmeans.fit_predict(X_std)
# Create a title for the output
title = f'Clusters of {feature_to_cluster} against Charges'
# Visualize the clusters with a legend
plt.figure(figsize=(8, 6))
for i in range(n_clusters):
    cluster_data = data[data['Cluster'] == i]
    plt.scatter(cluster_data['charges'], cluster_data[feature_to_cluster], c=plt.cm.viridis(i /
(n_clusters - 1)), label=f'Cluster {i+1}')
plt.xlabel('Charges')
plt.ylabel(feature_to_cluster)
plt.title(title, fontsize=16, fontweight='bold')
plt.legend(loc='lower right')
plt.show()
```

<i>Line</i>	<i>Code</i>	<i>Explanation</i>
1	import pandas as pd	Import the Pandas library for data handling and analysis.
2	import numpy as np	Import the NumPy library for numerical operations.
3	from sklearn.cluster import KMeans	Import the KMeans clustering algorithm from scikit-learn.
4	from sklearn.preprocessing import StandardScaler, LabelEncoder	Import StandardScaler and LabelEncoder from scikit-learn for data preprocessing.
5	import matplotlib.pyplot as plt	Import the Matplotlib library for data visualization.
6	from google.colab import files	Import the files module from Google Colab to allow file uploads.
8	uploaded = files.upload()	Allow the user to interactively upload a CSV dataset file using Google Colab.
11	for filename in uploaded.keys():	Iterate through the uploaded files (in case multiple files are uploaded).
12	data = pd.read_csv(filename)	Read the uploaded CSV file into a Pandas DataFrame named data.
18	feature_to_cluster = 'bmi'	Define the feature (column) from the dataset that will be used for clustering. In this case, 'bmi' is selected, but you can change it to any feature of your choice.
21	label_encoder = LabelEncoder()	Create a LabelEncoder object to convert categorical values (e.g., 'smoker' and 'sex') to numerical values.
22	data['smoker'] = label_encoder.fit_transform(data['smoker'])	Use the LabelEncoder to convert the 'smoker' column to numerical values in the DataFrame.
23	data['sex'] = label_encoder.fit_transform(data['sex'])	Use the LabelEncoder to convert the 'sex' column to numerical values in the DataFrame.
26	X = data[[feature_to_cluster, 'charges']]	Create a feature matrix X with the selected feature ('bmi') and the 'charges' feature.
29	scaler = StandardScaler()	Create a StandardScaler object to standardize the data.
30	X_std = scaler.fit_transform(X)	Standardize the data in the feature matrix X and store it in X_std.
33	n_clusters = 3	Define the number of clusters for the K-Means algorithm (in this case, 3 clusters).
34	kmeans = KMeans(n_clusters=n_clusters, random_state=0)	Create a KMeans clustering model with the specified number of clusters and a random seed.
35	data['Cluster'] = kmeans.fit_predict(X_std)	Assign cluster labels to the data based on the KMeans clustering. Store these labels in a new 'Cluster' column in the DataFrame.
39	title = f'Clusters of {feature_to_cluster} against Charges'	Create a title for the output visualization indicating the clustered feature.
42	plt.figure(figsize=(8, 6))	Create a Matplotlib figure with a specified size for plotting the clusters.
43	for i in range(n_clusters):	Iterate through the clusters to visualize them.



44	<code>cluster_data = data[data['Cluster'] == i]</code>	Filter the data for the current cluster i.
45	<code>plt.scatter(cluster_data['charges'], cluster_data[feature_to_cluster], c=plt.cm.viridis(i / (n_clusters - 1)), label=f'Cluster {i+1}')</code>	Create a scatter plot for the charges and selected feature of the current cluster, assigning different colors based on cluster index.
46	<code>plt.xlabel('Charges')</code>	Set the x-axis label of the plot to 'Charges'.
47	<code>plt.ylabel(feature_to_cluster)</code>	Set the y-axis label of the plot to the selected feature.
48	<code>plt.title(title, fontsize=16, fontweight='bold')</code>	Set the title of the plot to the previously defined title with specified font size and weight.
49	<code>plt.legend(loc='lower right')</code>	Add a legend to the plot at the 'lower right' location to distinguish between clusters.
50	<code>plt.show()</code>	Display the plot.

### OUTPUT:

