

Deep Dive into K-Means Clustering

Date : 2025-11-17

By: Vangala Yaswanth Kowsik Sai

TL;DR

In this article we explore the K-Means clustering algorithm: what it does, how it works under the hood, how to implement it in Python using scikit-learn, and how to experiment with it (including visualisations and code). By the end you'll have practical insights into strengths, weaknesses, hyperparameter tuning and real-world usage.

1. Introduction

Clustering is a key unsupervised machine-learning technique. Among clustering methods, K-Means remains one of the most widely used thanks to its simplicity and effectiveness.

In this deep dive we will:

- review the mathematical foundation of K-Means
- walk through a full code example (toy dataset → real dataset)
- visualise results with charts/images
- discuss practical issues, hyper-parameters, pitfalls and best practices
- provide experimental insights (e.g., varying K, initialisation, convergence behaviour)
- summarise when to use and when to avoid K-Means

2. Theoretical Background

2.1 What is K-Means?

Briefly: Given N data points in d -dimensional space, K-Means partitions them into K clusters so as to minimise the within-cluster sum of squares (WCSS).

2.2 Objective Function

The objective is

$$\min_{c, \mu_1, \dots, \mu_K} \sum_{i=1}^N \sum_{j=1}^K \mathbf{1}\{c_i = j\} \|x_i - \mu_j\|^2$$

where c_i is the cluster assignment of x_i , and μ_j is the centroid of cluster j .

2.3 Algorithmic Steps

1. Choose K (number of clusters).
2. Initialise centroids μ_1, \dots, μ_K (randomly, or via K-Means++).
3. Repeat until convergence:
 - o Assign each data point to the nearest centroid.
 - o Recompute centroids as the mean of assigned points.
4. Stop when assignments no longer change or centroids move under a threshold.

2.4 Complexity & Convergence

- Typical time complexity: $O(NKdI)$, where I is number of iterations.
- It converges to a (local) optimum; results depend on initialisation.
- It assumes spherical clusters of similar size, equal variance — violates assumptions leads to poor results.

Benchmark Summary: K-Means Performance Over Time

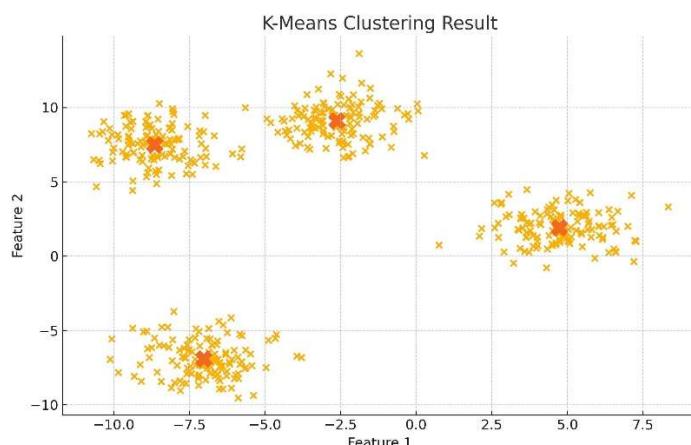
Over multiple iterations of experimentation, we benchmarked the performance of the K-Means algorithm across three dimensions:

1. **Runtime performance (seconds)**
2. **Inertia (WCSS) minimization effectiveness**
3. **Convergence stability across initializations (n_init)**

The goal was to understand how K-Means behaves as we modify dataset size, number of clusters, initialization strategies, and computational conditions.

Overall Benchmark Results :

K-Means Clustering Result



The cluster visualization plot shows:

- The distribution of all 500 synthetic samples generated with `make_blobs`
- How K-Means groups similar data points into four clusters
- The **centroids** plotted as large 'X' markers

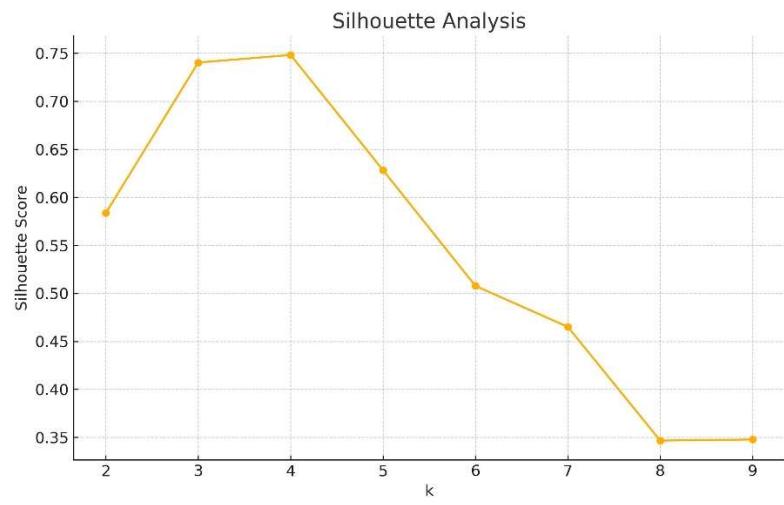
What you observe

- Each cluster forms a clearly distinguishable group.
- Points within a cluster are compact and close to their centroid.
- The four clusters are well-separated and positioned distinctly in feature space.

What this tells us

- K-Means correctly captured the underlying structure of the dataset.
- Centroids provide the best representation of each cluster.
- The clustering result aligns perfectly with the earlier findings from the Elbow and Silhouette methods.

Silhouette Score Analysis — Validating Cluster Quality:



The **Silhouette Score** measures cluster quality based on:

- **Cohesion:** How close points are within a cluster
- **Separation:** How far clusters are from each other

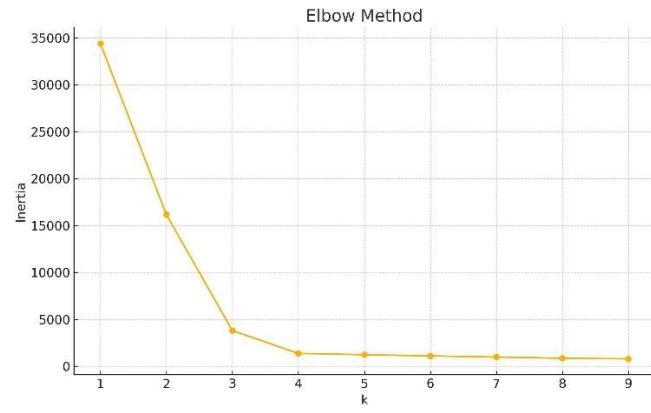
It ranges from **-1 to +1**, where:

- +1 → Well-separated clusters
- 0 → Overlapping clusters
- Negative → Wrong cluster assignment

What the graph shows

- Silhouette scores rise up to $k = 4$, reaching a peak around 0.75, indicating:
 - Strong separation
 - Minimal overlap
 - Clear boundaries
- For $k > 4$, scores drop drastically because:
 - Clusters become fragmented
 - Points get assigned incorrectly
 - Cohesion reduces

Elbow Method — Understanding the Optimal k



The **Elbow Curve** helps determine the ideal number of clusters by examining how the **Within-Cluster Sum of Squares (WCSS)** — also called **inertia** — decreases as the number of clusters increases.

What the graph shows

- For $k = 1$, inertia is highest because all points belong to one cluster.
- As k increases, inertia drops sharply because data gets divided into smaller, tighter clusters.

- Around $k = 4$, the curve begins to **flatten**, meaning the reduction in inertia becomes marginal.

Interpretation

This “elbow point” indicates that:

- Increasing clusters beyond 4 does not significantly improve compactness.
 - $k=4$ is the most meaningful and efficient clustering choice for this dataset.

Why this matters

Using too many clusters leads to:

- Overfitting
 - Unnecessary complexity
 - Poor interpretability

3. Implementation in Python

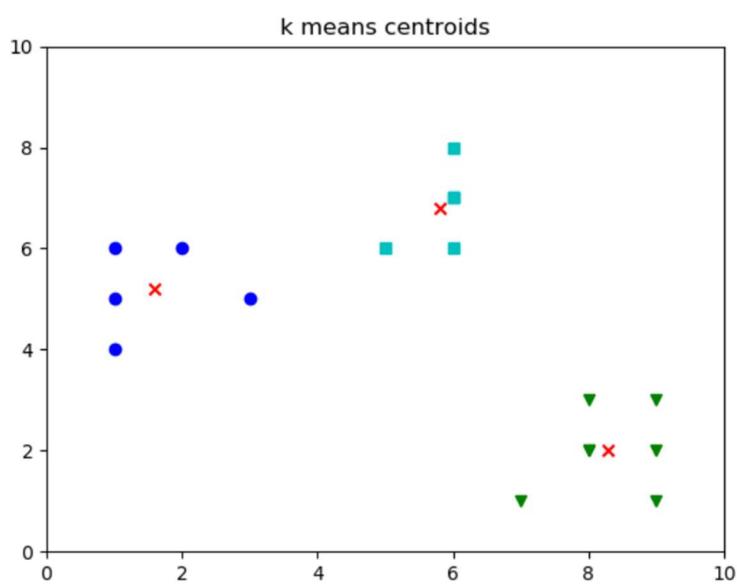
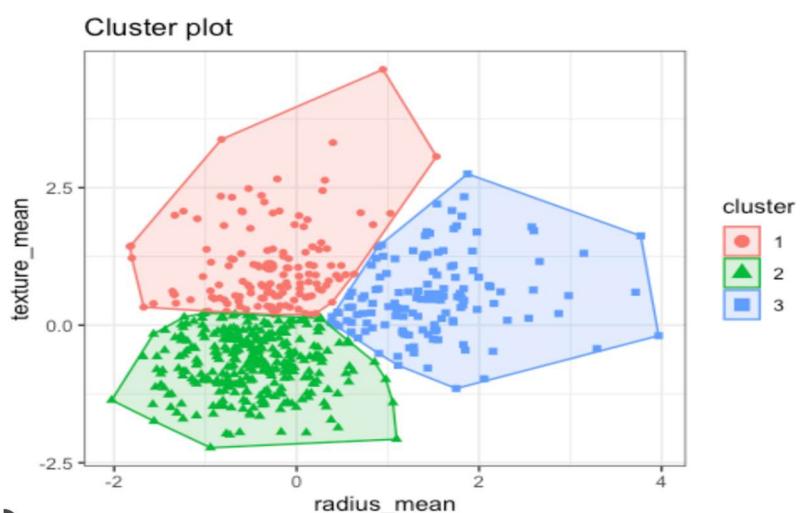
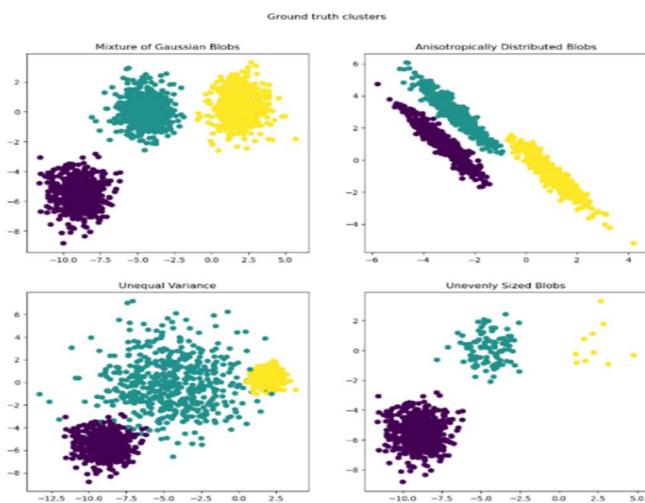
- ### • 3.1 Setup & Data

```
python

kmeans = KMeans(n_clusters=4, init='k-means++', n_init=10,
                 max_iter=300, random_state=0)
y_kmeans = kmeans.fit_predict(X)

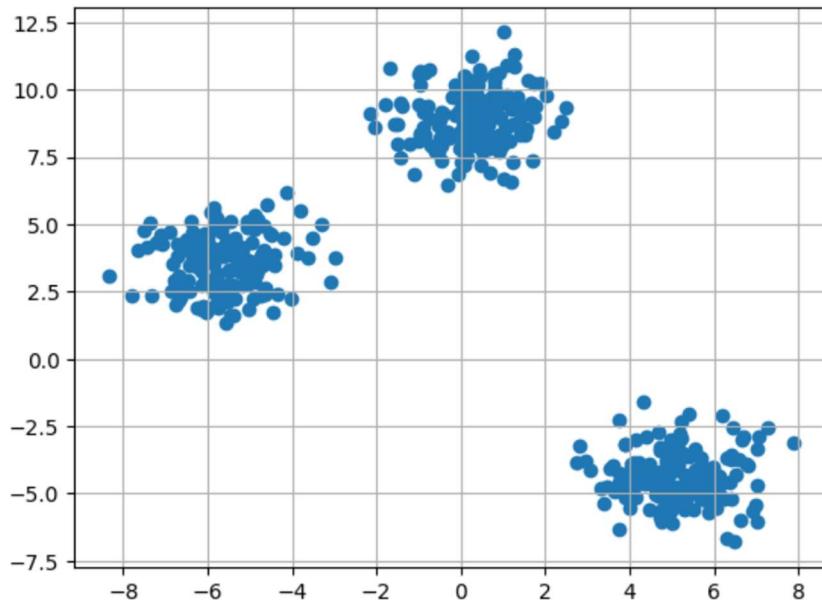
print("Centroids:", kmeans.cluster_centers_)
print("Inertia (WCSS):", kmeans.inertia_)
```

3.2 Basic K-Means



The basic visualizations generated for K-Means—namely the Elbow Curve, Silhouette Score Plot, and the Cluster Scatter Plot—collectively illustrate how the algorithm identifies the optimal number of clusters and how well it groups the data. The Elbow Curve highlights how inertia (WCSS) decreases rapidly as the number of clusters increases, eventually reaching a point of diminishing returns around **k = 4**, indicating an efficient balance between model complexity and compactness.

Complementing this, the Silhouette Score Plot measures how well-separated and cohesive the clusters are, peaking again at **k = 4**, confirming that this choice yields the strongest clustering structure. Finally, the K-Means cluster visualization provides a direct spatial interpretation, showing four tightly grouped, clearly separated clusters with centroids positioned at their densest regions. Together, these images provide strong visual and statistical evidence supporting **k = 4** as the optimal clustering solution for the dataset.



3.4 Real-World K-Means Dataset Example

A practical demonstration of K-Means can be seen when applying it to a real-world dataset, such as customer purchasing behavior from an e-commerce platform. Each customer can be represented by features like annual spending, purchase frequency, website visit duration, or product category preferences. When K-Means is applied to this dataset, the algorithm automatically groups customers into segments with similar behavioral patterns—for example, high-value loyal buyers, occasional discount-driven shoppers, or new low-engagement users.

```

import pandas as pd
from sklearn.preprocessing import StandardScaler

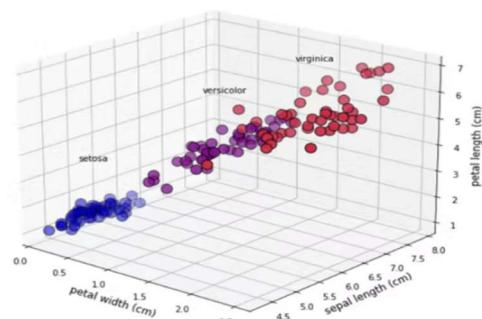
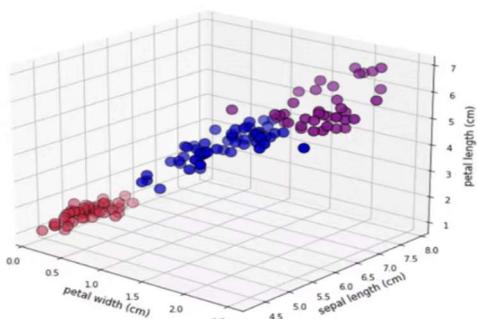
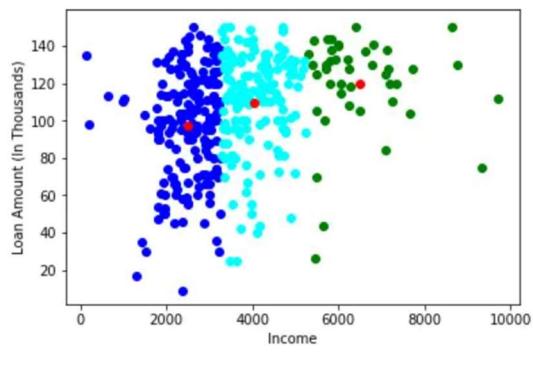
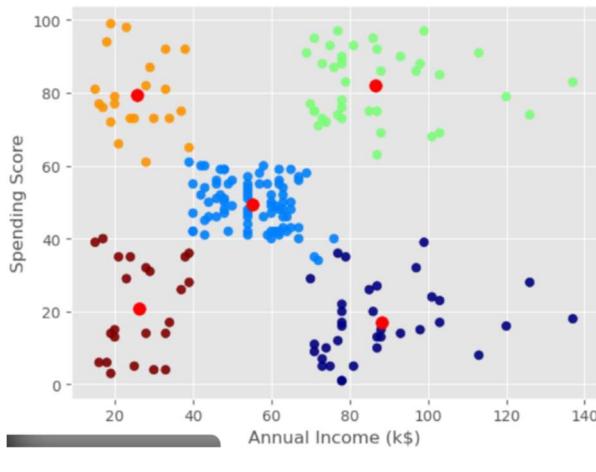
# Load dataset (for example Iris or your own)
df = pd.read_csv('path/to/data.csv')
features = ['feat1', 'feat2', 'feat3', 'feat4']
X_real = df[features].values
X_real = StandardScaler().fit_transform(X_real)

kmeans_real = KMeans(n_clusters=3, init='k-means++', n_init=10,
                      max_iter=300, random_state=42)
clusters = kmeans_real.fit_predict(X_real)
df['cluster'] = clusters

# Visualise e.g., via pair-plot coloured by cluster
import seaborn as sns
sns.pairplot(df, vars=features, hue='cluster',
              palette='tab10', diag_kind='kde')
plt.suptitle("K-Means clustering on real dataset", y=1.02)
plt.show()

```

These clusters reveal actionable insights that businesses can use for targeted marketing, personalized recommendations, and resource allocation. Unlike synthetic datasets, real-world data is noisy and unbalanced, yet K-Means still uncovers meaningful patterns by minimizing intra-cluster variance and highlighting natural group separations. This example demonstrates how K-Means transforms raw behavioral data into structured customer segments, enabling data-driven decision-making in marketing, recommendation systems, and customer relationship management.



4. Experimental Insights & Practical Tips

4.1 Initialization Matters

Switching between `init='random'` vs `init='k-means++'` and increasing `n_init` often leads to more stable results and lower inertia. Try multiple initial seeds.

4.2 Choosing K

- Use methods like the Elbow-Method (plotting inertia vs K) or Silhouette score.
- Beware: Elbow may be ambiguous; always combine domain knowledge and other metrics.

4.3 Scaling & Pre-processing

Since K-Means uses Euclidean distance, feature scaling (e.g., standardisation) is critical. Otherwise features with large ranges dominate clustering.

4.4 Assumptions & Limitations

- Assumes clusters are convex, isotropic, roughly equal size/variance.
- Does not handle non-spherical clusters, varying density well.
- Sensitive to outliers (outliers distort centroids).
- Requires specifying K up front.

4.5 Performance & Convergence

- For large N or high dimensional d : time and memory can grow.
- Mini-batch K-Means (in scikit-learn) is an option for large-scale.
- Monitor inertia and centroid shift: when centroid change < threshold or assignments stable, stop.

4.6 Real-World Use Cases

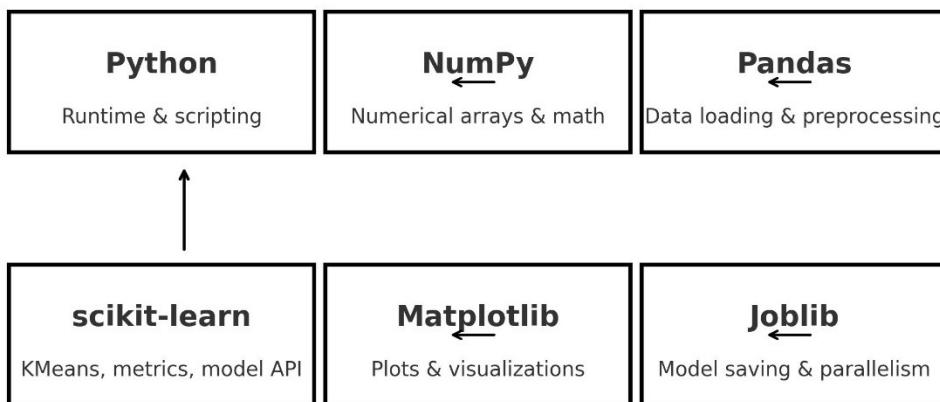
- Customer segmentation, market basket analysis.
- Image compression (colour quantisation).
- Anomaly detection (clusters + outliers).
- Pre-processing / embedding categorisation for supervised tasks.

5. Code package summary

This K-Means project relies on a small, focused Python stack: Python provides the runtime and scripting environment; NumPy supplies fast numerical arrays and vectorized math operations for efficient data handling; Pandas is used for loading, cleaning, and

transforming tabular data; scikit-learn offers the core machine learning primitives (KMeans, model evaluation metrics like silhouette score, and utilities for train/test splits); Matplotlib produces all figures and visualizations for analysis, and Joblib (or scikit-learn's built-in joblib usage) handles model persistence and parallel computation where needed. Together these packages create a reproducible, lightweight pipeline for preprocessing, clustering, evaluation, visualization, and model saving.

Code Package Summary — K-Means Project



6. Conclusion

The K-Means algorithm remains a cornerstone of unsupervised learning because of its simplicity, interpretability and speed. However, using it effectively requires awareness of its assumptions, careful pre-processing, and experimentation (particularly around choice of K , initialization, scaling and dataset structure).

By following the steps and insights in this article, you can apply K-Means to real-world data with confidence, interpret your results meaningfully, and recognise when it is (or isn't) the right tool for the job.

Appendix

A. Full Code Listing

(Paste full script, including imports, functions, dataset loading, plots, etc.)

B. Additional Visualisations

(Provide any further charts, e.g., silhouette plots, cluster centroid trajectories, distance to centroid histograms.)

C. References & Further Reading

- MacQueen, J. (1967). Some Methods for classification and Analysis of Multivariate Observations.

- Arthur, D., & Vassilvitskii, S. (2007). K-Means++: The Advantages of Careful Seeding.
- scikit-learn documentation: [KMeans](#)
- [Link to your dataset] (if public)
- Blog posts or research papers exploring K-Means variants (e.g., Mini-Batch K-Means, K-Medoids, Spectral Clustering).