



# A Practical Guide on K-Means Clustering

Going beyond the theory and getting the best out of your K-Means clustering algorithm



Dhruvil Karani · Follow

Published in Towards Data Science · 8 min read · Mar 27, 2022

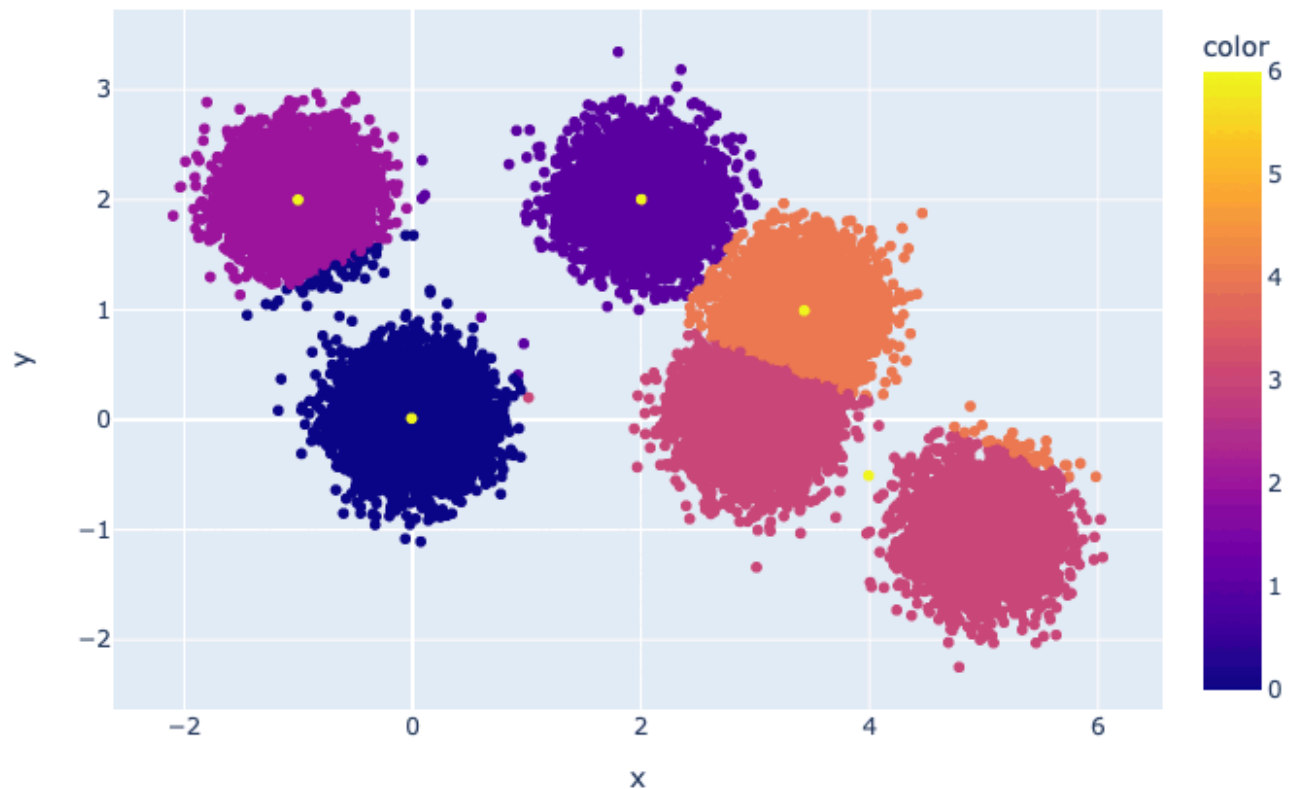


364



1





Iterations in KMeans. Original GIF.

## Table of contents

1. What is KMeans?
  - a. Python implementation
2. Things to know before using KMeans
  - a. K-Means cannot handle non-globular structure
  - b. K-Means is sensitive to outliers
  - c. Should you scale your data before using KMeans?
3. How do you measure the performance of your model?

*Note — The datasets used in the article are generated using sklearn's make\_blobs and make\_moons methods*

## What is KMeans?

K-Means divides the dataset into  $k$  (a hyper-parameter) clusters using an iterative optimization strategy. Each cluster is represented by a centre. A point belongs to a cluster whose centre is closest to it. For simplicity, assume that the centres are randomly initialized.

The goal of the model is to find clusters that minimize the sum of SSE over  $k$  clusters by shifting their centres. SSE or **Sum of Squared Errors** of a cluster is the sum of squared distances between its centre and its points.

Using calculus, one can prove that the best way to minimize the SSE is to move the cluster centre to the centroid (=average) of all points in the cluster. We re-assign the points to the clusters based on the same closest centre strategy with the updated centres.

Repeat the process until the centres no longer move too much and converge.

### K-Means summary

```
-----
X -> dataset of N points and M features ((N,M) matrix)
k -> Number of chosen centers (hyper-parameter)
centers -> randomly initialized k centers ((k,M) matrix)

for i in n_steps:
    - assign each point to a cluster based on the nearest center.
    points in jth cluster are denoted by X_j
    - updated_centers -> randomly initialized k centers ((k,M)
matrix)

    - for j in [1, 2, .. k]
        - updated_center[j] = mean(X_j)

    - if distance(updated_centers, centers) is small
        - exit
    - else
        - centers = updated_centers
        - continue
```

# Python Implementation

```

1  import numpy as np
2
3  class CustomKMeans:
4      def __init__(self, k, centroids, X):
5          '''
6          k: number of clusters
7          centroids: initial value of centres.
8                     np.array of shape (k, n_features)
9          X: dataset of m points and n_features.
10             array of shape (m, n_features)
11          '''
12          self.k = k
13          self.centroids = centroids
14          self.X = X
15          self.dim = X.shape[1]
16          self.index = {i:[] for i in range(len(self.centroids))}
17
18
19      def get_squared_distance_from_centroid(self, centroid):
20          '''
21          compute the squared distance of a
22          centroid from each point in X
23          '''
24          return np.sum(np.square(self.X-centroid), axis=1)
25
26
27      def get_cluster_variances(self):
28          '''
29          For each centroid,
30          get get_squared_distance_from_centroid
31          '''
32          intra_cluster_variances = []
33          for i in range(self.k):
34              centroid = self.centroids[i]
35              distance = self.get_squared_distance_from_centroid(centroid)
36              intra_cluster_variances.append(
37                  distance.reshape(-1,1)
38              )
39          return np.hstack(intra_cluster_variances)
40
41
42      def update_centroids(self, min_variance_clusters):
43          '''
44          Update the index based on closest centroid
45          for a point and then update the centroid

```

```

45         for a point and then update the centroid
46         based on this updated index
47         '''
48         for i in range(self.k):
49             self.index[i] = np.where(min_variance_clusters==i)[0]
50             self.centroids[i] = np.mean(
51                 self.X[self.index[i]],
52                 axis=0
53             )
54
55
56     def update(self):
57         '''
58         Update step in the KMeans algorithm.
59         '''
60         intra_cluster_variances = self.get_cluster_variances()
61         min_variance_clusters = np.argmin(
62             intra_cluster_variances,
63             axis=1
64         )
65         self.update_centroids(min_variance_clusters)
66
67
68
69     centroids = np.array([
70         [0, 0.5],
71         [1, 0]
72     ])
73     k = 2
74     X = np.random.randn(1000, 2)
75
76     model = CustomKMeans(k, centroids, X)

```

kmeans.py hosted with ❤ by GitHub

[view raw](#)

Naive K-Means Python implementation.Original code.

## Things to know before using KMeans

Most ML models have assumptions about the data they fit on. It is essential to check these before inferring anything from a trained model. For K-Means, here they are —

## K-Means cannot handle non-globular structure

Datasets can have any number of patterns that can be interpreted visually. The job of clustering algorithms is to be able to capture this information. Different algorithms use different strategies. Prototype-based algorithms like K-Means use centroid as a reference (=prototype) for each cluster. Density-based algorithms like DBSCAN use the density of data points to form clusters.

Consider the two datasets below —

### Globular (Globe like)

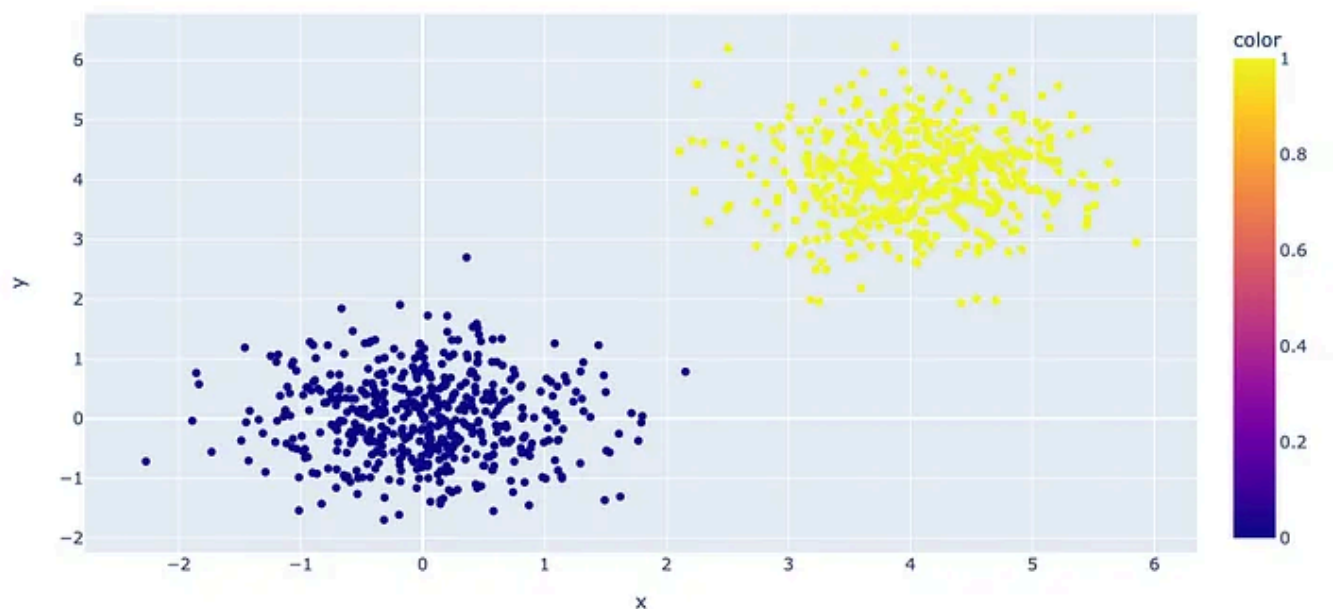


Fig. 1. Original Image.

### Non-Globular

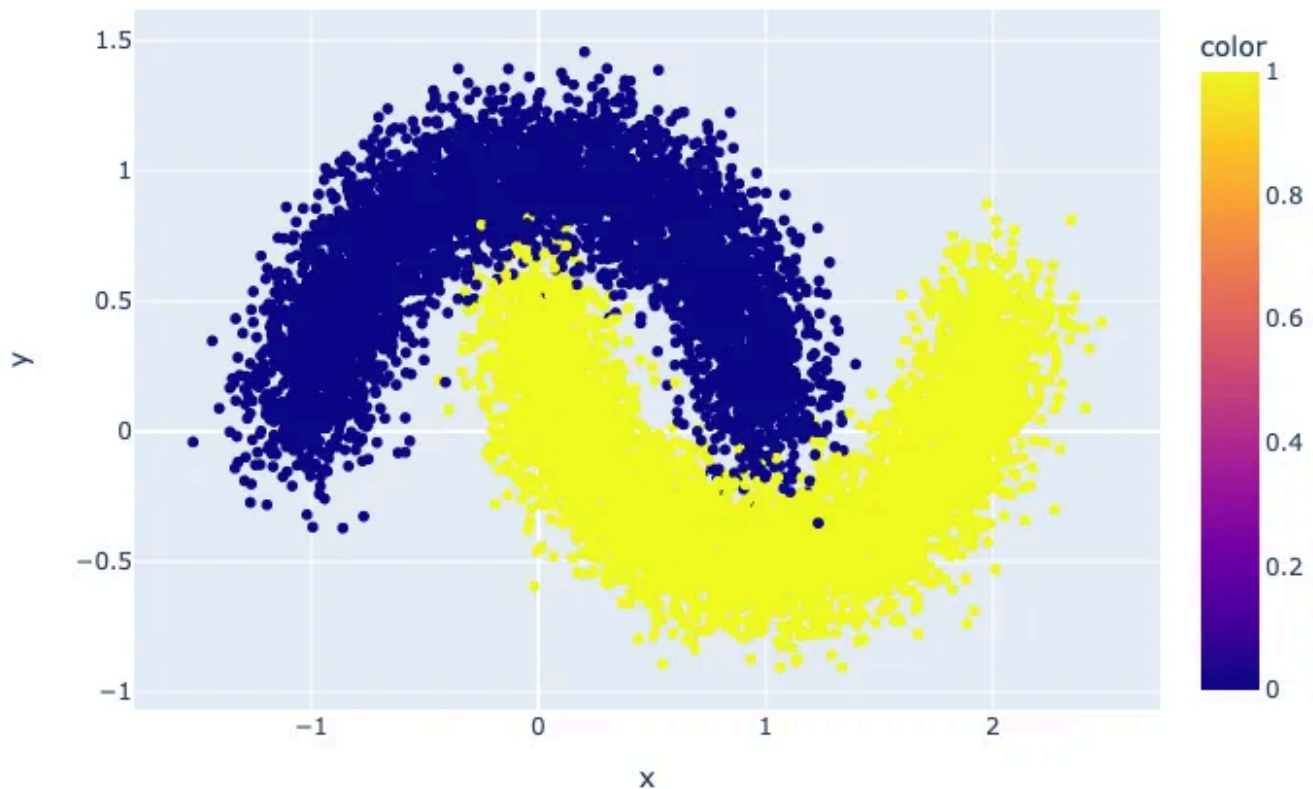


Fig. 2. Original Image.

K-Means would capture better structural semantics for the globular data. This is evident from how K-Means is fitted on data. We know that K-Means does the following

*Each cluster has a centroid. A point belongs to a cluster with the closest centroid.  
K-Means minimizes the sum of SSE by optimally iteratively moving the centroids.*

In a way, K-means works by creating a hard partition in the dataset, which act as the cluster boundaries. For example —



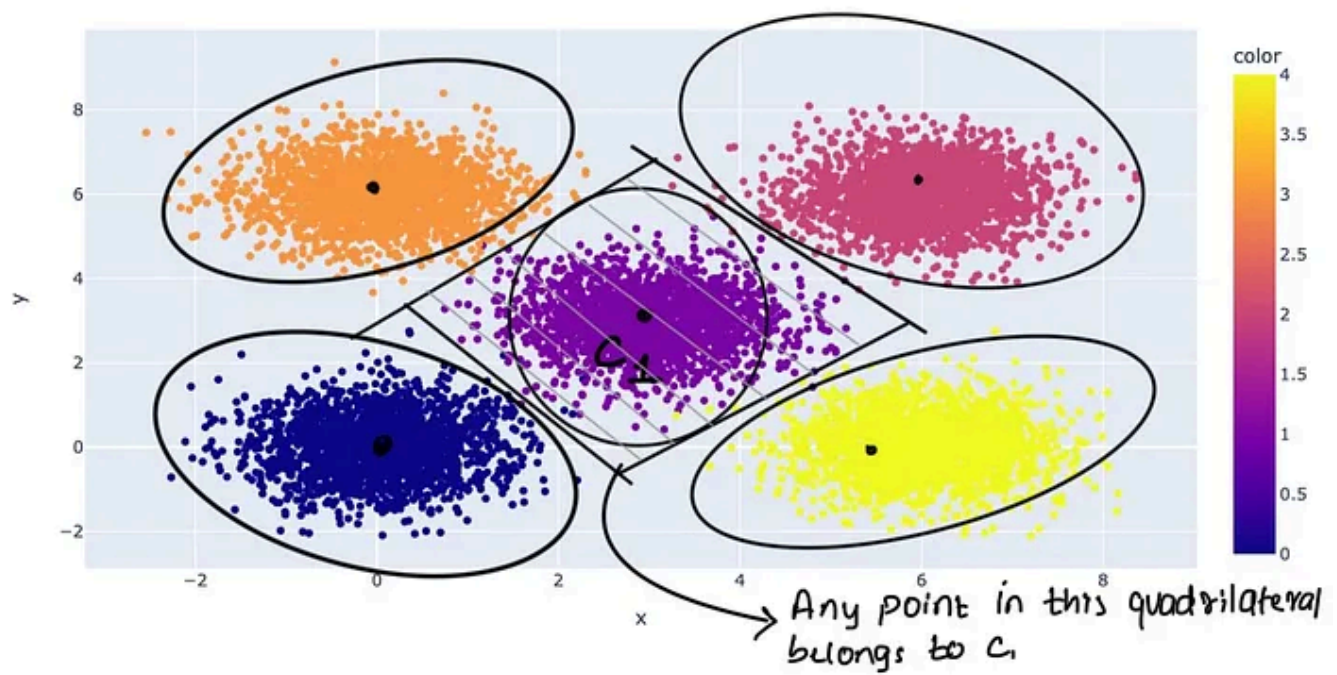


Fig. 3. The centre cluster is named cluster-one (C1). Original image.

In this example, with five clusters, the centre one (C1) is flanked by the rest four on all sides. The four-sided enclosure is the decision boundary for any point to be or not to be placed in cluster-1, based on the closest centroid.

For a new point, cluster assignment is solely dependent on the distance from the centroids. The position of the point does not matter as long as it is closest to a particular centroid. This works well in a globular structure. Recall that the boundary of a circle is the locus of point having a fixed distance from the centre.

For contrast, let's look at how K-Means works out in a non-globular setting.

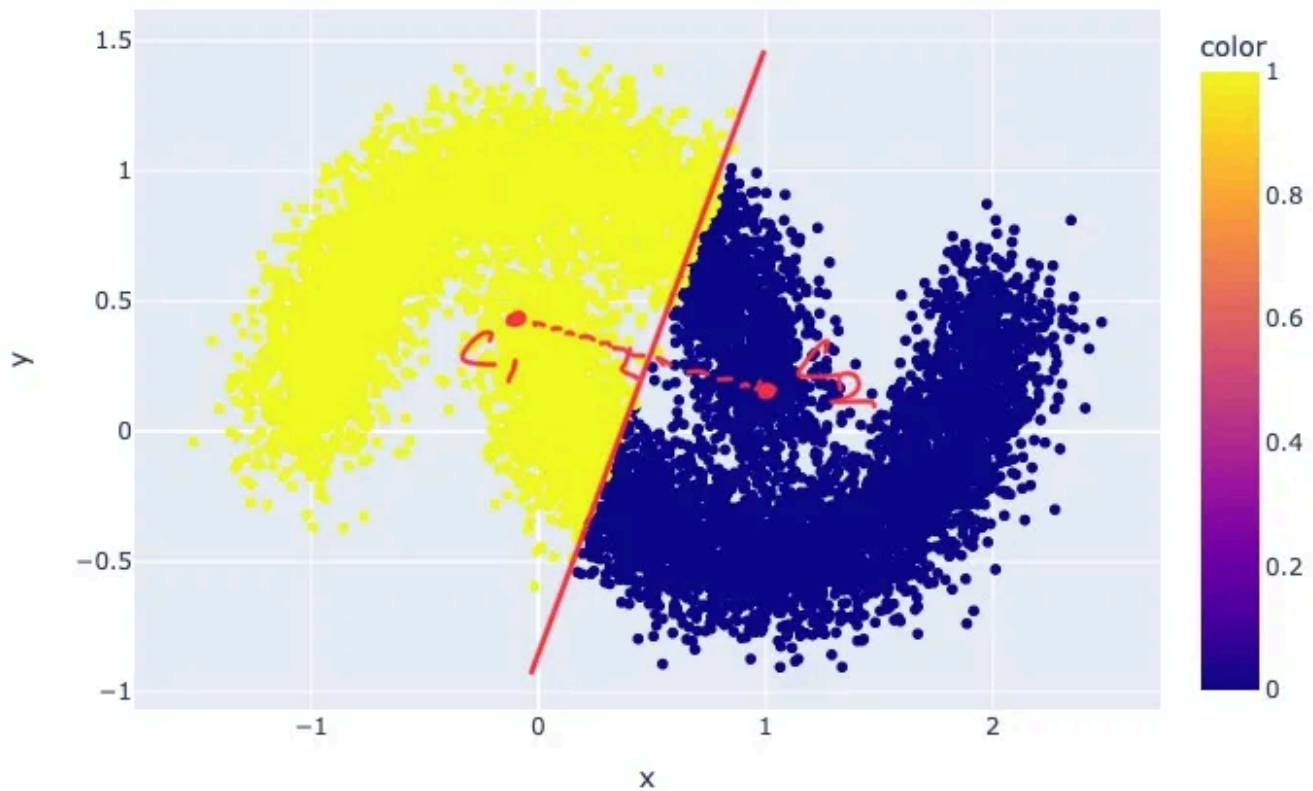


Fig. 4. Results of KMeans clustering on a non-globular dataset. As you can see that the clusters don't capture the actual semantics in the dataset. Original Image

### **KMeans is sensitive to outliers**

Since K-Means is a distance-based algorithm, it is susceptible to outliers. At every update step, the centroids are re-computed by averaging the points in a cluster. Averages, as we know, can be sensitive to outliers. For example

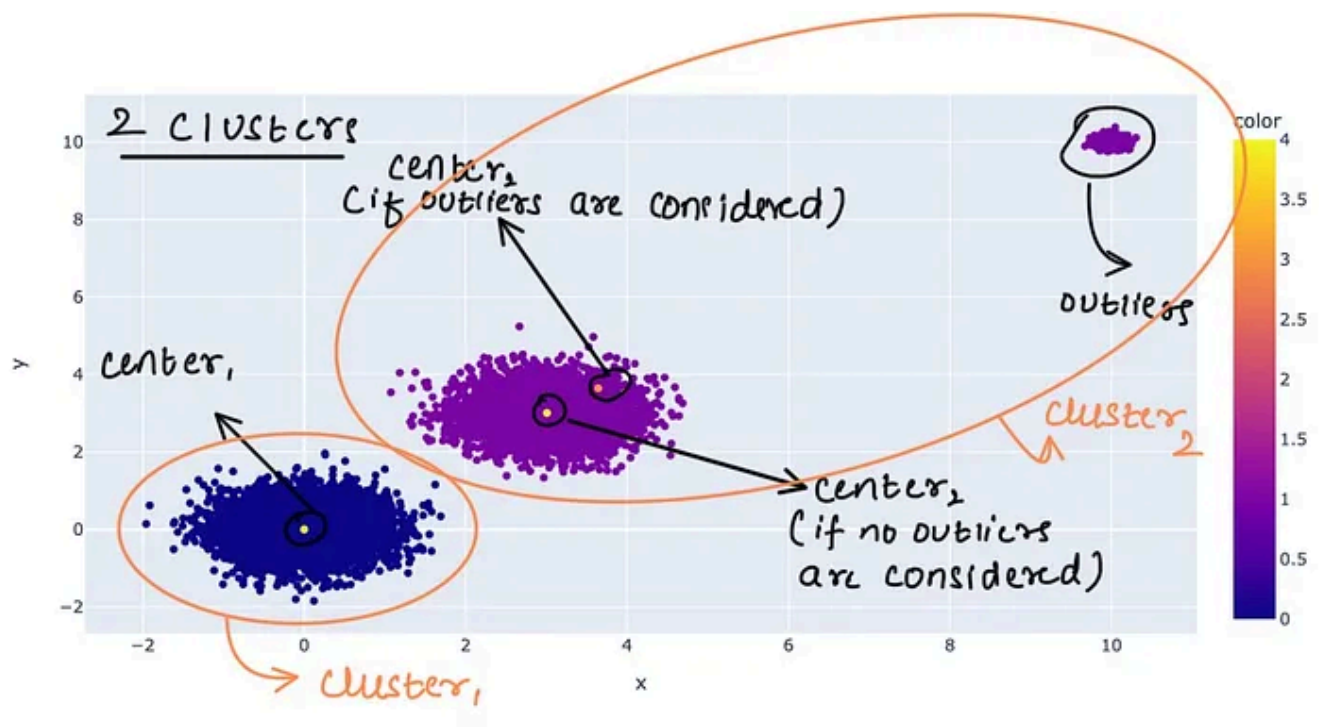


Fig. 5. Notice the centre of cluster2 in two different cases — clustering with and without outlier points. When outlier points are considered, the centre shifts a bit to the outliers. Original image.

### *How to solve this issue?*

Remember that outliers contribute disproportionately to the update step. You can spot outliers by plotting the density of centroid-to-points distances. In the above example, the distribution of distances of points in cluster2 to center2 look like —

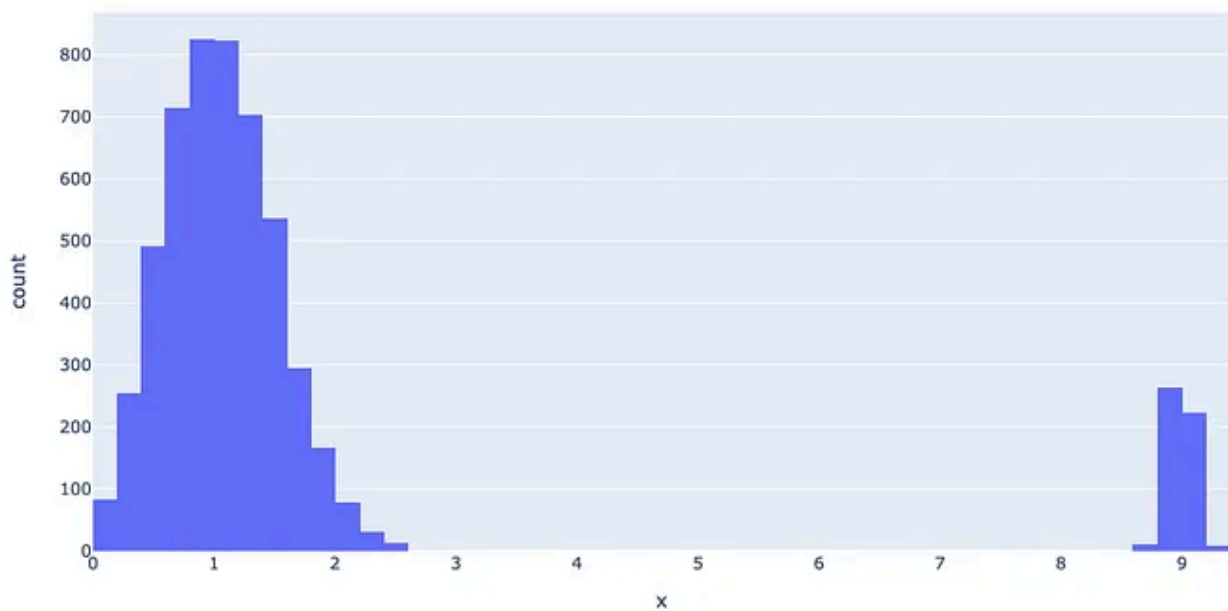


Fig. 6. Notice the small densities on the extreme right. They correspond to the outliers.

You can remove this point from the dataset to get the actual clusters.

Another way is to increase the number of clusters hoping that outliers can form a cluster of their own. In the above case (Fig. 5.), if we set the number of clusters=3, we get

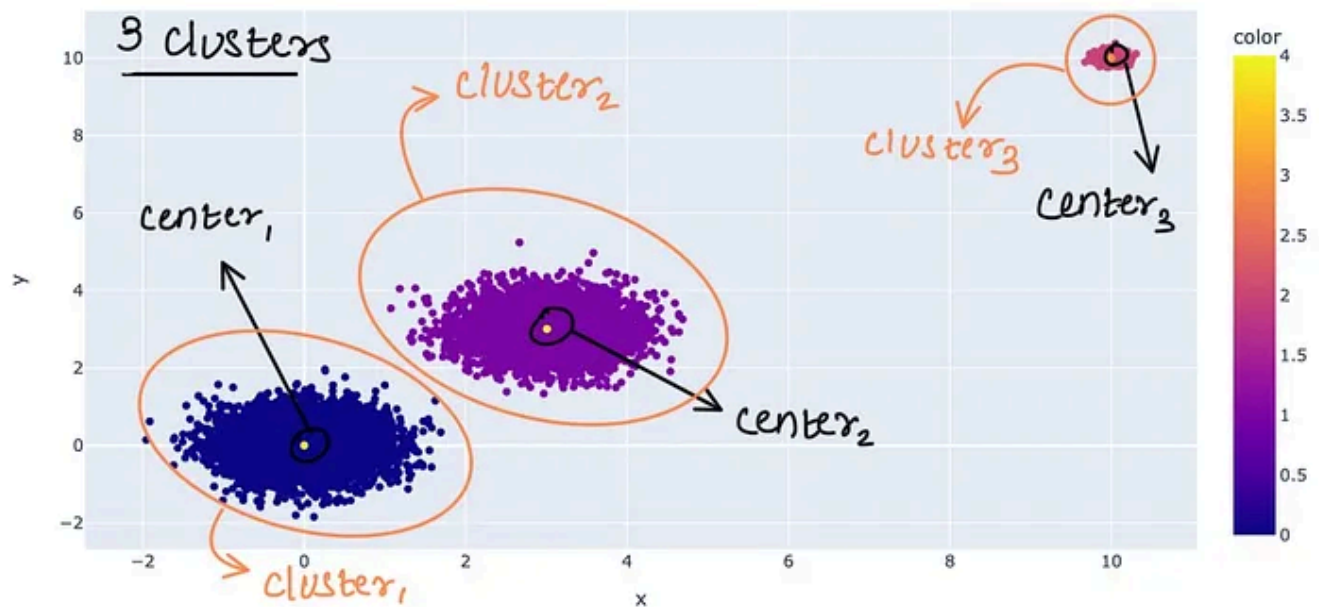


Fig. 7. Notice that the third cluster is completely made of outliers. Original image.

This reflects in the elbow plot (discussed above). Adding the third cluster drastically reduces the sum of intracluster variances. Hence, one would notice a sharp drop in the plot.

### Should you scale your data before using KMeans?

Many ML algorithms benefit from scaling the features using methods like min-max scaling or standard scaling. The ‘benefit’ is measured by an increase in the metric.

How does scaling affect KMeans? How do we know if it would be good or not? Let’s understand what scaling does to our model.

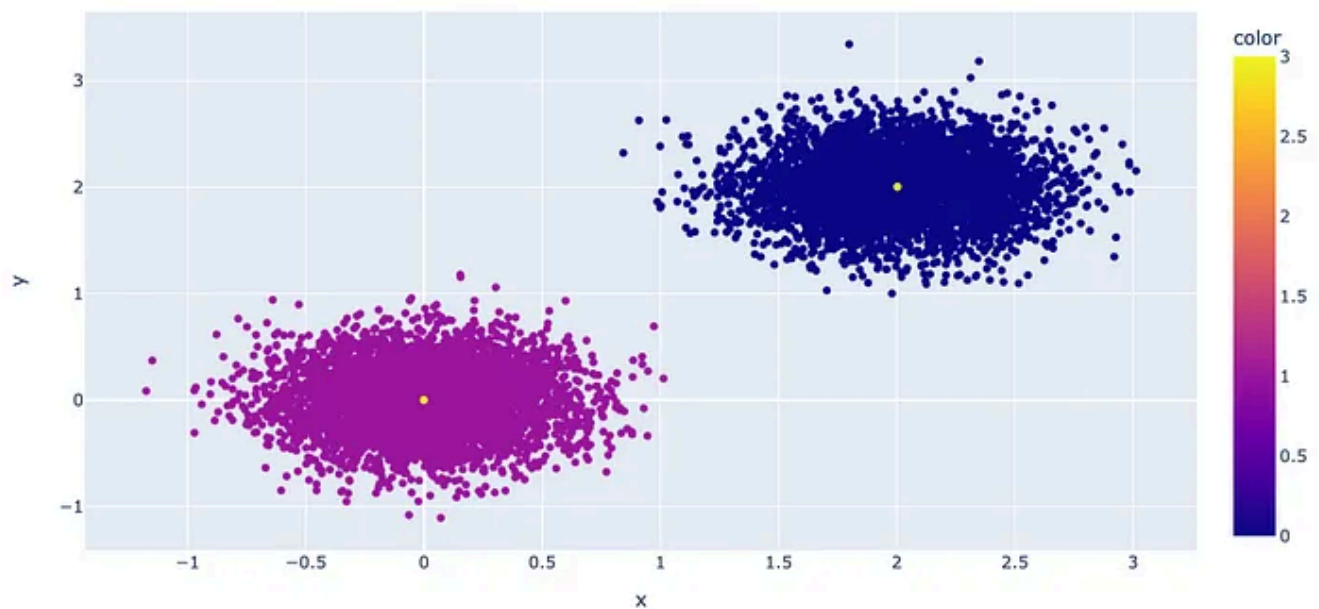
If we have two features,  $X_1$ ,  $X_2$ . The range of  $X_1$  is -1 to 1, and  $X_2$  is -100 to 100. While computing the intracluster variances,  $X_2$  will contribute more to the SSE than  $X_1$ . Hence, the model can minimize this SSE more by

minimizing the contribution from  $X_2$ . This won't happen if you use standard scaling where you transform the feature as —

$$X_{transformed} = (X - X_{mean}) / X_{std\_dev}$$

Let's look at examples!

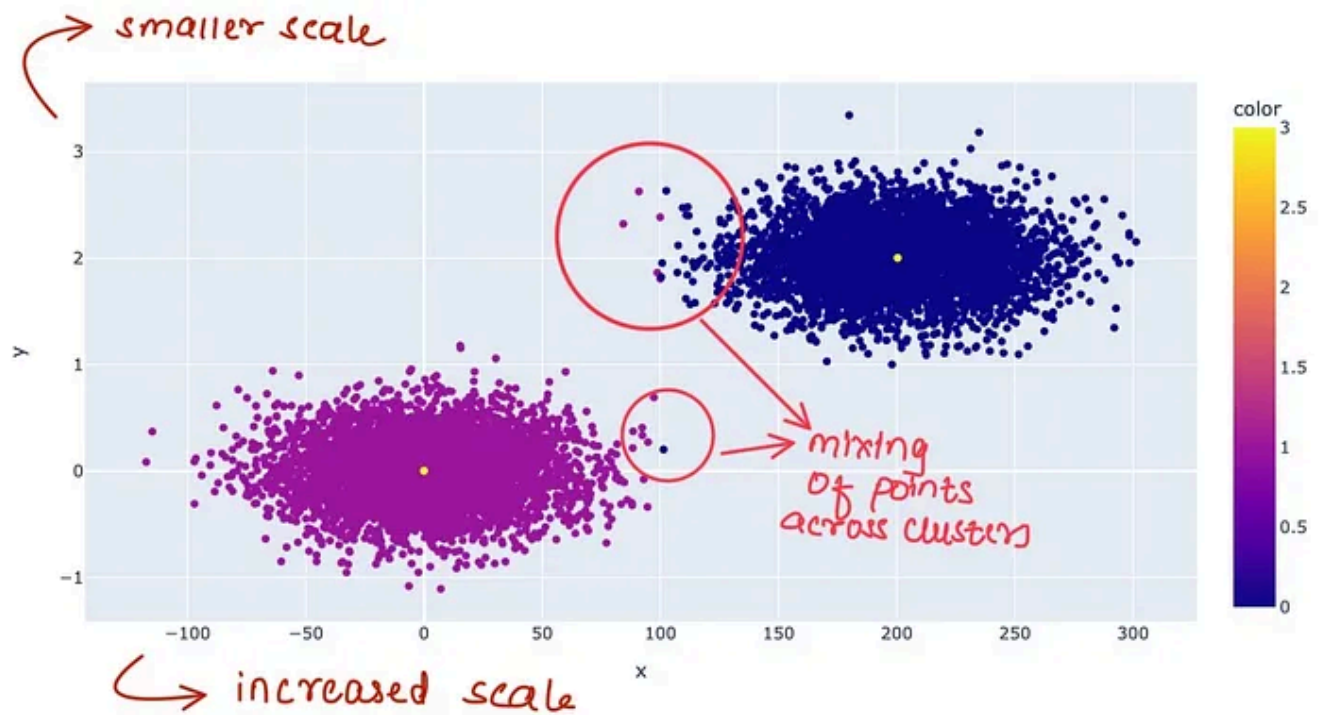
Scaled features —



Clusters with scaled features. The partition is perfect! Original image.

Same data, but features are unscaled —





Notice the scale on the x-axis and the mixing of points across clusters. Original image.

*Do you want a small set of features to dominate your cluster analysis, or do you want all of your features to have the same contribution?*

## How to measure the model performance?

The unavailability of labels limits what you can say about a clustering model. But it's not all gone. We mainly want to look at how well the clusters are formed in clustering. The definition of 'well' is not very precise.

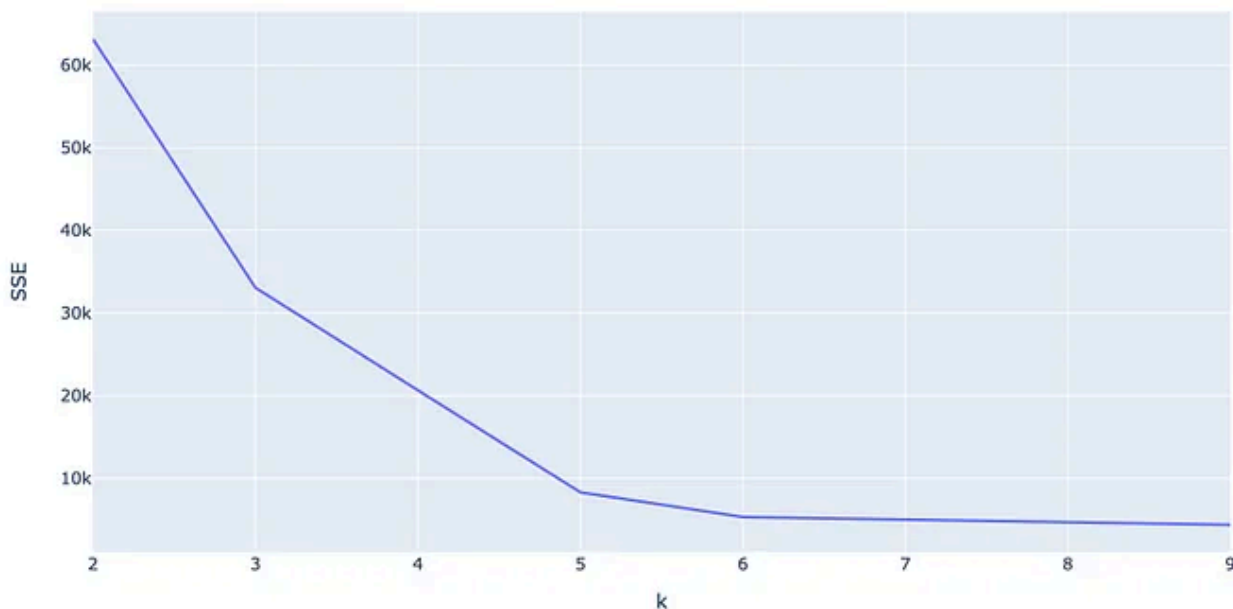
*Ideally, we want the clusters to be well separated, large but dense.*

You can get the largest cluster by setting  $k$  (the number of clusters) as small as one. You can get the densest clusters by using a large  $k$ , creating many dense micro-clusters. We lose any insightful information we might want to infer from the cluster in both cases.

Luckily, we can measure largeness, density and separation.

Largeness --> number of points in the clusters.  
Density --> Average of distance of two points in a cluster.  
Separation --> overlap between two cluster.

Consider a KMeans model being fitted on a dataset. How do we decide the number of clusters? Remember that KMeans minimizes the SSE across clusters. As we increase  $k$ , our SSE across clusters would decrease. At  $k$ =number of points in the dataset, the SSE will be 0 for all clusters because each point will be its own cluster and its own centroid. However, this is not useful. Therefore, we want to increase  $k$ , but only to a point where the drop in SSE on further increment is marginal —



Y axis=sum of SSE across clusters. X axis= $k$ . This is also known as scree plot or elbow curve. Original image

We see that the drop in SSE after  $k=5$  is marginal. Therefore, we may choose 5 clusters.

Are our clusters large enough? Let's check —



cluster index	number of points
0	5001
1	5001
2	9952
3	5047
4	4999

Seems so. Each cluster has a considerable number of points. It is tempting to choose  $k=5$  and move on in life. However, we haven't checked for the degree of separation. You can have significant and dense clusters that overlap. We don't want that.

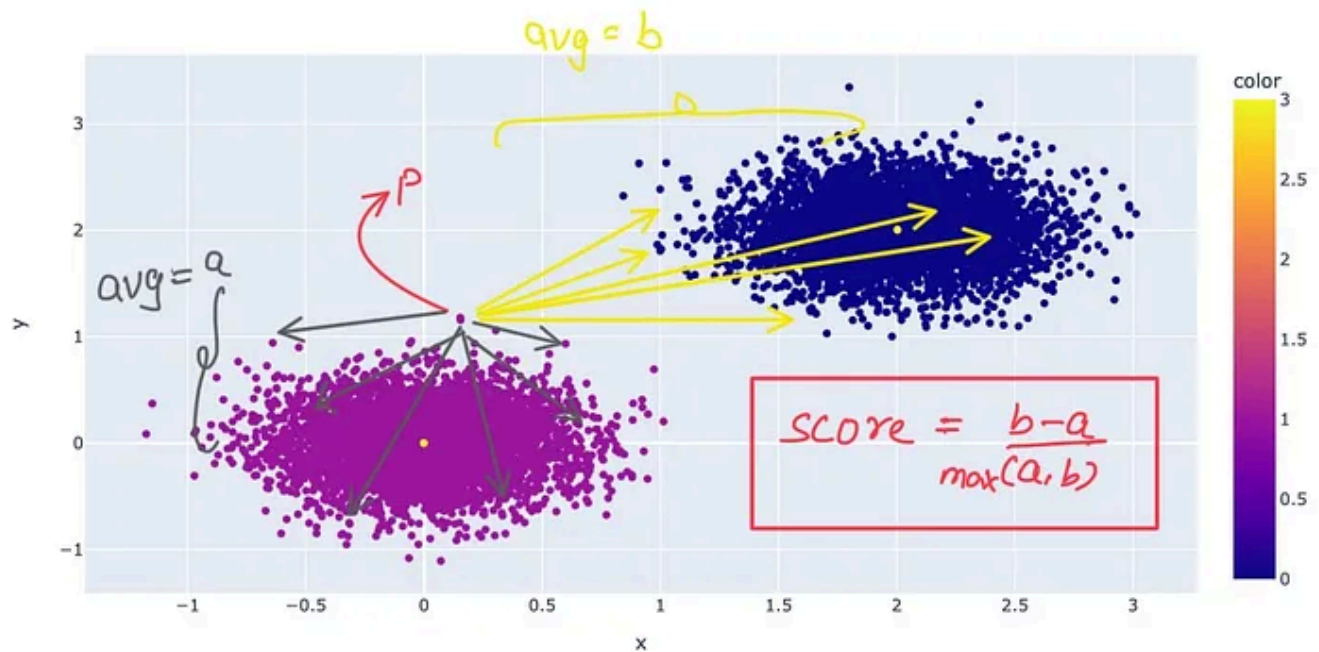
Let me introduce you to the **silhouette score**. It is a metric that tells you how much overlap exists between clusters. Ranging between -1 and 1, the larger the score, the lesser the overlap. It is calculated for each point  $p$  in the dataset by computing two measures —

1.  $a$  = Average intra-cluster distance of  $p$  with all the points in the same cluster . .
2.  $b$  = Average distance of  $p$  with any cluster that is not the one  $p$  belongs to. If there are  $N$  clusters, we get  $N-1$  such averages. Take the minimum of these and call it  $b$ .

$$\text{silhouette-score for } p = (b - a) / \max(b, a)$$

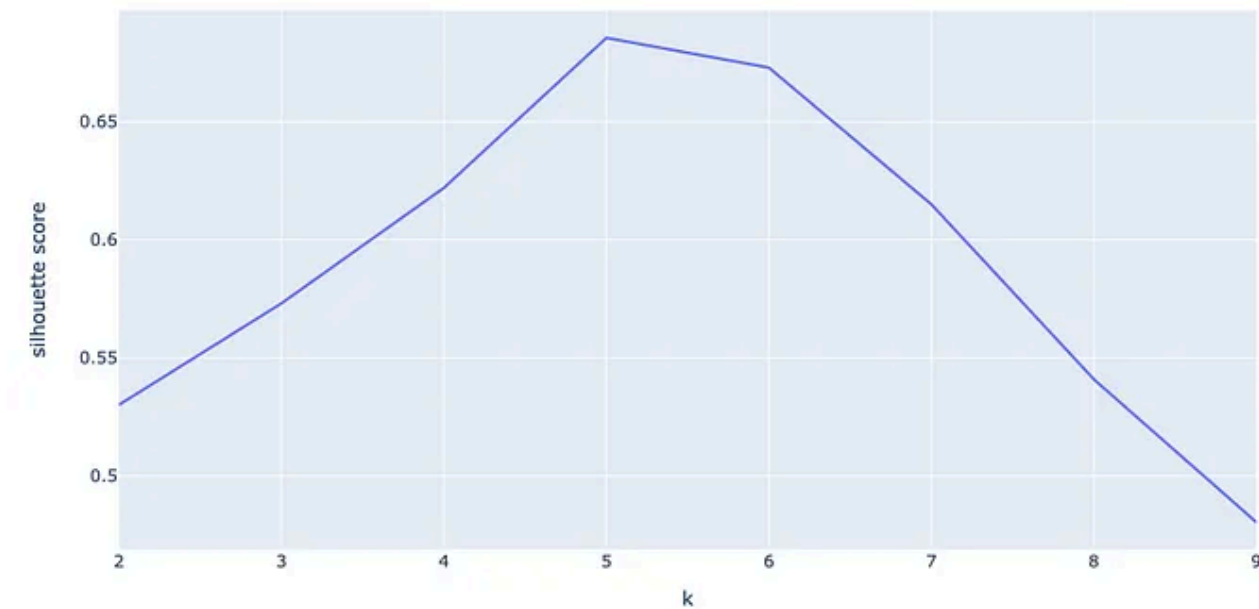
To get a score on a cluster level, average the scores of each point in the cluster.

Let's look at it intuitively. If two clusters overlap, many overlapping points will have lower **b** and vice-versa. Lower **b** means a lower score. If **a**=0, then the score is 1. This only happens when there is a single point in the cluster. If the above expression is unclear, refer to the visualization below —



Example visualization for silhouette score for a point  $p$ . Original image.

Plot silhouette score vs number of clusters —



silhouette score vs k. We get the highest score for k=5.

We can assure that  $k=5$  yields better clusters in size, density, and separation. Note that these measures focus more on the distribution of the embedding space. The semantics of the cluster depends on the application. For example, inspecting for topic dominance in clusters of documents.

## Conclusion

K-Means is a powerful tool to dig down into a complex dataset and find patterns using euclidean mathematics. Make sure you follow the above guidelines to avoid any inaccuracies. Hope you found them useful.

If you liked the article, do subscribe to my list and hit the follow button on your right. It would mean a lot to me!

Good day :)

Data Science

Machine Learning

Programming

Data Analysis

K Means Clustering



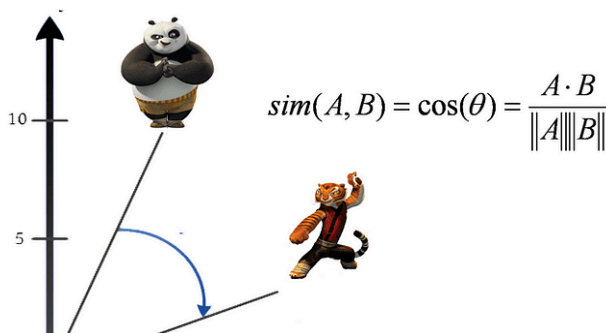
## Written by Dhruvil Karani

1K Followers · Writer for Towards Data Science

NLP Engineer | IIT Guwahati

Follow

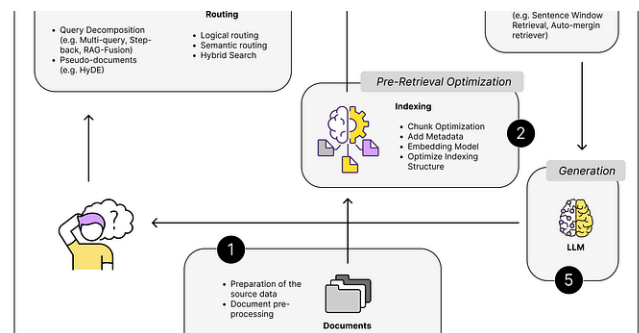
### More from Dhruvil Karani and Towards Data Science



 Dhruvil Karani in Towards Data Science

## Introduction to Word Embedding and Word2Vec

Word embedding is one of the most popular representation of document vocabulary. It is...



 Dominik Polzer in Towards Data Science

## 17 (Advanced) RAG Techniques to Turn Your LLM App Prototype into...

A collection of RAG techniques to help you develop your RAG app into something robus...

Sep 1, 2018 🖱️ 4.6K 💬 18



★ Jun 26 🖱️ 2.1K 💬 21

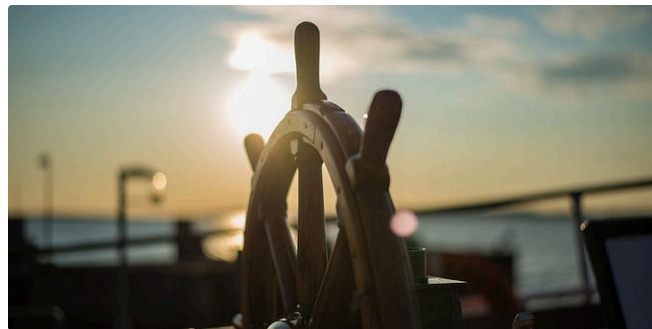


 Mauro Di Pietro in Towards Data Science

## GenAI with Python: RAG with LLM (Complete Tutorial)

Build your own ChatGPT with multimodal data and run it on your laptop without GPU

★ Jun 28 🖱️ 836 💬 14



 Dhruvil Karani in Towards Data Science

## How does Python work?

A simple explanation of how Python code is executed differently than older programmin...

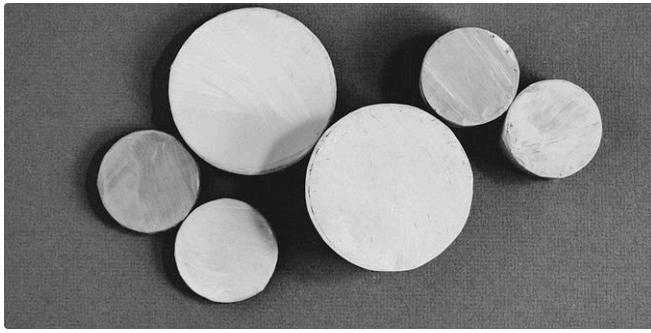
Jan 9, 2020 🖱️ 2K 💬 9



See all from Dhruvil Karani

See all from Towards Data Science

## Recommended from Medium



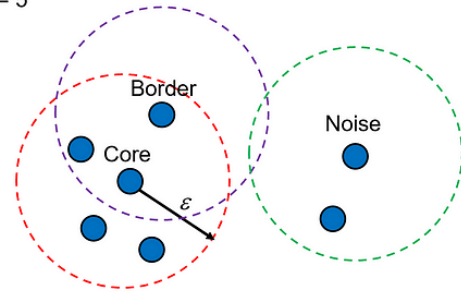
 Kay Jan Wong in Towards Data Science

## 6 Types of Clustering Methods— An Overview

Types of clustering methods and algorithms  
and when to use them

★ Mar 24, 2023 🖱 462 💬 2 📌<sup>+</sup>

MinPts = 5



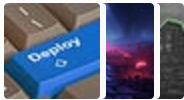
 Dr. Roi Yehos... in Artificial Intelligence in Plain En...

## DBSCAN: Density-Based Clustering

In-depth explanation of the algorithm  
including examples in Python

★ Oct 17, 2023 🖱 83 💬 1 📌<sup>+</sup>

### Lists



#### Predictive Modeling w/ Python

20 stories · 1424 saves



#### General Coding Knowledge

20 stories · 1452 saves



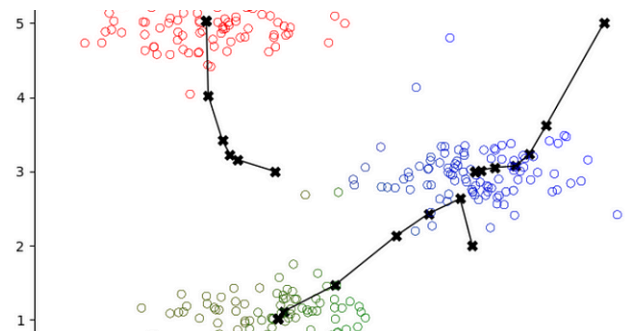
#### Practical Guides to Machine Learning


10 stories · 1723 saves

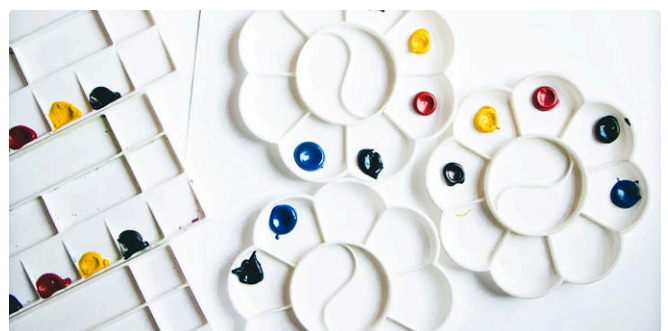


#### Coding & Development

11 stories · 726 saves



 Hasan Shahriar

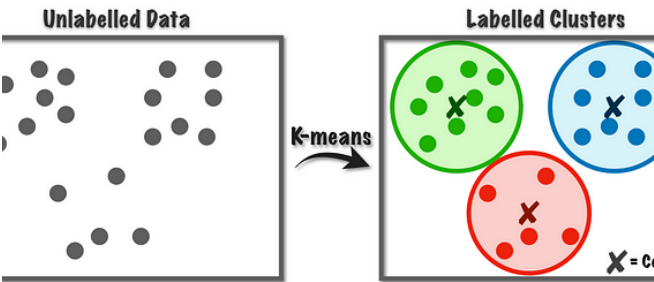



 Kirshi Yin in Technology Hits

# K-means Clustering

The K-means algorithm is a method to automatically cluster similar data points...

Apr 2



 Saarthak Gupta in Level Up Coding

## Understanding how K-Means Clustering Works (A detailed...

This article contains detailed and visual explanation of how K-Means algorithms...

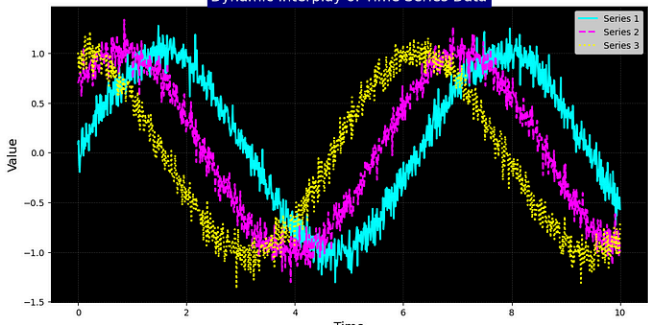
Jul 23  73



# Reveal the Dominant Colors in Your Images Using Python and K-Mean...

How to create a web app to extract dominant colors from an image using the K-Means...

★ Apr 9  107



 Bernhard Brugger in CodeX

## Understanding Time Series Clustering: Hands-On Hierarchica...

Unlike supervised learning, which depends on predefined outcomes for model training,...

★ Apr 5  139



See more recommendations