Determining the Optimal Number of Clusters in K-Means: A Comparative Analysis of the Elbow Method, Silhouette Coefficient, and Davies-Bouldin Index

Evaluating cluster quality metrics to enhance K-Means performance and ensure meaningful data partitioning.

Hernández Pérez Erick Fernando

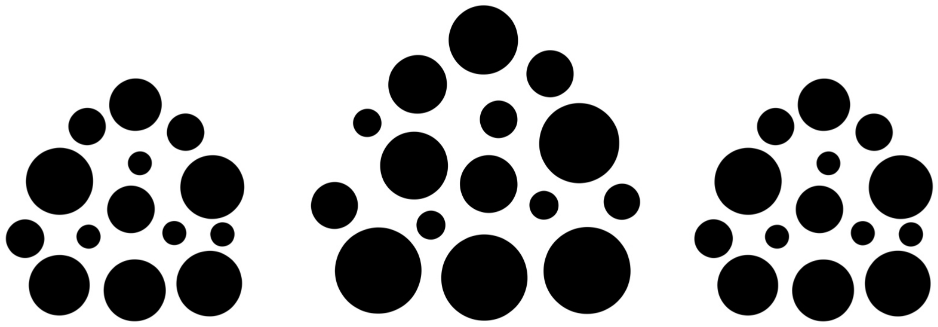

Figure 1

Clustering is a fundamental task in unsupervised learning, used to identify patterns and group similar data points without predefined labels. However, one of the main challenges in clustering, particularly in the widely used k-means algorithm, is determining the optimal number of clusters (k). Unlike supervised learning, where model performance can be directly evaluated against known labels, clustering lacks a definitive answer. The "correct" number of clusters is often subjective and depends on the dataset's structure, domain knowledge, and the chosen evaluation metric.

Several methods have been developed to guide this decision, including the Elbow Method, the Silhouette Coefficient, and the Davies-Bouldin Index. Each of these techniques provides a different perspective on cluster quality, yet none offer an absolute solution. The Elbow Method helps identify an inflection point where adding more clusters yields diminishing returns, while the Silhouette Coefficient evaluates how well-separated the clusters are. Meanwhile, the Davies-Bouldin Index assesses intra-cluster similarity relative to inter-cluster differences. Despite their usefulness, these metrics may sometimes provide conflicting results, making it necessary to analyze them collectively rather than relying on a single criterion.

Ultimately, choosing the right number of clusters is as much an analytical challenge as it is an interpretative one. This article explores these three methods in detail, highlighting their strengths, limitations, and practical applications in determining a meaningful k for k-means clustering.

Elbow Method

A natural starting point is to use the within-cluster SSE (Sum of Squared Errors), as discussed in the previous entry 15. Clustering in Action: Exploring K-Means with Scikit-Learn. A straightforward approach might be to simply choose the k that minimizes SSE, assuming that this would immediately solve the problem. However, this approach is not as simple as it seems.

The SSE [also called Inertia (Géron, 2019)[1] and Distortion (Raschka et al., 2022)[2]] is not a reliable metric for determining the optimal number of clusters because it consistently decreases as k increases. This happens because adding more clusters reduces the distance between data points and their nearest centroids, naturally lowering the inertia value. As a result, inertia alone does not provide a meaningful criterion for selecting k, since a higher number of clusters will always lead to a smaller inertia, even if the clustering structure is not necessarily optimal.

However, we can still extract a useful insight from this. First, let's generate a synthetic dataset for instructional purposes.

Synthetic Dataset

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs

# Generate synthetic data
X, y = make_blobs(n_samples=500,
                  n_features=2,
                  centers=5,
                  cluster_std=2,
                  shuffle=True,
                  random_state=20)

# Function to visualize the data
def plot_data(X):
    plt.figure(figsize=(7, 5))
    plt.scatter(X[:, 0], X[:, 1],
    c='royalblue', marker='o',
    edgecolor='black', s=60, alpha=0.7)
    plt.xlabel('Feature 1', fontsize=12)
    plt.ylabel('Feature 2', fontsize=12)
    plt.title('Generated Data from make_blobs', fontsize=14, fontweight='bold')
    plt.grid(True, linestyle="--", alpha=0.6)
    plt.show()

# Call the visualization function
plot_data(X)
```

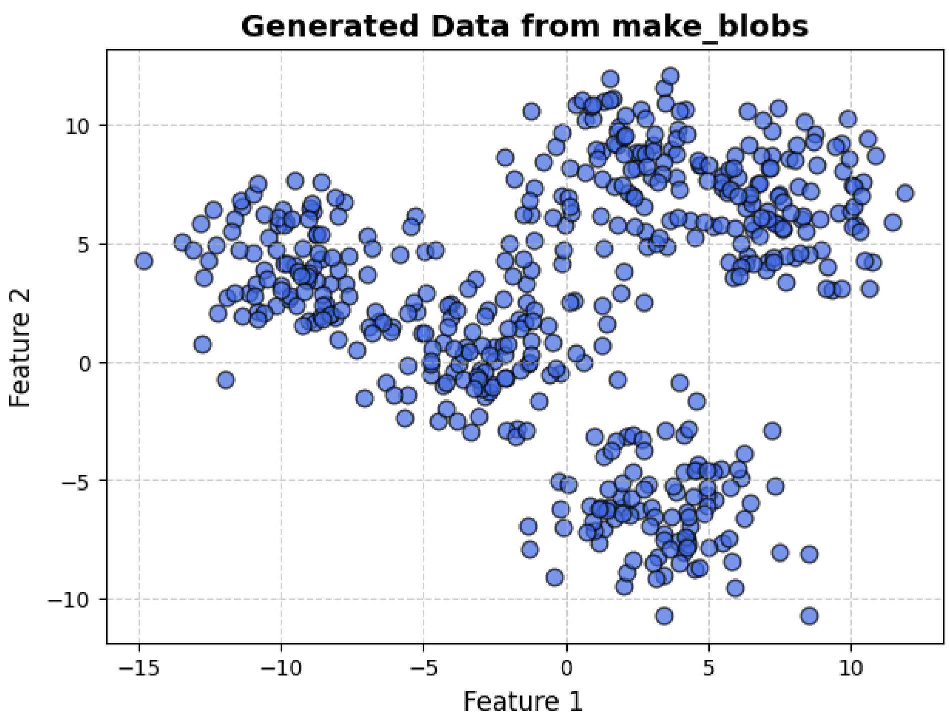

Figure 2: Five clusters

The fundamental concept of the elbow method is to determine the point at which increasing the number of clusters (k) no longer results in a significant reduction in distortion (or inertia). In other words, this method seeks to identify the optimal trade-off between minimizing within-cluster variance and avoiding an excessive number of clusters. As k increases, the distortion naturally decreases, but at some point, the rate of improvement slows down noticeably. This turning point, where adding more clusters yields only marginal benefits, is referred to as the "elbow" and is considered a strong candidate for the optimal number of clusters.

The elbow method

```
from sklearn.cluster import KMeans

inertias = []
k_values = range(1, 16)

for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(X)
    inertias.append(kmeans.inertia_)

# Graficar el método del codo
plt.figure(figsize=(8, 5))
plt.plot(k_values, inertias, marker='o', linestyle='-')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal k')
plt.xticks(k_values)
plt.grid(True)
plt.show()
```

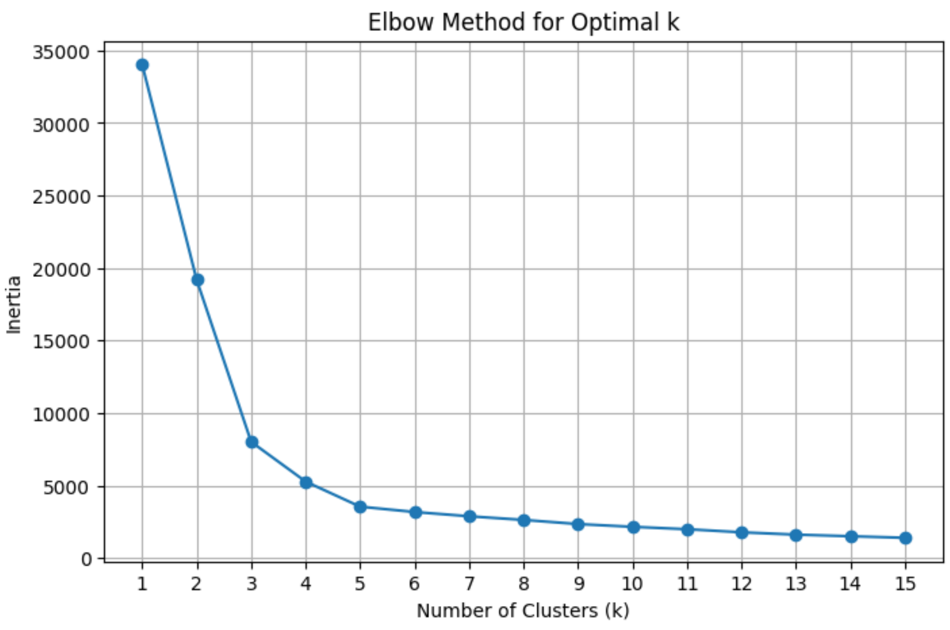

Figure 3: The elbow method applied

In Figure 3, we observe the characteristic elbow shape, with noticeable bending around k = 3, 4, and 5. As k increases beyond this range, the decrease in inertia becomes much more gradual. This indicates that adding more clusters provides diminishing improvements in clustering quality. Based on this pattern, k = 5 emerges as a strong candidate for the optimal number of clusters.

Silhouette Coefficient

Silhouette analysis is another internal metric used to assess the quality of a clustering solution. Unlike inertia, this method is not exclusive to k-means and can be applied to various clustering algorithms. It serves as both a numerical measure and a visual diagnostic tool, helping to determine how well-defined and compact the formed clusters are.

At its core, the silhouette score evaluates the degree to which an individual data point is correctly assigned to its designated cluster. This is done by comparing its cohesion—how closely it aligns with other points in the same cluster—with its separation from points in different clusters.

The silhouette score ranges from -1 to +1, where higher values indicate that a point is strongly associated with its own cluster and distinctly separate from others. Conversely, negative or low values suggest that the clustering structure may be suboptimal, possibly due to an excessive or insufficient number of clusters. A well-defined clustering configuration is characterized by most data points having high silhouette scores, signifying clear and well-separated groupings.

Let's assume the data have been clustered via any technique into k clusters. For data point $i \in C_I$ let

$$a(i) = \frac{1}{|C_I| - 1} \sum_{j \in C_I, i \neq j} d(i,j)$$

be the mean distance between i and all other data points in the same cluster (we exclude de case $i = i$).

We define the mean dissimilarity of point i to some cluster C_J as the mean distance from i to all points in C_J (where C_J). For each data point $i \in C_I$, we define

$$b(i) = \min_{J \neq I} \frac{1}{|C_J|} \sum_{j \in C_J} d(i,j)$$

to be the smallest mean distance of i to all points in any other cluster.

We now define the silhouette (value) of one data point i

$$s(i) = \begin{cases} \frac{b(i)-a(i)}{\max\{a(i),b(i)\}} \text{ if } |C_I| > 1 \\ 0 \text{ if } a(i) = b(i) \end{cases}$$

We can see that we get close to an ideal silhouette coefficient of 1 if $b(i) >> a(i)$

Silhouette score

```python
from sklearn.metrics import silhouette_score

silhouettes = []
k_values = range(2, 16)

for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(X)
    silhouettes.append(silhouette_score(X, kmeans.labels_))

plt.figure(figsize=(8, 5))
plt.plot(k_values, silhouettes, marker='o', linestyle='-')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Silhouette score')
plt.title('Silhouette')
plt.xticks(k_values)
plt.grid(True)
plt.show()
```

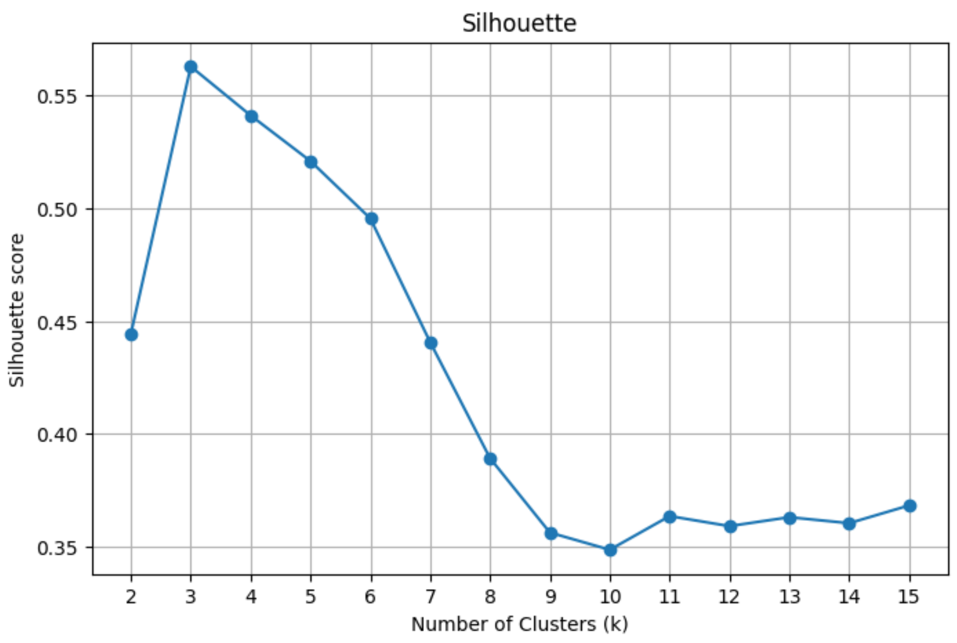

Figure 4: Silhouette scores

In this case, we observe that the highest silhouette scores correspond to k = 3, 4, and 5, in decreasing order. While the silhouette analysis suggests that k = 3 is the optimal choice, we can refine our decision by incorporating insights from the elbow method, providing additional support for selecting the most suitable number of clusters.

A more insightful way to analyze clustering quality is through a silhouette diagram, which visually represents the silhouette coefficients of individual data points. In this plot, each instance is arranged based on the cluster it belongs to and is further sorted by its silhouette score. This ordering provides a clear depiction of how well-defined each cluster is.

A key feature of the silhouette diagram is the vertical dashed line, which indicates the average silhouette score for a given number of clusters. Ideally, most points within a cluster should extend beyond this reference line, confirming that they are well-separated from other clusters. However, if a significant number of points fall short of this threshold (appearing to the left of the dashed line), it suggests that the cluster is poorly formed, meaning its points are too close to neighboring clusters, reducing the overall clustering effectiveness.

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score

def plot_silhouette_diagram(X, k_values):
    """Generates silhouette diagrams for the specified k values in k_values."""
    fig, axes = plt.subplots(1, len(k_values), figsize=(18, 5))

    for i, k in enumerate(k_values):
        # Apply K-Means with k clusters
        kmeans = KMeans(n_clusters=k, random_state=42, n_init='auto')
        labels = kmeans.fit_predict(X)

        # Compute silhouette coefficients
        silhouette_vals = silhouette_samples(X, labels)
        avg_silhouette = silhouette_score(X, labels)

        # Sort silhouette values within each cluster
        y_lower = 10  # Initial y-axis position for plotting clusters
        axes[i].set_xlim([-0.1, 1])  # X-axis limits from -0.1 to 1

        for cluster in np.unique(labels):
            cluster_silhouettes = silhouette_vals[labels == cluster]
            cluster_silhouettes.sort()
            size_cluster = cluster_silhouettes.shape[0]
            y_upper = y_lower + size_cluster

            # Color the silhouette plot for each cluster
            axes[i].fill_betweenx(np.arange(y_lower, y_upper), 0, cluster_silhouettes,
            ↪  alpha=0.7)
            axes[i].text(-0.05, y_lower + 0.5 * size_cluster, str(cluster))  # Cluster
            ↪  labels
            y_lower = y_upper + 10  # Space between clusters

        # Draw a vertical line representing the average silhouette score
        axes[i].axvline(avg_silhouette, color='red', linestyle='--')

        # Labels and titles
        axes[i].set_title(f'Silhouette Diagram (k = {k})')
        axes[i].set_xlabel('Silhouette Coefficient')
        axes[i].set_ylabel('Cluster')
        axes[i].grid(True)

    plt.tight_layout()
    plt.show()
```

```python
# Call the function with k = 3, 4, and 5
plot_silhouette_diagram(X, k_values=[3, 4, 5])
```

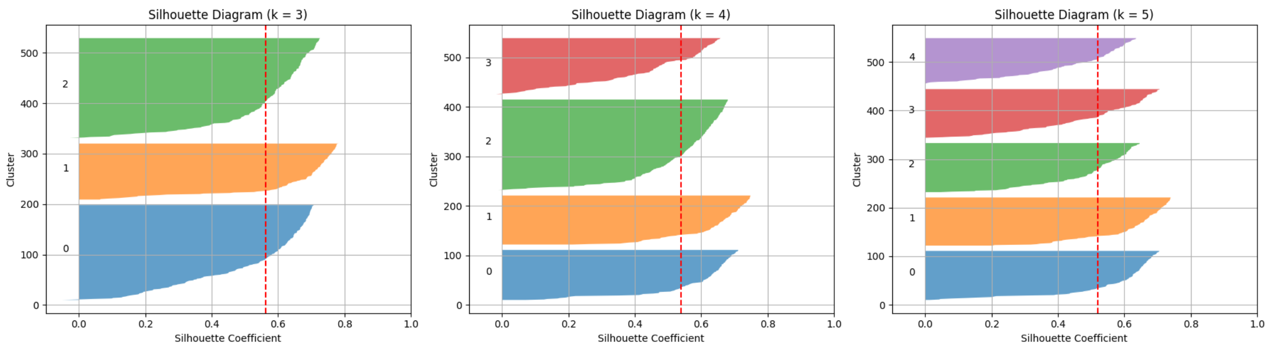

Figure 5: Silhouettes diagrams for k = 3, 4 and 5

Davies-Bouldin Index

The Davies-Bouldin Index (DBI) is a statistical measure designed to assess the effectiveness of clustering results by analyzing the degree of separation and compactness within the clusters formed by a given algorithm. This index operates under the fundamental assumption that an ideal clustering solution should exhibit two key characteristics:

- Compactness – Each cluster should contain data points that are closely packed together, minimizing intra-cluster variance.
- Separation – Clusters should be well-distinguished from one another, ensuring that their centroids are as distant as possible.

The DBI accomplishes this by quantifying the ratio between intra-cluster dispersion and inter-cluster distance. In other words, it calculates how much each cluster spreads out in comparison to how far apart different clusters are. A lower DBI value indicates a more optimal clustering structure, as it suggests that the clusters are internally cohesive while remaining well-separated from one another. Conversely, a higher DBI value signifies poor clustering, implying that the clusters either overlap significantly or contain high internal dispersion.

The Davies-Bouldin Index for k clusters can be calculated as

$$DBI = \frac{1}{k}\sum_{I=1}^{K}\max\{\frac{\Delta(C_I)+\Delta(C_J)}{\delta(C_I,C_J)}\}$$

Where $\Delta(C_K)$ is the intracluster distnace within the cluster C_K, and $\delta(C_I, C_J)$ is the intercluster distance between the clusters C_I and C_J

The Davies-Bouldin index is very effective compared to other clustering evaluation metrics because:

- It is flexible and works for any number of clusters.
- It makes no assumptions about the shape of the clusters.
- It is easy to use and intuitive.

Davies-Bouldin Index

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.metrics import davies_bouldin_score

# Define range of k values
k_values = range(2, 16)  # DBI is not defined for k=1
```

```python
# Store Davies-Bouldin Index scores
dbi_scores = []

# Compute DBI for each k
for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    labels = kmeans.fit_predict(X)
    dbi_scores.append(davies_bouldin_score(X, labels))

# Plot Davies-Bouldin Index vs. Number of Clusters
plt.figure(figsize=(8, 5))
plt.plot(k_values, dbi_scores, marker='o', linestyle='-', color='b')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Davies-Bouldin Index')
plt.title('Davies-Bouldin Index for Different k Values')
plt.xticks(k_values)
plt.grid(True)
plt.show()
```

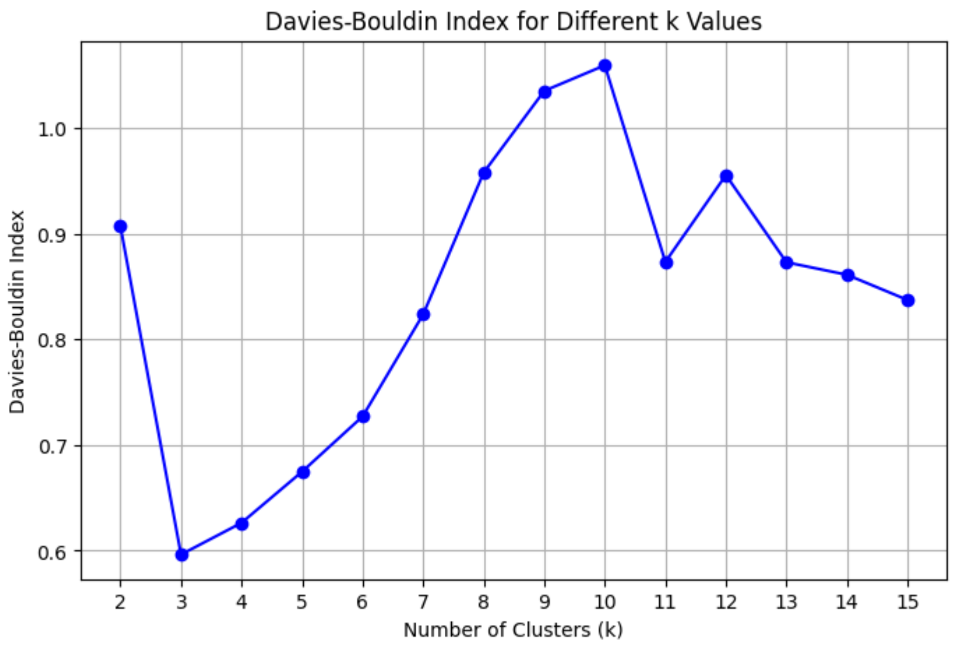

Figure 6: DBI for different k values

Based on the Davies-Bouldin Index results, we once again observe that our top candidates for k are 3, 4, and 5, ranked from best to worst in terms of cluster separation and compactness. This finding aligns with our previous analyses, reinforcing the idea that these values of k are the most promising options.

Now that we have performed three different validation tests—the Elbow Method, the Silhouette Score, and the Davies-Bouldin Index—it is time to visualize how the clustering results look for our two strongest candidates:

- k = 3, which emerged as the best choice according to the Silhouette Score and the Davies-Bouldin Index due to its balance between cohesion and separation.
- k = 5, which was favored by the Elbow Method and also happens to match the original number of clusters in our dataset.

By examining these two configurations, we can gain a deeper understanding of how well the data is grouped and make a more informed final decision.

The k-means models

The models

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Define k values for analysis
k_values = [3, 5]

# Create subplots for both k=3 and k=5
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

for i, k in enumerate(k_values):
    # Train K-Means model
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    labels = kmeans.fit_predict(X)
    centroids = kmeans.cluster_centers_

    # Plot clusters
    axes[i].scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', edgecolor='k', alpha=0.6)
    axes[i].scatter(centroids[:, 0], centroids[:, 1], marker='X', s=200, c='red',
    ↪  label='Centroids')

    # Formatting
    axes[i].set_title(f'K-Means Clustering (k={k})')
    axes[i].set_xlabel('Feature 1')
    axes[i].set_ylabel('Feature 2')
    axes[i].legend()
    axes[i].grid(True)

# Show the final plot
plt.tight_layout()
plt.show()
```

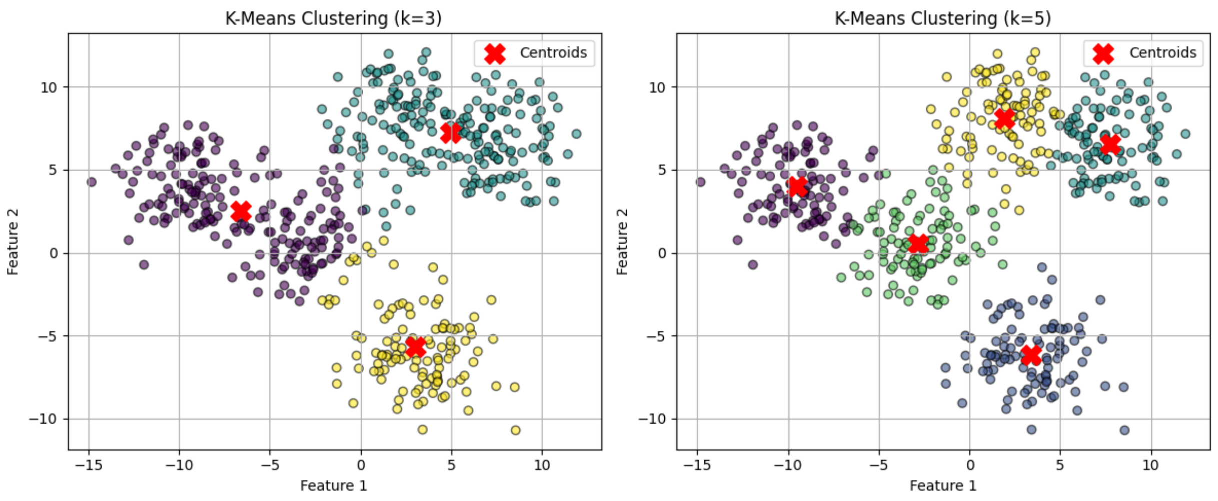

Figure 7: “3-means and 5-means”

References

[1] Géron, A. (2019). *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow.* O'Reilly

[2] Raschka, S., Liu, Y. & Mirjalili, V. (2022). *Machine Learning with PyTorch and Scikit-Learn.* Packt.

[3] Rodríguez, D. (2023). *El índice de Davies-Bouldin para estimar los clústeres en k-means e implementación en Python.* [The Davies-Bouldin index for estimating clusters in k-means and its implementation in Python]. Analytics Lane. Available at: https://www.analyticslane.com/2023/06/30/el-indice-de-davies-bouldinen-para-estimar-los-clusteres-en-k-means-e-implementacion-en-python/