K-Means Clustering Tutorial

Student Name: Giridhar Reddy Goddilla
Student ID: 23067661
Module: Machine Learning and Neural Networks
Assignment Type: Individual Report
Submission Date: 27 March 2025
Tutor: Peter Scicluna
GitHub link:https://github.com/Giridharreddygoddilla/K_Means_Clustering_Tutorial

K-Means Clustering Tutorial

Table of Contents

K-Means Clustering Tutorial

Introduction:

1.1 What is K-Means Clustering?

K-Means is an unsupervised learning algorithm that divides data into K distinct clusters. Each cluster groups similar data points together, minimizing the variance within each group. The algorithm aims to find patterns in the data without any predefined labels, making it ideal for discovering natural groupings.

1.2 Why Use K-Means Clustering?

K-Means is favoured for its:

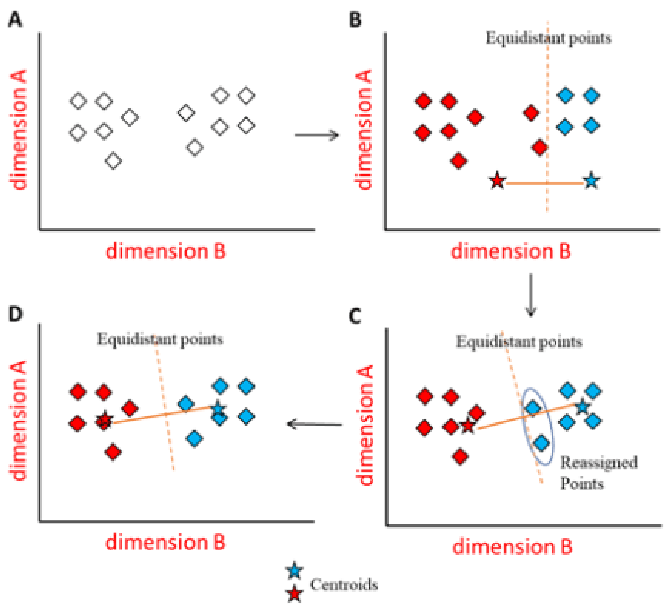
- **Simplicity:** Easy to understand and implement.
- **Scalability:** Works well with large datasets.
- **Efficiency:** Converges quickly, making it a good choice for practical applications.

It's commonly used in areas like customer segmentation, image compression, and anomaly detection.

1.3 How K-Means Works: A Conceptual Overview

K-Means starts by randomly selecting K centroids. Each data point is assigned to the nearest centroid. After assignment, the centroids are recalculated as the mean of the points in each cluster. The process repeats until the centroids stabilize, meaning the algorithm has converged and the clusters are finalized.

Explaining the Plot



- **Panel A:**
  Initially, data points are scattered, with no clusters yet formed.

- **Panel B:**
  K centroids are randomly placed, and points are assigned to the nearest centroid. The points are coloured according to their assigned cluster.

- **Panel C:**
  The centroids are recalculated by averaging the points in each cluster. Some points may switch clusters if they are closer to a new centroid.

- **Panel D:**
  The process repeats until the centroids no longer move significantly, achieving stable clusters.

This process shows how K-Means iteratively refines the clusters, grouping data points around their centroids.

2. The K-Means Algorithm: Foundations and Core Concepts

K-Means is an iterative **unsupervised learning** algorithm that clusters data points into **K** groups. The goal is to minimize the **variance** within each cluster. Here's how it works:

2.1 Step-by-Step Process of K-Means

1. **Initialization**: Randomly select **K centroids** as the initial cluster centers.
2. **Assignment**: Assign each data point to the nearest centroid using a distance metric (typically **Euclidean distance**).
3. **Recalculation**: Update the centroids to the mean of the points in each cluster.
4. **Reassignment**: Reassign points to the nearest centroid after the update.
5. **Convergence**: Repeat the process until the centroids stabilize, meaning the algorithm has converged.

2.2 Distance Metrics in K-Means

The core of the K-Means algorithm is based on **distance** how far apart data points are from each other. The algorithm typically uses **Euclidean distance**, but there are other options as well:

- **Euclidean Distance**:
  This is the most commonly used distance metric in K-Means. It calculates the straight-line distance between two points in multi-dimensional space. It's intuitive and works well in most cases, defined as:

$$d(x,y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

  Where x and y are two data points and n is the number of features.

- **Manhattan Distance**:
  Also known as **L1 distance**, it calculates the sum of the absolute differences between corresponding coordinates. This is especially useful when the data is on a grid or when features have discrete values.

$$d(x,y) = \sum_{i=1}^{n} |x_i - y_i|$$

  where x and y are two data points, and n is the number of features. Unlike Euclidean distance, Manhattan distance sums the absolute differences along each dimension.

K-Means Clustering Tutorial

- **Minkowski Distance**:
  This is a generalization of both Euclidean and Manhattan distances. It introduces a parameter p to adjust the distance measure:

$$d(x, y) = \left(\sum_{i=1}^{n} |\, x_i - y_{i|^p} \right)^{1/p}$$

If p=1, this becomes Manhattan distance; if p=2, it becomes Euclidean distance.

2.3 Finding Optimal Clusters

K-Means minimizes **inertia** (or **WCSS**), the sum of squared distances from each point to its assigned centroid. The goal is to have compact, well-separated clusters, with low inertia indicating effective clustering.

2.4 Convergence

K-Means converges when:

- Centroids no longer change significantly.
- Cluster assignments stay the same.

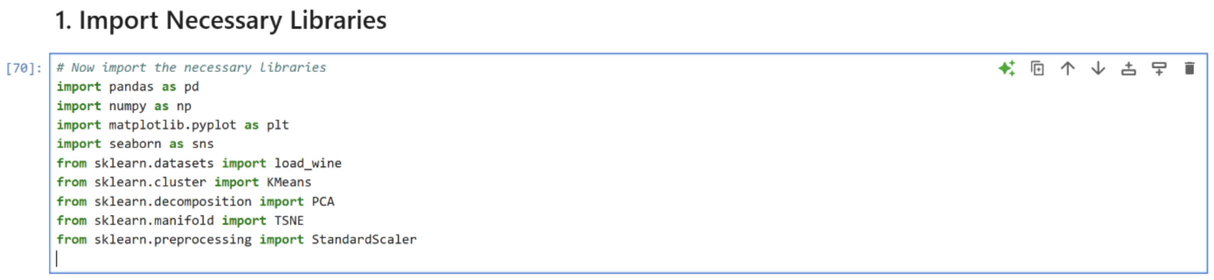
At this point, the algorithm stops. However, K-Means may converge to a suboptimal solution, which is why **K-Means++** is often used to improve centroid initialization.

3. Building and Evaluating a K-Means Clustering Model

In this section, we perform clustering analysis on the Wine dataset using K-Means clustering, followed by visualizations to interpret the clustering results.

3.1 Import Necessary Libraries:

We begin by importing essential libraries:

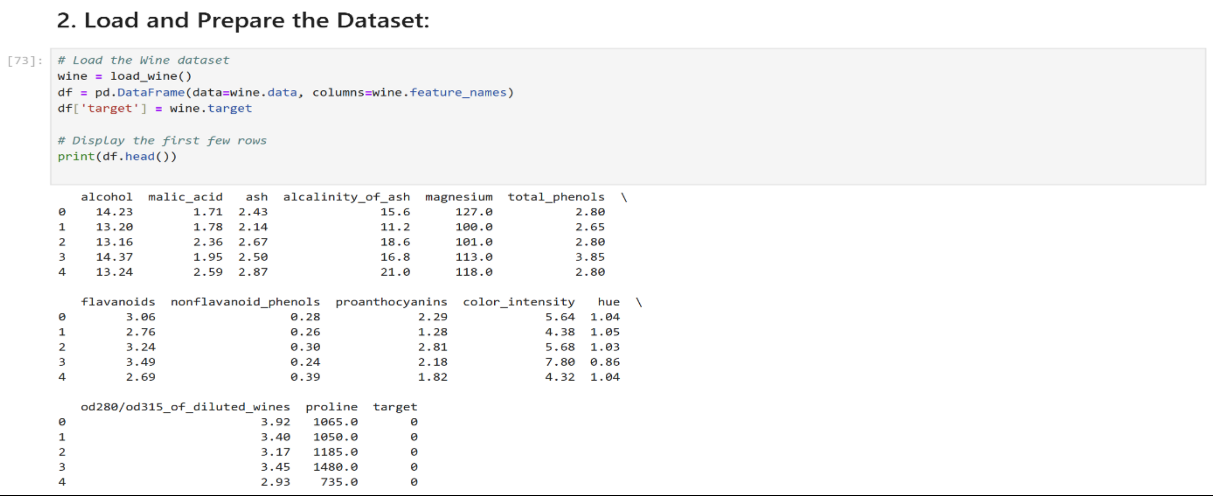


1. Import Necessary Libraries

[70]:
```
# Now import the necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_wine
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.preprocessing import StandardScaler
```

3.2 Load and Prepare the Dataset:

Here, we load the Wine dataset and standardize its features to ensure each contributes equally to the clustering process:



2. Load and Prepare the Dataset:

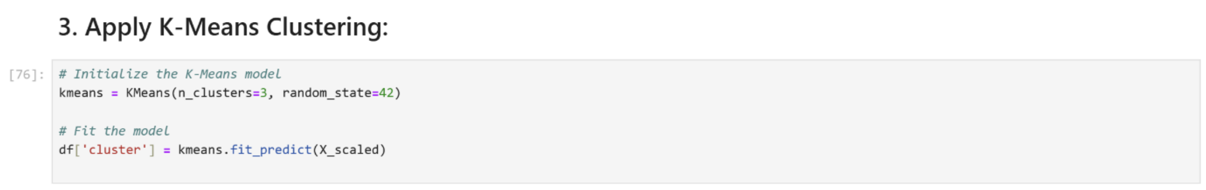
[73]:
```
# Load the Wine dataset
wine = load_wine()
df = pd.DataFrame(data=wine.data, columns=wine.feature_names)
df['target'] = wine.target

# Display the first few rows
print(df.head())
```

```
   alcohol  malic_acid   ash  alcalinity_of_ash  magnesium  total_phenols  \
0    14.23        1.71  2.43               15.6      127.0           2.80   
1    13.20        1.78  2.14               11.2      100.0           2.65   
2    13.16        2.36  2.67               18.6      101.0           2.80   
3    14.37        1.95  2.50               16.8      113.0           3.85   
4    13.24        2.59  2.87               21.0      118.0           2.80   

   flavanoids  nonflavanoid_phenols  proanthocyanins  color_intensity   hue  \
0        3.06                  0.28             2.29             5.64  1.04   
1        2.76                  0.26             1.28             4.38  1.05   
2        3.24                  0.30             2.81             5.68  1.03   
3        3.49                  0.24             2.18             7.80  0.86   
4        2.69                  0.39             1.82             4.32  1.04   

   od280/od315_of_diluted_wines  proline  target  
0                          3.92   1065.0       0  
1                          3.40   1050.0       0  
2                          3.17   1185.0       0  
3                          3.45   1480.0       0  
4                          2.93    735.0       0  
```

K-Means Clustering Tutorial

3.3 Apply K-Means Clustering:

We apply K-Means clustering to partition the dataset into three clusters, corresponding to the three wine cultivars:



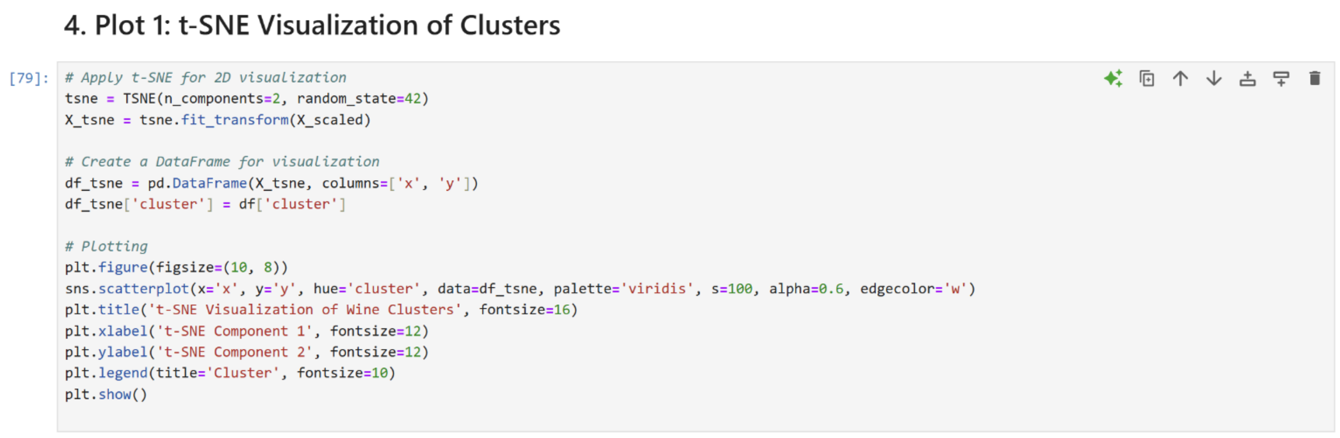
3. Apply K-Means Clustering:

[76]:

```python
# Initialize the K-Means model
kmeans = KMeans(n_clusters=3, random_state=42)

# Fit the model
df['cluster'] = kmeans.fit_predict(X_scaled)
```

3.4 Plot 1: t-SNE Visualization of Clusters

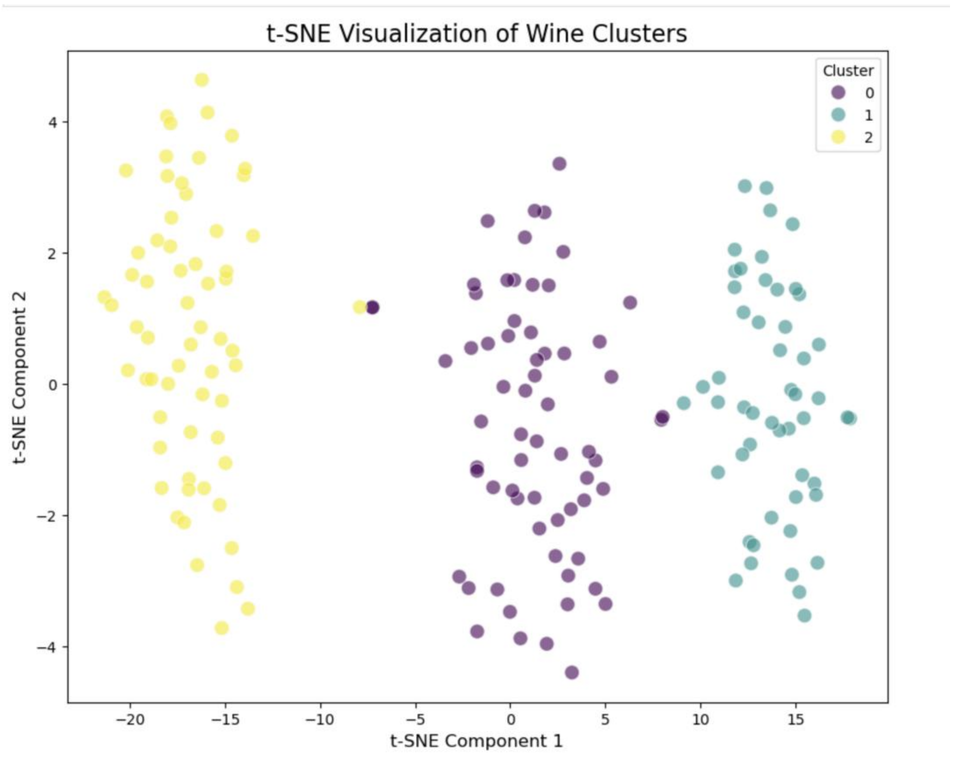
We use t-SNE to reduce the dataset's dimensionality to two components for visualization, coloring the points by their cluster assignments:



4. Plot 1: t-SNE Visualization of Clusters

[79]:

```python
# Apply t-SNE for 2D visualization
tsne = TSNE(n_components=2, random_state=42)
X_tsne = tsne.fit_transform(X_scaled)

# Create a DataFrame for visualization
df_tsne = pd.DataFrame(X_tsne, columns=['x', 'y'])
df_tsne['cluster'] = df['cluster']

# Plotting
plt.figure(figsize=(10, 8))
sns.scatterplot(x='x', y='y', hue='cluster', data=df_tsne, palette='viridis', s=100, alpha=0.6, edgecolor='w')
plt.title('t-SNE Visualization of Wine Clusters', fontsize=16)
plt.xlabel('t-SNE Component 1', fontsize=12)
plt.ylabel('t-SNE Component 2', fontsize=12)
plt.legend(title='Cluster', fontsize=10)
plt.show()
```

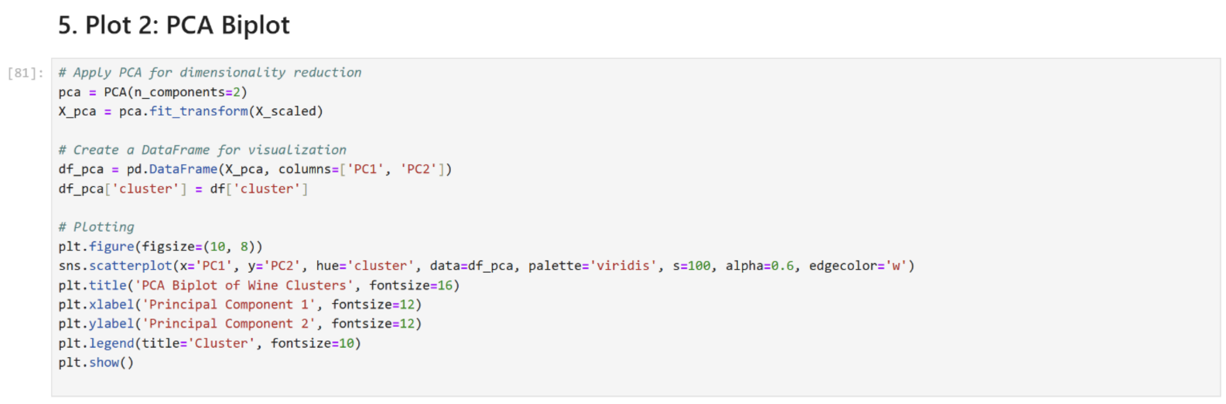


This t-SNE plot visualizes the K-Means clustering results on the Wine dataset. Each point represents a wine sample, color-coded by cluster (purple, teal, yellow). The clear separation of clusters shows that K-Means effectively grouped the wines, with t-SNE reducing the high-dimensional data to two components for easy visualization.

K-Means Clustering Tutorial

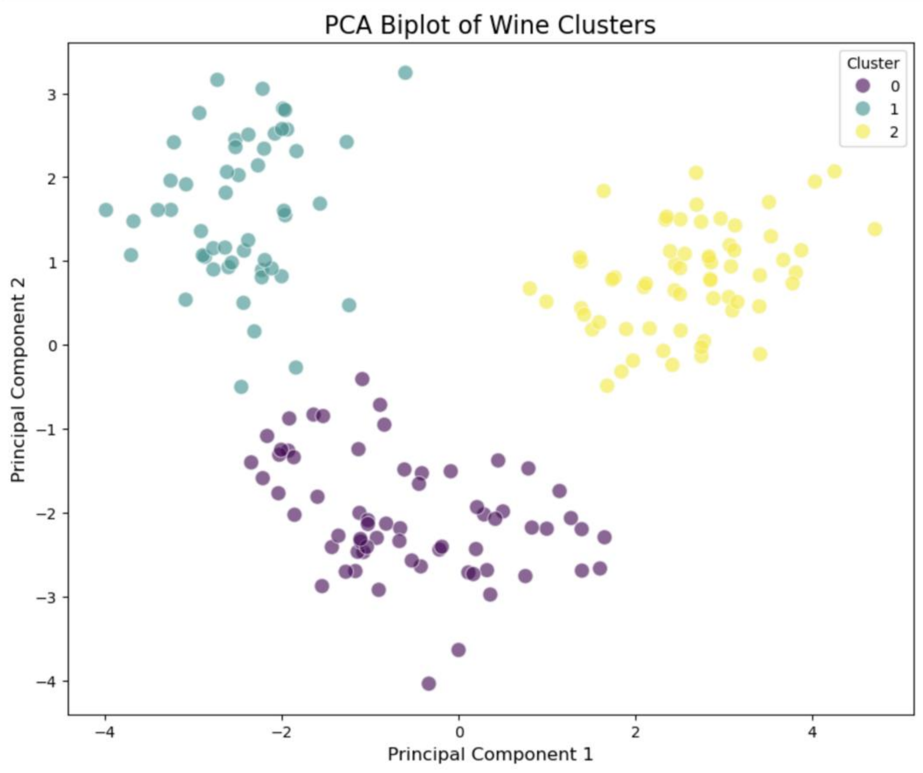
3.5 Plot 2: PCA Biplot

PCA is applied to reduce the dataset to two principal components, and a biplot is created to visualize the clustering in this reduced space:



5. Plot 2: PCA Biplot

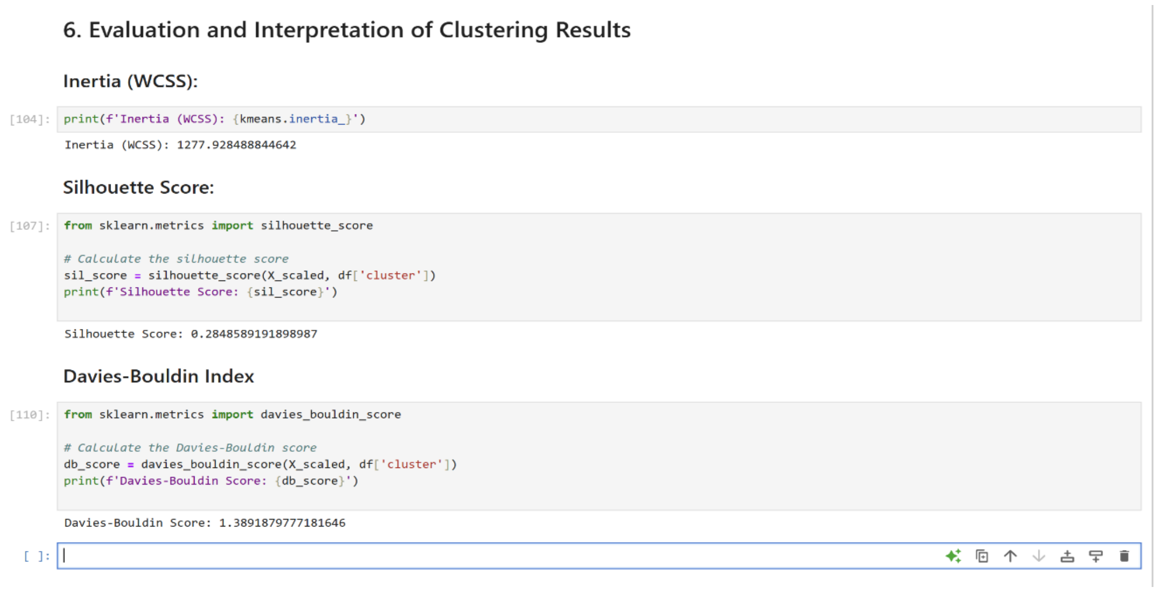
```
[81]: # Apply PCA for dimensionality reduction
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Create a DataFrame for visualization
df_pca = pd.DataFrame(X_pca, columns=['PC1', 'PC2'])
df_pca['cluster'] = df['cluster']

# Plotting
plt.figure(figsize=(10, 8))
sns.scatterplot(x='PC1', y='PC2', hue='cluster', data=df_pca, palette='viridis', s=100, alpha=0.6, edgecolor='w')
plt.title('PCA Biplot of Wine Clusters', fontsize=16)
plt.xlabel('Principal Component 1', fontsize=12)
plt.ylabel('Principal Component 2', fontsize=12)
plt.legend(title='Cluster', fontsize=10)
plt.show()
```



This PCA biplot visualizes the K-Means clustering results on the Wine dataset after reducing the data to two principal components. Each wine sample is color-coded by cluster (purple, teal, yellow), with clear separation between clusters. The plot demonstrates how PCA captures the main variance, effectively distinguishing the clusters in 2D space.

The clustering analysis using K-Means, along with dimensionality reduction techniques like t-SNE and PCA, effectively reveals distinct patterns in the Wine dataset. These visualizations help in understanding how the wines are grouped based on their features.

4. Evaluation and Interpretation of Clustering Results



6. Evaluation and Interpretation of Clustering Results

Inertia (WCSS):

```
[104]: print(f'Inertia (WCSS): {kmeans.inertia_}')
```

```
Inertia (WCSS): 1277.928488844642
```

Silhouette Score:

```
[107]: from sklearn.metrics import silhouette_score

       # Calculate the silhouette score
       sil_score = silhouette_score(X_scaled, df['cluster'])
       print(f'Silhouette Score: {sil_score}')
```

```
Silhouette Score: 0.2848589191898987
```

Davies-Bouldin Index

```
[110]: from sklearn.metrics import davies_bouldin_score

       # Calculate the Davies-Bouldin score
       db_score = davies_bouldin_score(X_scaled, df['cluster'])
       print(f'Davies-Bouldin Score: {db_score}')
```

```
Davies-Bouldin Score: 1.3891879777181646
```

```
[ ]:
```

In this section, we also implemented and evaluated the clustering results, as shown in the image above, using three key metrics:

1. **Inertia (WCSS):**
   This metric, which measures the compactness of the clusters, was calculated and showed an inertia value of 1277.93. This suggests that the data points are relatively close to their centroids, although there's room for further refinement in cluster tightness.
2. **Silhouette Score:**
   We calculated the silhouette score, which returned a value of 0.28. This indicates that while the clusters are somewhat separated, the separation could be improved for more distinct groupings.
3. **Davies-Bouldin Index:**
   The Davies-Bouldin score, calculated as 1.39, suggests that the clusters are moderately well-separated. Lower values would indicate more distinct clusters.

These metrics were implemented and calculated to provide a quantitative assessment of how well the K-Means algorithm performed in clustering the data, as demonstrated in the above image.

5. Real-World Applications of K-Means Clustering

K-Means clustering is a widely used algorithm with applications across various fields due to its simplicity and efficiency in grouping similar data points.

5.1 Customer Segmentation

K-Means is used in **customer segmentation** to group customers based on purchasing behaviour, enabling businesses to target specific groups with personalized marketing.

- **Example:** Retailers can segment customers into high-value or frequent shoppers to offer tailored promotions.

5.2 Image Compression

In **image compression**, K-Means reduces the number of colors in an image, decreasing its size while maintaining visual quality.

- **Example:** Digital images are compressed by clustering similar colors, reducing file sizes for easier storage and sharing.

5.3 Anomaly Detection

K-Means is applied in **anomaly detection** to identify outliers by clustering normal data and flagging points that don't fit.

- **Example:** In financial systems, it detects fraudulent transactions by identifying behaviour outside normal patterns.

6. Best Practices and Model Transparency

To ensure effective K-Means clustering, follow these best practices:

6.1 Ensuring Quality Clusters

- Choosing the Right K: Use the Elbow Method or Silhouette Score to determine the optimal number of clusters.

6.2 Improving Initialization with K-Means++

- K-Means++ improves centroid initialization by spreading centroids more evenly, reducing the risk of poor results.

"kmeans = KMeans (n_clusters=3, init='k-means++', random_state=42)"

6.3 Evaluating Cluster Quality

- **Inertia** and **Silhouette Score** are key metrics to assess clustering quality. Lower inertia and higher silhouette scores indicate well-defined clusters.

6.4 Visualizing Results

- **t-SNE and PCA** help validate and interpret clusters, ensuring the results are meaningful and easy to understand.

By following these best practices, you ensure that K-Means clustering produces reliable, interpretable results

7. Challenges and Limitations of K-Means Clustering

While K-Means is effective, it has limitations:

7.1 Sensitivity to Initial Centroids

K-Means is sensitive to the initial placement of centroids, which can lead to suboptimal clusters. **K-Means++** initialization helps address this.

7.2 Handling Non-Spherical Clusters

K-Means struggles with clusters that are not spherical or vary in density. Alternatives like **DBSCAN** or **GMM** are better for such data.

7.3 Scalability with Large Datasets

K-Means can become slow with large datasets. **MiniBatchKMeans** can help scale the algorithm to handle bigger datasets more efficiently.

7.4 Choosing the Optimal K

Determining the right number of clusters can be challenging. Methods like the **Elbow Method** and **Silhouette Score** can guide the selection of K.

8. Ensuring Accessibility in K-Means Clustering

Ensuring accessibility in K-Means clustering helps improve collaboration and decision-making for all users, including those with disabilities.

8.1 Accessible Visualizations

Use **colour-blind-friendly palettes** and **alternative text** for charts to make visualizations accessible to everyone.

8.2 Clear Result Interpretation

Simplify explanations of metrics like inertia and silhouette score, avoiding jargon to make results understandable for all users.

8.3 Interactive Visuals

Interactive tools like **Plotly** allow users to explore clustering results dynamically, making the model more accessible and user-friendly.

9. Exploring Future Enhancements and Opportunities

K-Means clustering is a powerful tool, but there are always opportunities to improve and explore new techniques. Here are some future enhancements and areas for further exploration:

9.1 Incorporating Other Clustering Algorithms

While K-Means is widely used, exploring other clustering algorithms like **DBSCAN**, **Agglomerative Clustering**, or **Gaussian Mixture Models (GMM)** can help address some of K-Means' limitations, such as handling non-spherical clusters and varying densities.

9.2 Using Hybrid Approaches

Combining K-Means with other techniques, such as **PCA for dimensionality reduction** or **supervised learning models**, can lead to more accurate clustering results. Hybrid approaches can leverage the strengths of multiple algorithms for more robust performance.

9.3 Optimizing with Advanced Techniques

Applying more advanced techniques like **K-Means with Constraints**, or integrating **deep learning-based clustering** (e.g., autoencoders) can further improve clustering performance, especially on complex and high-dimensional datasets.

10. Conclusion

K-Means clustering is a widely used and effective unsupervised learning algorithm that groups similar data points into clusters. Throughout this tutorial, we explored how K-Means works, applied it to the Wine dataset, and evaluated the clustering results using visualization techniques like **t-SNE** and **PCA**, along with key metrics such as **inertia**, **silhouette score**, and **Davies-Bouldin index**.

Key Takeaways:

- **Simplicity and Efficiency:** K-Means is easy to implement and performs well with large datasets.
- **Evaluation is Crucial:** Using both visualizations and metrics ensures that the clustering results are meaningful and reliable.
- **Limitations:** K-Means has challenges with non-spherical clusters and sensitivity to initial centroids.
- **Future Exploration:** Exploring other clustering algorithms and advanced techniques can help address K-Means' limitations and improve performance.

In summary, K-Means is an effective tool for clustering, and by using proper evaluation techniques, it can provide valuable insights into data. However, it’s important to consider its limitations and explore alternatives when necessary.

11. References

- **scikit-learn Documentation**
  Comprehensive guide to implementing K-Means clustering and other algorithms.
  Available at: https://scikit-learn.org/stable/modules/clustering.html#k-means

- **Bishop, C. M. (2006).** *Pattern Recognition and Machine Learning.*
  Covers machine learning algorithms, including clustering techniques.
  Springer, 2006.

- **Hastie, T., Tibshirani, R., & Friedman, J. (2009).** *The Elements of Statistical Learning.*
  A key resource on machine learning methods with a focus on clustering.
  Springer, 2009.

- **MacQueen, J. (1967).** *Some Methods for Classification and Analysis of Multivariate Observations.*
  Introduced the K-Means clustering algorithm.
  Proceedings of the 5th Berkeley Symposium, 1967.

- **Sculley, D. (2010).** *Web-Scale K-Means Clustering.*
  Optimizations for using K-Means on large datasets.
  International Conference on WWW, 2010.

- **De Moura, R., & Estivill-Castro, V. (2004).** *Improved K-Means Clustering Algorithm.*
  Enhancements to K-Means for better centroid initialization and convergence.
  CVPR, 2004.

- **Blopig (2020).** *K-Means Clustering Made Simple.*
  Introductory image sourced from Blopig.

K-Means Clustering Tutorial

Available at: https://www.blopig.com/blog/2020/07/k-means-clustering-made-simple/

- **Wine Dataset**
  The Wine dataset used for K-Means clustering is available from the UCI Machine Learning Repository.
  UCI Machine Learning Repository - Wine Dataset