**Implementing K-Means Clustering from Scratch**

[[Deept Panchal](https://deeptpanchal.medium.com/?source=post_page-----ed60faace2e6--------------------------------)](https://deeptpanchal.medium.com/?source=post_page-----ed60faace2e6--------------------------------)

[Deept Panchal](https://deeptpanchal.medium.com/?source=post_page-----ed60faace2e6--------------------------------)

·

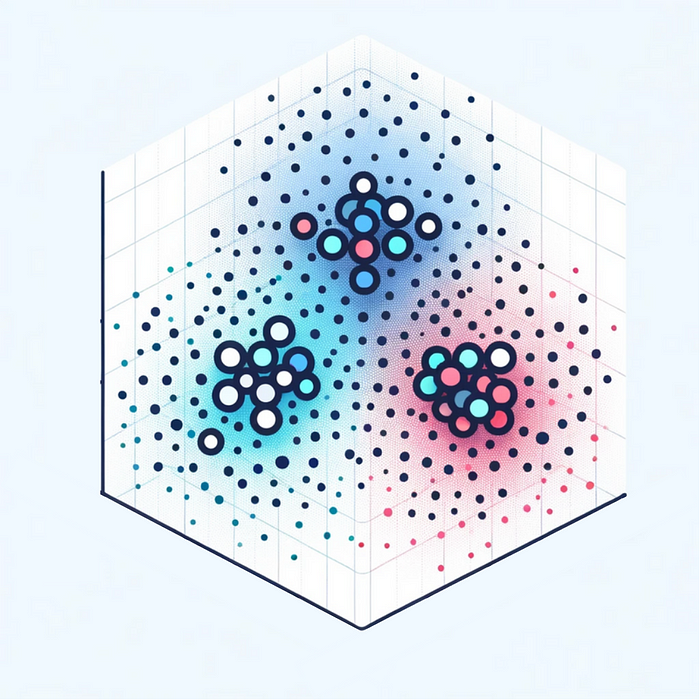
[Follow](https://medium.com/m/signin?actionUrl=https%3A%2F%2Fmedium.com%2F_%2Fsubscribe%2Fuser%2Fb54a0b2b8f20&operation=register&redirect=https%3A%2F%2Fdeeptpanchal.medium.com%2Fimplementing-k-means-clustering-from-scratch-ed60faace2e6&user=Deept+Panchal&userId=b54a0b2b8f20&source=post_page-b54a0b2b8f20----ed60faace2e6---------------------post_header-----------)

6 min read

·

5 days ago

129



**Introduction**

**K-means clustering** is a fundamental **unsupervised** learning algorithm that aims to partition a dataset into *k-*clusters, where each data point belongs to the cluster with the nearest mean. The process iteratively assigns data points to one of *k-*groups based on the closeness to the mean of each cluster, then recalculates the means **(centroids)** of these clusters until the algorithm converges.

**CSV Data Import and Column Parsing Function**

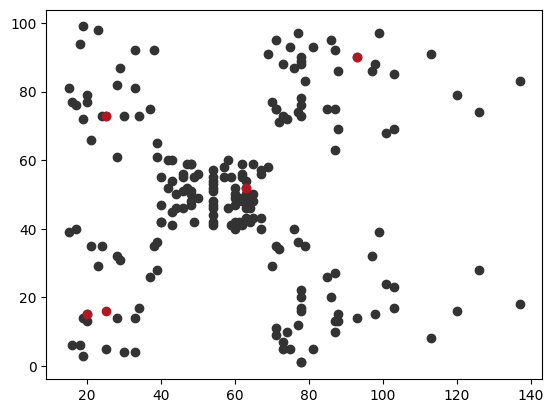
import random  
import math  
import matplotlib.pyplot as plt

def read\_csv(path, columns):  
 header = []  
 data = []  
 size = 0  
 with open(path, 'r') as file:  
 header = file.readline().strip().split(',')  
 for line in file:  
 data.append(line.strip().split(','))  
 size += 1  
 for i in range(size):  
 for j in columns:  
 data[i][j] = float(data[i][j])  
 return header, data, size

**Initializing Centroids**

It creates an initial set of centroids by **randomly selecting distinct data points**from the given dataset. This random initialization of centroids is a key step in K-means, as it influences the clustering process's efficiency and outcome. The algorithm then iteratively refines these centroids by assigning data points to the nearest centroid and recalculating the centroid positions until convergence.

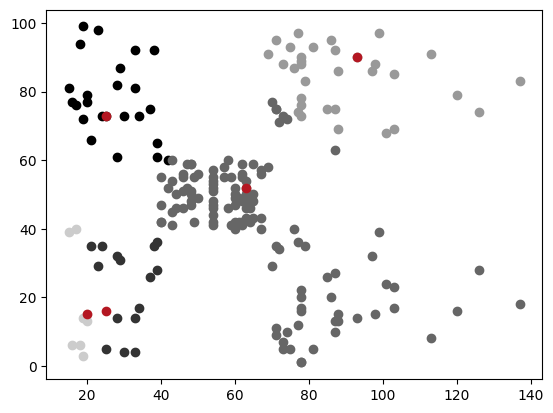
def initialize\_centroids(data, columns, size, n\_clusters):  
 centroids = [[0 for \_ in columns] for \_ in range(n\_clusters)]  
 for c\_i, d\_i in enumerate(random.sample(range(size), n\_clusters)):  
 for c\_j, d\_j in enumerate(columns):  
 centroids[c\_i][c\_j] = data[d\_i][d\_j]  
 return centroids



**Cluster Assignment**

It iterates through every data point and computes the Euclidean distance to each centroid. The function then updates the label for each data point to the index of its nearest centroid, determined by the shortest squared Euclidean distance. This assignment process is fundamental in K-means, as it categorizes each data point into a cluster, leading to the formation of distinct groups based on similarities in their features.

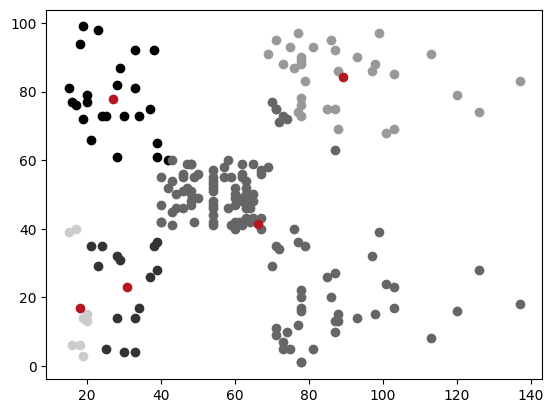
def assign\_labels(data, columns, size, centroids):  
 labels = [[float('inf'), None] for \_ in range(size)]  
 for d\_index, d\_list in enumerate(data):  
 for c\_index, c\_list in enumerate(centroids):  
 distance = 0  
 for index, value in enumerate(columns):  
 distance += (d\_list[value]-c\_list[index])\*\*2  
 squared\_distance = math.sqrt(distance)  
 if squared\_distance < labels[d\_index][0]:  
 labels[d\_index] = [squared\_distance, c\_index]  
 return labels



**Refining Cluster Centers**

After all data points have been assigned to the nearest centroid, this function recalculates the centroid of each cluster. It does so by averaging the features of all data points assigned to each cluster. As updating the centroids based on the current cluster assignments helps the algorithm to converge towards optimal cluster centers

def update\_centroids(data, columns, n\_clusters, labels):  
 new\_centroids = [[0 for \_ in columns] for \_ in range(n\_clusters)]  
 count = [0 for \_ in range(n\_clusters)]  
 for d\_index, d\_list in enumerate(data):  
 count[labels[d\_index][1]] += 1  
 for index, value in enumerate(columns):  
 new\_centroids[labels[d\_index][1]][index] += d\_list[value]  
 for c\_index, c\_value in enumerate(new\_centroids):  
 for index, value in enumerate(c\_value):  
 new\_centroids[c\_index][index] /= count[c\_index]  
 return new\_centroids



**Evaluating Cluster Cohesiveness**

Inertia, in K-means, refers to the sum of squared distances between each data point and its corresponding centroid. This function iterates through all data points, accumulating the squared distances from each point to its assigned centroid.

The lower the inertia, the more cohesive (or tightly grouped) the clusters are, indicating better clustering. High inertia suggests that data points are spread out and far from their centroids, implying less optimal clustering.

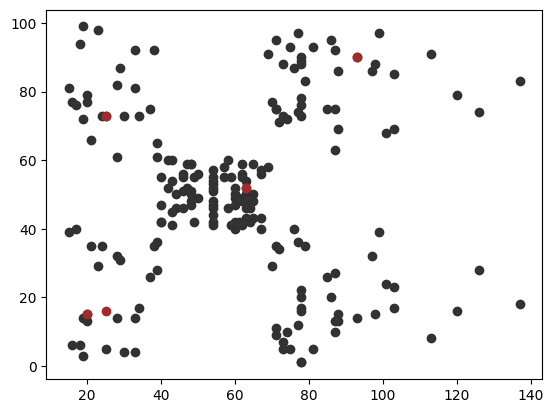
def calculate\_inertia(labels):  
 inertia = 0  
 for \_, value in enumerate(labels):  
 inertia += value[0] \*\* 2  
 return inertia

**Implementing Optimal Cluster Formation**

The KMeans function encapsulates the entire process of the K-means clustering algorithm. It begins by initializing centroids and iterates through a series of steps to form clusters. For each iteration (determined by n\_init), it initializes centroids using initialize\_centroids, then enters a loop where it continuously assigns labels to data points based on the nearest centroid using assign\_labels, and updates the centroids' positions with update\_centroids.

This loop continues until the centroids' positions stabilize (i.e., no further changes occur), indicating that the clusters have converged. After each iteration, it calculates the inertia (the sum of squared distances from each point to its centroid) using calculate\_inertia. The algorithm compares inertia across different initializations and retains the clustering (centroids, labels) with the lowest inertia, as this represents the most compact and well-separated clustering. This function effectively illustrates the iterative nature of K-means, seeking to minimize within-cluster variance (inertia) to achieve optimal clustering.

def KMeans(data, columns, size, n\_clusters, n\_init=30):  
 centroids = None  
 labels = None  
 inertia = float('inf')  
 for \_ in range(n\_init):  
 centroids\_ = initialize\_centroids(data, columns, size, n\_clusters)  
 try:  
 while True:  
 labels\_ = assign\_labels(data, columns, size, centroids\_)  
 new\_centroids = update\_centroids(  
 data, columns, n\_clusters, labels\_)  
 if new\_centroids == centroids\_:  
 break  
 centroids\_ = new\_centroids  
 inertia\_ = calculate\_inertia(labels\_)  
 if inertia\_ < inertia:  
 centroids, labels, inertia = centroids\_, labels\_, inertia\_  
 except:  
 continue  
 return centroids, labels, inertia



**Initial Data Exploration for K-Means Clustering**

dataset: <https://www.kaggle.com/datasets/shwetabh123/mall-customers>

filePath = './Mall\_Customers.csv'  
columns = [3, 4]  
header, data, size = read\_csv(filePath, columns)

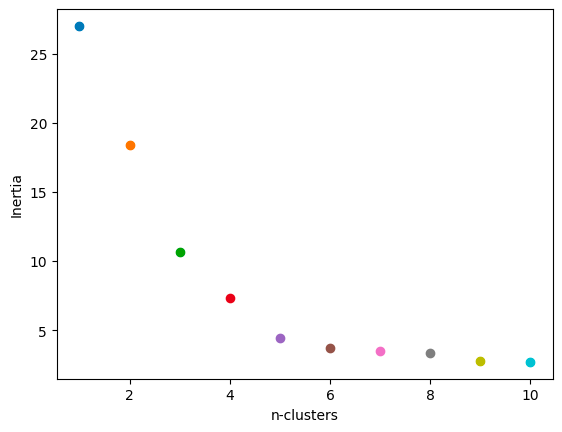
for d in data:  
 plt.scatter(float(d[3]), float(d[4]), color='#1f77b4')  
plt.show()

**Determining Optimal Cluster Count**

the given code snippet is performing an elbow method analysis to determine the optimal number of clusters for the dataset. For each number of clusters *i*, it calculates the clustering’s inertia, which is a measure of how internally coherent the clusters are. The inertia values are then plotted against the number of clusters.

Typically, as the number of clusters increases, inertia decreases; however, the rate of decrease sharply changes at the optimal number of clusters. This point, often resembling an “elbow” in the plot, is considered a good choice for *k* as it represents a balance between the number of clusters and the tightness of the clustering. This technique is a heuristic for finding a suitable number of clusters in a dataset.

for i in range(1, 11):  
 centroids, labels, inertia = KMeans(data, columns, size, i)  
 plt.scatter(i, inertia/10000)  
plt.xlabel('n-clusters')  
plt.ylabel('Inertia')  
plt.show()

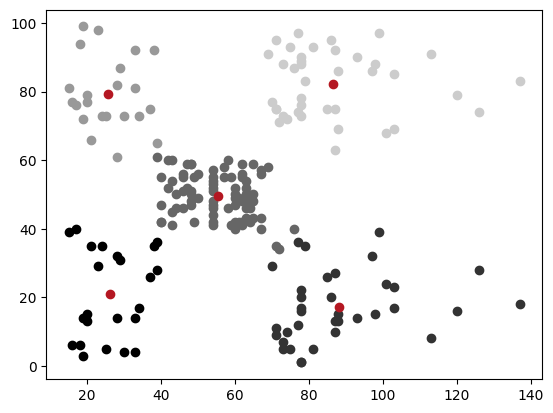


Looking at the inertia values, there is a noticeable angle in the plot at the point where the number of clusters is 5. This suggests that increasing the number of clusters beyond 5 does not result in a significant decrease in inertia, indicating that 5 is the optimal number of clusters for this particular dataset.

**Visualizing Cluster Distribution and Centroids in 5-Cluster**

n\_clusters = 5  
centroids, labels, inertia = KMeans(data, columns, size, n\_clusters)  
print(inertia)  
print(centroids)

for i, d in enumerate(data):  
 c = str(round(labels[i][1]/n\_clusters, 1))  
 plt.scatter(d[3], d[4], color=c)  
for c in centroids:  
 plt.scatter(c[0], c[1], color='brown')  
plt.show()



**Advantages of K-means Clustering**

* **Simplicity and Efficiency:** K-means is easy to understand and implement. It’s also computationally efficient, making it suitable for large datasets.
* **Flexibility:**It can be used with any type of data for which a distance measure can be defined.

**Disadvantages of K-means Clustering**

* **Choice of *k*:** The number of clusters *k* needs to be specified in advance, which can be challenging if the underlying distribution of the data is unknown.
* **Sensitivity to Initial Centroids:** The final result can be significantly influenced by the initial choice of centroids. Poor initialization can lead to suboptimal clustering.
* **Assumption of Spherical Clusters:** K-means assumes that clusters are spherical and equally sized, which might not always be the case, leading to poor performance on elongated or irregularly shaped clusters.
* **Outliers:** K-means is sensitive to outliers, as they can skew the position of centroids.

Despite its limitations, k-means clustering remains a cornerstone technique in unsupervised learning due to its simplicity and effectiveness in many practical scenarios. Its often use it as a baseline method for clustering and as a means to gain insights into the structure of data.

[Machine Learning](https://medium.com/tag/machine-learning?source=post_page-----ed60faace2e6---------------machine_learning-----------------)

[K Means](https://medium.com/tag/k-means?source=post_page-----ed60faace2e6---------------k_means-----------------)

[Clustering](https://medium.com/tag/clustering?source=post_page-----ed60faace2e6---------------clustering-----------------)

[Unsupervised Learning](https://medium.com/tag/unsupervised-learning?source=post_page-----ed60faace2e6---------------unsupervised_learning-----------------)

[Python](https://medium.com/tag/python?source=post_page-----ed60faace2e6---------------python-----------------)