Foundations of Machine Learning

K-means Clustering Assignment

Emmanuel Kirui Barkacha
GitHub Repository

May 27, 2025

1 Introduction

Clustering is an unsupervised learning technique used to group similar data points together. It's used in: Customer segmentation, Document classification, Image compression or Anomaly detection. One of the most used clustering algorithm is the K-Means clustering for partitioning data into distinct groups based on similarity. It works by dividing a dataset into K non-overlapping clusters, where each data point belongs to the cluster with the nearest center (centroid). The goal of K-Means is to minimize the within-cluster sum of squares(WCSS), also known as inertia.

In this project, K-Means clustering is applied to a customer dataset containing the following features: Customer ID, Gender, Age, Annual Income (k$), and Spending Score (1–100). The aim is to perform customer segmentation, identifying groups of customers with similar purchasing behavior and demographics, which can be valuable for targeted marketing and business strategy.

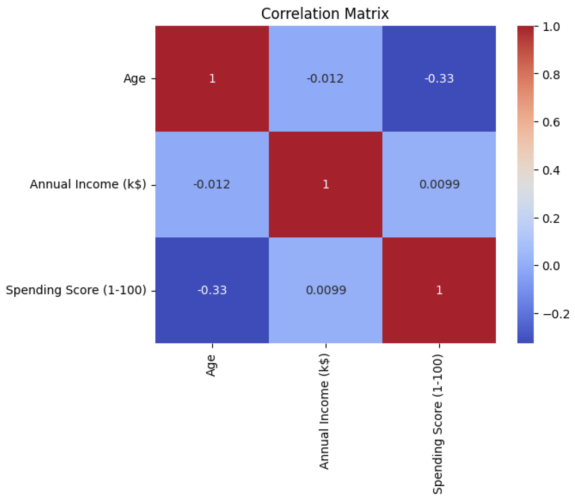
We implement and compare several initialization techniques of K-Means clustering: Random initialization, Kmean++ initialization and Forgy initialization. To evaluate clustering performance, we use metrics such as inertia and the Silhouette Score. We also implement the Elbow Method to determine the optimal number of clusters. Finally, results from the custom implementation are compared with scikit-learn's KMeans to validate correctness and performance.

2 Methodology

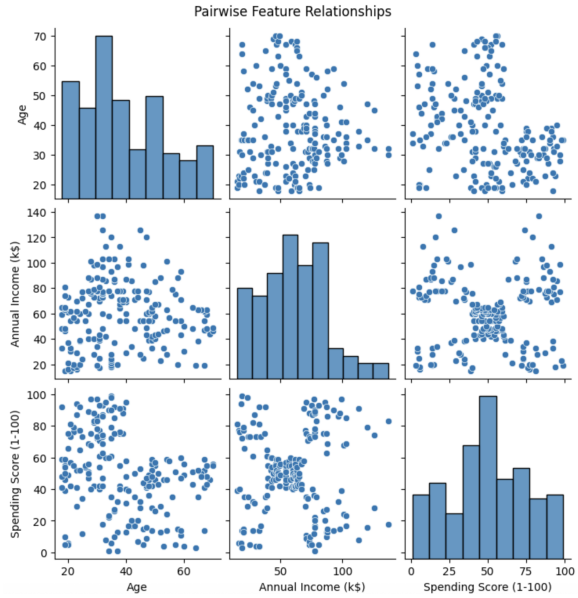
In this section we describe the approach we used to perform clustering using K-Means on Customer Segmentation Dataset from Kaggle link.

2.1 Data Preprocessing

The dataset contains five features: Customer ID, Gender, Age, Annual Income (k$), and Spending Score (1–100) and 200 samples. Since Customer ID is simply a unique identifier, it is excluded from clustering. For Gender, which is categorical, we encode it to numerical form using label encoding (Male = 0, Female = 1). Continuous features are standardized using 'StandardScaler' from sklearn to ensure equal contribution to distance calculations. We also conducted an exploratory analysis to better understand the relationships between features in the dataset. For example we plotted the correlation matrix to identity if there is any correlation between the features.From the correlation matrix 2.1 we could see that there is no strong correlation among the features. We also plotted distribution plot among the features to examine feature Relationships as in 2.1 which helps to visualize potential cluster structures in different feature combinations.



Correlation Matrix



Features Relationship

From Features Relationship plot 2.1 we note that in the Annual Income vs Spending Score plot, we observe distinct groupings, suggesting these features are well-suited for K-Means clustering. Including Age, it adds some separation but also introduces overlap between clusters. Hence when visualizing all features, it becomes clear that combinations of Annual Income and Spending Score consistently show natural segmentation, supporting their selection for the primary clustering experiments.

2.2 K-Means Algorithm

1: Choose the number of clusters K
2: Initialize centroids $\mu_1, \mu_2, \ldots, \mu_K$ (randomly or using KMeans++)
3: **repeat**
4: **for** each data point x_i in dataset D **do**
5: Assign x_i to the cluster with the nearest centroid:

$$c_i = \arg\min_j \text{distance}(x_i, \mu_j)$$

6: **end for**
7: **for** each cluster $j \in \{1, \ldots, K\}$ **do**
8: Update centroid μ_j to be the mean of all points assigned to cluster j:

$$\mu_j = \frac{1}{|C_j|} \sum_{x \in C_j} x$$

9: **end for**
10: **until** centroids do not change significantly or assignments stabilize

In the K-Means algorithm, the Euclidean distance is one of the common methods used to measure the similarity between data points and centroids. For a data point $\mathbf{x} \in \mathbb{R}^n$ and a centroid $\mathbf{c} \in \mathbb{R}^n$, the Euclidean distance is computed as:

$$\text{dist}(\mathbf{x}, \mathbf{c}) = \sqrt{\sum_{i=1}^{n} (x_i - c_i)^2}$$

During each iteration, data points are assigned to the nearest centroid based on this distance.

For convergence, we compare the centroids from the current iteration ($\mathbf{C}_{\text{new}}$) to those from the previous iteration ($\mathbf{C}_{\text{old}}$). Convergence is achieved when the change in centroids is less than a specified tolerance ϵ:

$$\|\mathbf{C}_{\text{new}} - \mathbf{C}_{\text{old}}\|_2 < \epsilon$$

This ensures the algorithm stops once the centroids have stabilized, avoiding unnecessary computation and iterations. Another method for checking convergence is equating the previous centroid to the current one were we checks whether the centroids remain exactly the same between iterations:

$$\mathbf{C}_{\text{new}} = \mathbf{C}_{\text{old}}$$

2.3 Initialization Techniques

In this project we experimented with three different centroid initialization techniques.

2.3.1 Random Initialization

Centroids are initialized by assigning random cluster labels to each point and computing their means.

Random Initialization Algorithm

1: Choose number of clusters K
2: Randomly assign a label from 1 to K to each data point
3: **for** each cluster $j \in \{1, \ldots, K\}$ **do**
4: Compute initial centroid μ_j as:

$$\mu_j = \frac{1}{|C_j|} \sum_{x \in C_j} x$$

5: **end for**
6: Return centroids $\mu_1, \mu_2, \ldots, \mu_K$

2.3.2 K-Means++ Initialization

The first centroid is chosen randomly, and subsequent centroids are chosen based on a probability proportional to their squared distance from the nearest existing centroid.

K-Means++ Initialization Algorithm

1: Choose number of clusters K
2: Randomly select the first centroid μ_1 from the dataset
3: **for** $j = 2$ to K **do**
4: **for** each data point x_i in dataset D **do**
5: Compute $D(x_i) = \min_{\mu \in \{\mu_1, \ldots, \mu_{j-1}\}} \|x_i - \mu\|^2$
6: **end for**
7: Select a new centroid μ_j from D with probability proportional to $D(x_i)$
8: **end for**
9: Return centroids $\mu_1, \mu_2, \ldots, \mu_K$

2.3.3 Forgy Initialization

Randomly selects K data points directly as initial centroids.

Forgy Initialization of Centroids Algorithm

1: Choose number of clusters K
2: Randomly select K distinct data points from dataset D
3: Assign them as initial centroids $\mu_1, \mu_2, \ldots, \mu_K$

4: Return centroids

2.4 Evaluation Metrics

To evaluate clustering performance, we used:

1. **Inertia:** Sum of squared distances between each point and its assigned centroid. Lower inertia indicates compact clusters hence better results.

2. **Silhouette Score:** Measures how similar an object is to its own cluster compared to others. Ranges from -1 to 1. A higher score indicates well-separated clusters.

2.5 Optimal K Selection

We used two methods to determine the ideal number of clusters:

1. **Elbow Method:** Plot inertia values against various K and identify the point where the decrease in inertia begins to level off.

2. **Silhouette Analysis:** Evaluate silhouette scores for different values of K and select the one with the highest score.

3 Results And Discussion

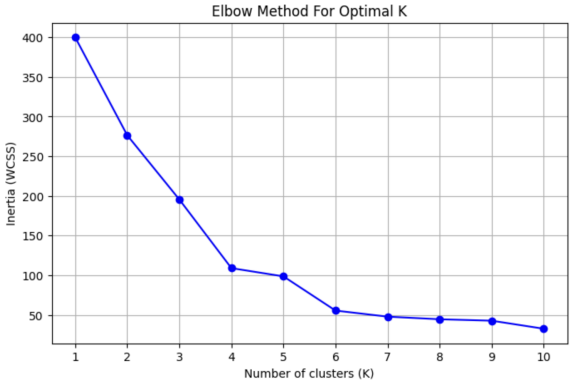
3.1 Comparison of Clustering Performance Across Initialization Methods

Method	**Iterations**	**Inertia**	**Silhouette**
Random	7	57.2862	0.4236
KMeans++	7	55.3758	0.5212
Forgy	5	55.3773	0.5205

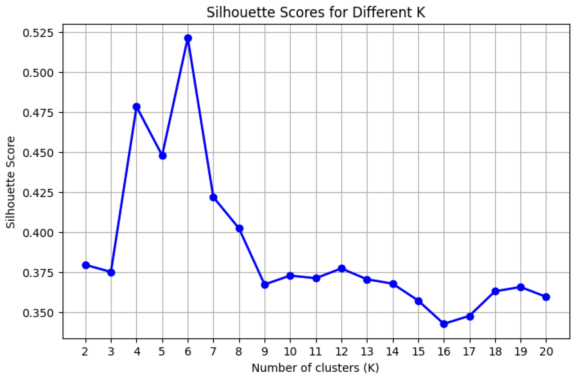
Table 1: Clustering Performance Metrics by Initialization Method with seed 42

The table above 3.1 summarizes the clustering performance of our three centroid initialization methods: Random, KMeans++, and Forgy. The number of iterations until convergence, the final inertia, and the silhouette scores are recorded. Both KMeans++ and Forgy methods converge faster (5 to 7 iterations) and achieve lower inertia values compared to Random initialization, indicating more compact clusters. Additionally, KMeans++ and Forgy show higher silhouette scores, reflecting better-defined and more separated clusters. This highlights the advantage of smart centroid initialization in improving clustering quality and efficiency.

3.2 Determining the Optimal Number of Clusters



Elbow Method For Optimal K

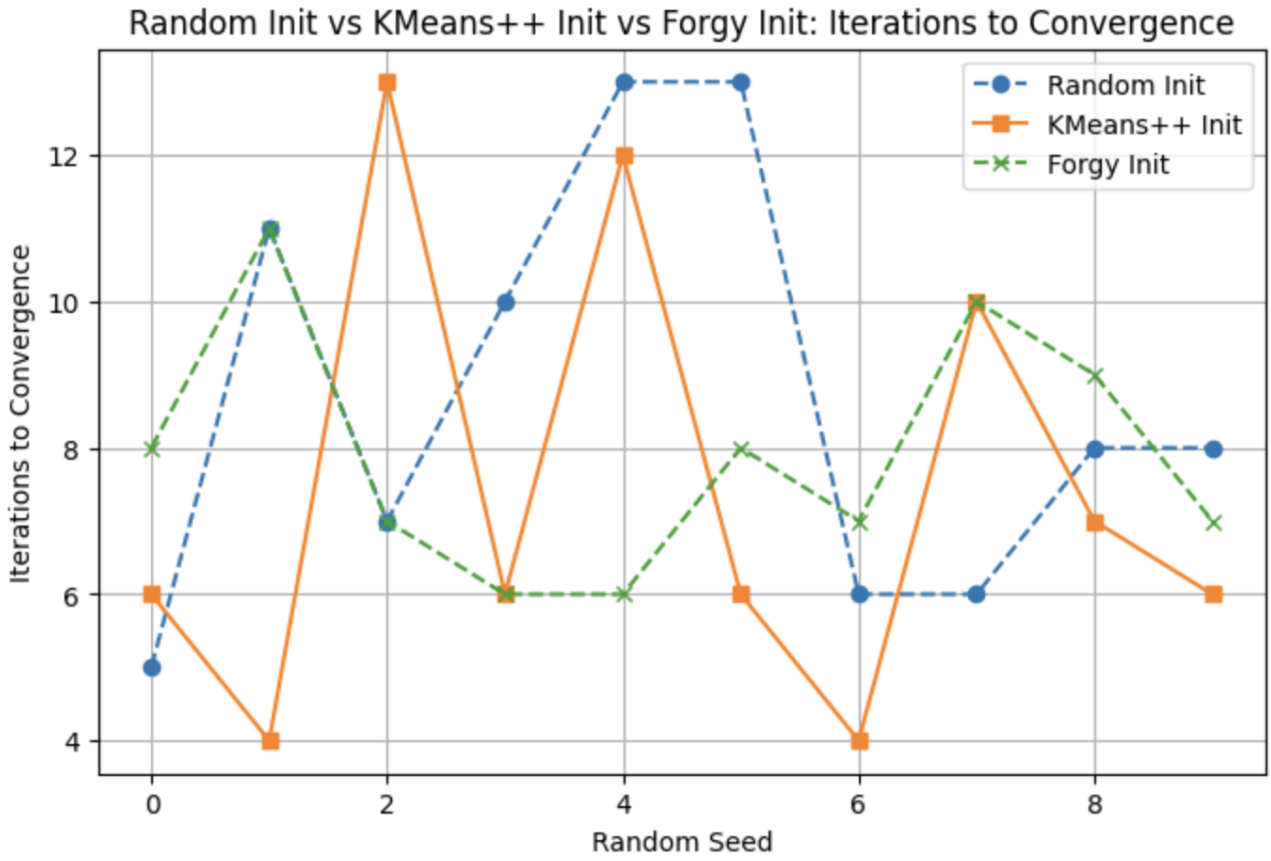


Silhouette Score

To determine the optimal number of clusters K, we applied both the Elbow Method and Silhouette Analysis. The Elbow Method involves plotting the inertia (sum of squared distances within clusters) against different values of K and identifying the point where the rate of decrease sharply changes ("elbow"). This point suggests a balance between model complexity and cluster compactness. Silhouette Analysis evaluates the quality of clustering by measuring how similar each data point is to its own cluster compared to other clusters, with scores ranging from -1 to 1. Higher silhouette scores indicate better-defined and more separated clusters.

Our analysis showed that inertia decreases rapidly up to $K = 6$, after which the improvement slows, suggesting that 6 clusters provide a good trade-off. Similarly, silhouette scores peak is at $K = 6$, confirming this choice as optimal for segmenting the customer data.

3.3 Convergence Behavior Across Initialization Methods

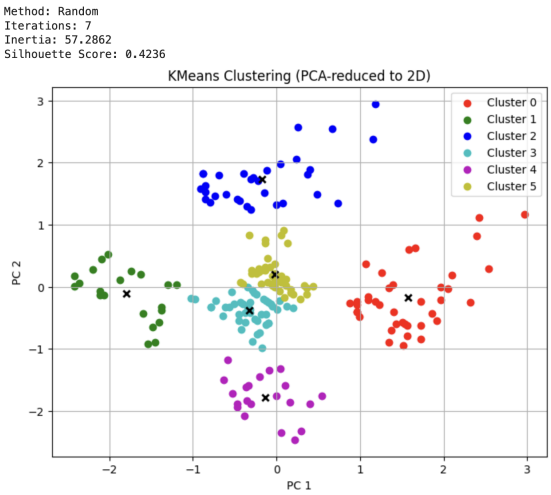


Convergence Behavior Across Initialization Methods

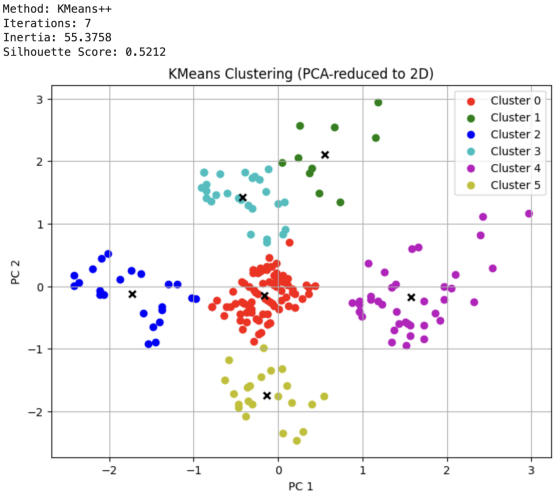
We evaluated the number of iterations to convergence for K-Means using Random, KMeans++, and Forgy centroid initializations over 10 different random seeds. The iteration counts varied considerably across seeds for all methods, showing a high degree of randomness in convergence behavior. While KMeans++ and Forgy initializations generally tended to converge in fewer iterations than Random initialization, the differences were not consistent across all runs. Some seeds resulted in very similar iteration counts regardless of the initialization method.

This variability suggests that, despite the theoretical advantages of smarter initialization strategies like KMeans++, in practice the convergence speed can still be influenced by the initial seed and data distribution. Therefore, multiple runs with different seeds are important to robustly assess algorithm performance.

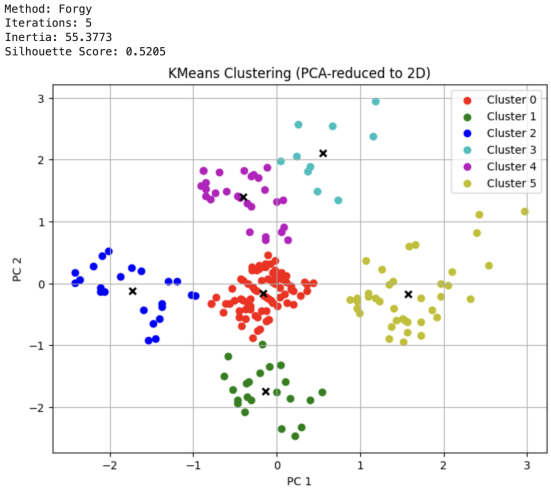
3.4 Impact of Initialization Methods on Clustering Quality



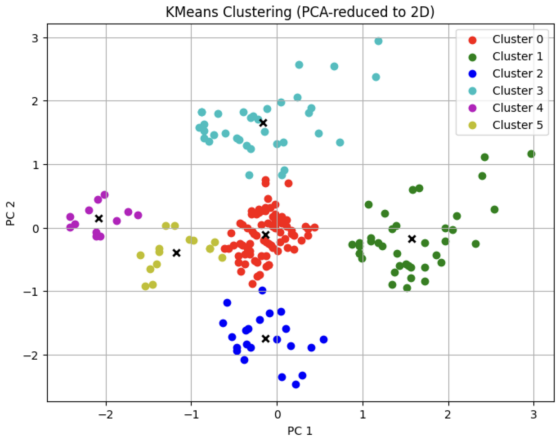
Random Initialization Clusters



Kmeans++ Initialization Clusters



Forgy Initialization Clusters



Sklearn Kmeans Clusters

The PCA-reduced scatter plots generated for the three initialization methods: Random, KMeans++, and Forgy—highlight noticeable differences in cluster formation and convergence behavior. Random initialization tends to produce clusters that are less compact and sometimes overlapping, due to the arbitrary selection of initial centroids. This often results in slower convergence and less optimal clustering. On the other hand, KMeans++ initialization typically produces a well-separated and more balanced clusters by choosing initial centroids based on distance probabilities, which helps avoid poor starting positions and speeds up convergence. Forgy initialization, which selects initial centroids randomly from the data points themselves, generally performs better than Random initialization and shows convergence speeds comparable

to KMeans++, though it can be less consistent depending on the random seed. Overall, these visualizations demonstrate that the choice of initialization method significantly impacts the efficiency and quality of K-Means clustering, with KMeans++ providing more reliable and interpretable cluster assignments. We can also note that our result from the three initialization methods were different from the one from sklearn Kmeans package 3.4.

3.5 Feature Selection Based on Clustering Performance

Features	**Silhouette Score**	**Inertia**
Annual Income (k$), Spending Score (1–100)	0.5013	55.3758
Age, Annual Income (k$), Spending Score (1–100)	0.3695	160.2330
Gender, Age, Annual Income (k$), Spending Score (1–100)	0.3140	297.5029
Age, Spending Score (1–100)	0.3094	63.6689
Age, Annual Income (k$)	0.3020	68.2646

Table 2: Clustering Performance for Different Feature Sets

To determine the most effective feature combination for clustering, we evaluated K-Means++ clustering performance across several feature subsets using Silhouette Score and Inertia as metrics. The feature set 'Annual Income (k$)' and 'Spending Score (1-100)' demonstrated the best performance, achieving the highest Silhouette Score (0.5013) and the lowest Inertia (55.3758). This indicates that these two features capture meaningful customer segments with compact and well-separated clusters. Including additional features such as Age and Gender generally reduced clustering quality, suggesting that these may introduce noise or less relevant information. Based on these results, we selected the feature set with 'Annual Income (k$)' and 'Spending Score (1-100)' for subsequent analysis, ensuring better cluster interpretability and robustness.

4 Conclusion

In this study, we looked at the impact of different centroid initialization methods on the performance of the K-Means clustering algorithm applied to customer segmentation data. Our custom implementations of Random, KMeans++, and Forgy initialization methods were evaluated using metrics such as inertia, silhouette score and iterations to convergence. The results consistently demonstrated that KMeans++ and Forgy initialization outperform Random initialization in both clustering quality and convergence speed though sometimes it was random. KMeans++ showed a slight edge by producing more compact and well-separated clusters, due to its distance-based centroid selection strategy. Moreover, the Elbow Method and Silhouette Analysis both supported the choice of six clusters (K=6) as the optimal segmentation, balancing model complexity and cluster cohesion. Feature selection analysis revealed that using 'Annual Income (k$)' and 'Spending Score (1-100)' as the primary features yields the most meaningful clusters, whereas adding age and gender tended to dilute clustering effectiveness. This result ensures a focused and interpretable segmentation model.

Overall, the study showed the importance of careful centroid initialization and feature selection in enhancing K-Means clustering results.

All of this analysis can be found in my github repository.