110705067 洪薇欣 HW1

Methods

1. Synthesized and Toy Datasets：

a. make_blobs (300 samples, 4 centers, cluster_std=0.60)**:**

- **Generates points around specified centers with Gaussian noise.** Each cluster is spherical and **well-separated**.
- Ideal for benchmarking clustering algorithms with spherical assumptions.

b. make_moons (300 samples, noise=0.05)**:**

- Generates **two interleaving half-circle shapes (moons), not linearly separable.**
- Useful to test clustering algorithms that can handle non-convex clusters.

C. Iris Dataset (150 samples, 4 features, 3 classes)**:**

- **Real-world dataset** of iris flowers with 4 features and 3 classes.
- Benchmark for clustering or classification tasks. It contains some overlapping classes.

2. Clustering Algorithms：

a. KMeans:

- **Partitions data into k clusters by minimizing the sum of squared distances between data points and their nearest cluster centroid**.
- Fast and effective on well-separated, spherical clusters.
- **Fails on non-convex** clusters (e.g., moons); sensitive to initialization.

b. MiniBatchKMeans:

- A variant of KMeans using **small, random subsets (mini-batches) of the dataset to speed up computation.**
- Much faster on large datasets.
- Less accurate than KMeans on small data; still inherits KMeans' geometric assumptions.

c. DBSCAN (Density-Based Spatial Clustering of Applications with Noise):

- **Groups together points that are densely packed**; points in low-density regions are labeled as noise.
- Handles arbitrary shapes and can detect outliers automatically.
- Struggles with varying densities; sensitive to eps and min_samples.

d. Agglomerative Clustering (Hierarchical):

- **Starts with each point as its own cluster and merges them bottom-up using linkage criteria** (e.g., Ward minimizes variance)**.**
- Doesn’t need centroids; builds a dendrogram (cluster tree).
- Computationally expensive for large datasets; no "natural" number of clusters unless you specify n_clusters.

e. Spectral Clustering:

- **Constructs a similarity graph of data, computes the graph Laplacian, and uses eigenvectors to embed data in lower dimensions before clustering.**

- Excellent at identifying non-convex clusters, like moons.
- **May fail if graph is disconnected**; sensitive to similarity metric.

f. Gaussian Mixture Models (GMM):

- **Assumes data is generated from a mixture of Gaussian distributions; uses Expectation-Maximization (EM) to assign probabilities to cluster membership.**
- Soft clustering—**a point can belong partially to multiple clusters.**
- **Assumes data follows a Gaussian distribution**; can overfit or underperform on non-Gaussian shapes.

3. Evaluation Metrics

External Metrics (require true labels):

a. Adjusted Rand Index (ARI):

- Measures **how closely predicted clusters match true labels,** adjusted for chance**.** Best = 1.
- Handles permutations and chance groupings well.

b. Normalized Mutual Information (NMI):

- Measures **shared information between predicted and true labels.** Best = 1.
- Not affected by label ordering; interpretable scale from 0 to 1.

Internal Metrics (no true labels required):

a. Silhouette Score:

- Measures cohesion and separation. Measures **how similar a point is to its own cluster vs others.**
- Range: [-1, 1], **higher is better.**
- Very intuitive, good overall quality check.

b. Calinski-Harabasz Index:

- Measures **ratio of between-cluster variance to within-cluster variance**.
- Higher is better. Fast to compute.

c. Davies-Bouldin Index:

- Measures average **similarity between each cluster and its most similar one** (closest neighbor).
- Lower is better.

Experiments

Task 1: Experiment with existing clustering algorithms

a. Clustering algorithm and parameters:

- KMeans(n_clusters=4, random_state=0)
- MiniBatchKMeans(n_clusters=4, random_state=0)
- DBSCAN(eps=0.3, min_samples=5)
- AgglomerativeClustering(n_clusters=4, linkage='ward')
- SpectralClustering(n_clusters=4, affinity='nearest_neighbors', n_neighbors=10, random_state=0)
- GaussianMixture(n_components=4, random_state=0)

b. **Results:**

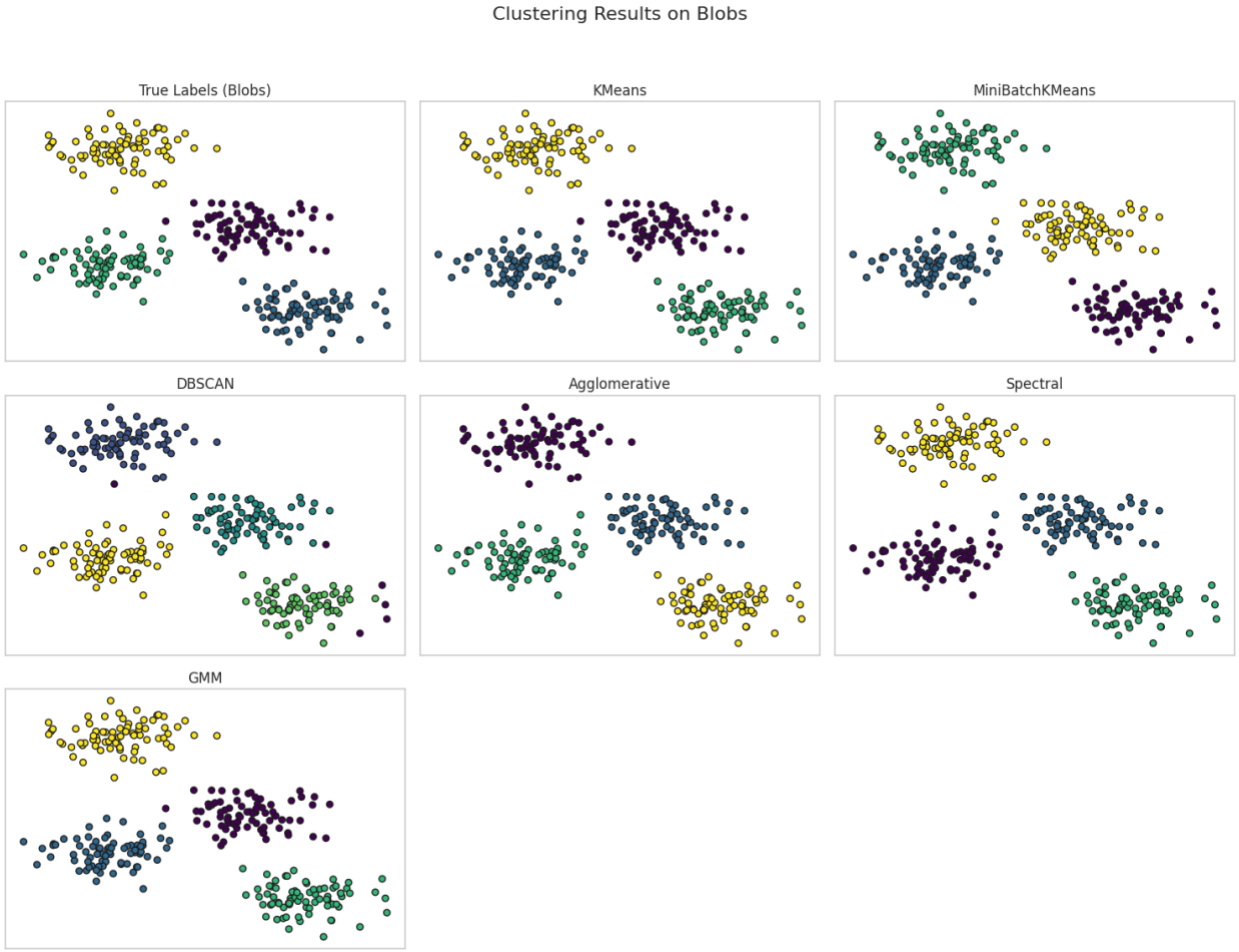
```
--- Clustering Evaluation Results ---

Dataset       Algorithm  Silhouette  Calinski-Harabasz  Davies-Bouldin      ARI      NMI
  Blobs          KMeans    0.656923         958.008789        0.461370 1.000000 1.000000
  Blobs MiniBatchKMeans    0.656923         958.008789        0.461370 1.000000 1.000000
  Blobs        Spectral    0.656923         958.008789        0.461370 1.000000 1.000000
  Blobs             GMM    0.656923         958.008789        0.461370 1.000000 1.000000
  Blobs   Agglomerative    0.656559         957.423275        0.459937 0.991081 0.987214
  Blobs          DBSCAN    0.626156         632.486096        1.619661 0.964288 0.950674
   Iris             GMM    0.231027         138.894565        1.321992 0.781868 0.819843
   Iris   Agglomerative    0.400636         201.251454        0.978821 0.587941 0.663414
   Iris MiniBatchKMeans    0.413613         206.074734        0.921677 0.571359 0.618694
   Iris        Spectral    0.388799         197.990077        0.836786 0.539216 0.650875
   Iris          KMeans    0.386941         207.265914        0.869814 0.472818 0.597298
   Iris          DBSCAN   -0.194195          16.881096        2.141522 0.087595 0.278307
  Moons          DBSCAN    0.382642         254.823961        1.031711 1.000000 1.000000
  Moons        Spectral    0.394261         269.787362        0.884242 0.498777 0.667080
  Moons   Agglomerative    0.420235         317.404307        0.906065 0.396999 0.506214
  Moons MiniBatchKMeans    0.446566         385.207016        0.910788 0.305111 0.359343
  Moons          KMeans    0.435171         383.619623        0.921077 0.283118 0.351712
  Moons             GMM    0.407430         347.239084        0.891659 0.201797 0.304600
```

c. **Analysis:**

- **Blobs:**
  - Data Characteristics:
    - **Well-separated, spherical**, equally sized clusters.
    - Ideal for **centroid-based and Gaussian mixture models**.
  - Expected Outcomes:
    - **KMeans, MiniBatchKMeans, GMM, and Spectral** should perform **perfectly**.
      - These algorithms assume or work best with **spherical, convex clusters**.
      - **GMM fits Gaussian distributions and blobs resemble isotropic Gaussians**.
      - **Spectral performs similarly because the similarity graph reflects clear separability**.
    - **Agglomerative** Clustering should do **very well**.
      - **Ward linkage (if used) minimizes variance**, which works well on blobs.
      - **Slight variation** in results due to **linkage decisions**.
    - **DBSCAN** should perform **well but not perfectly**.
      - It's designed for **density-based clusters**. Since blobs are **evenly dense with little noise**, DBSCAN might:
        - ✓ Misclassify border points.
        - ✓ Split or merge clusters if eps and min_samples are not ideal.
  - Why Results Match Expectations:
    - The **blobs dataset is designed to match centroid-based and Gaussian assumptions**.
    - Algorithms like **KMeans and GMM** are built exactly for this type of structure.
    - **DBSCAN struggles** slightly, as expected, because it **doesn't rely on centroids or Gaussian shapes.**
  - The **results met expectations**.
    - **Perfect or near-perfect scores from KMeans, MiniBatchKMeans, GMM, and Spectral** confirm their assumptions align with the dataset.
    - Slightly lower but still strong results for **Agglomerative are typical given linkage variation.**
    - **DBSCAN's slightly lower score reflects its sensitivity to parameter tuning**, as expected.

```
=== Dataset: Blobs ===
KMeans: Silhouette=0.657, CH=958.0, DB=0.461, ARI=1.000, NMI=1.000
MiniBatchKMeans: Silhouette=0.657, CH=958.0, DB=0.461, ARI=1.000, NMI=1.000
DBSCAN: Silhouette=0.626, CH=632.5, DB=1.620, ARI=0.964, NMI=0.951
Agglomerative: Silhouette=0.657, CH=957.4, DB=0.460, ARI=0.991, NMI=0.987
Spectral: Silhouette=0.657, CH=958.0, DB=0.461, ARI=1.000, NMI=1.000
GMM: Silhouette=0.657, CH=958.0, DB=0.461, ARI=1.000, NMI=1.000
```



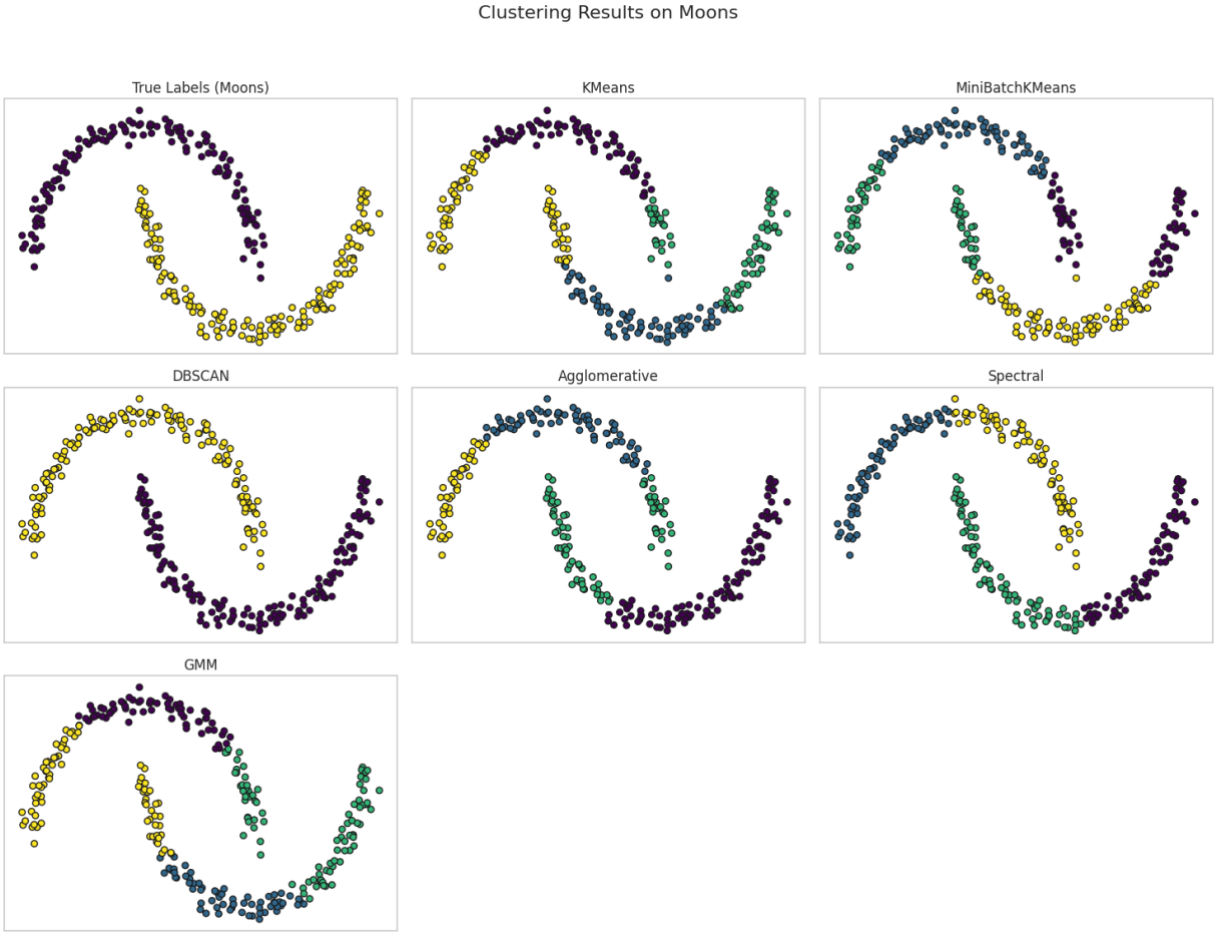
- **Moons:**
  - Data Characteristics:
    - **Non-convex, crescent-shaped clusters.**
    - Classic challenge for linear and centroid-based algorithms.
  - Expected Outcomes:
    - **DBSCAN** should perform **perfectly.**
      - Designed for **arbitrarily shaped** clusters.
      - Uses **local density, not distance to centroids,** which is **ideal for curved shapes.**
    - **Spectral** Clustering should perform **reasonably well**.
      - **Graph-based approach** can capture the non-linearity in the moon shape.
    - **Agglomerative** Clustering should do **moderately well**.
      - Depends heavily on **linkage method**.
      - **Single-linkage could follow shape better than others**.
    - **KMeans, MiniBatchKMeans, and GMM** should perform **poorly**.

- Assume **convex** clusters.
- These methods will likely slice across the curved moons, grouping parts of different arcs.

- Why Results Match Expectations:
  - **Spectral's moderate success also aligns with theory.**
  - KMeans/GMM fail due to **fundamental mismatch** with the data geometry.

- The **results met expectations**.
  - **DBSCAN achieves perfect clustering, exactly as theory predicts.**
  - Spectral and Agglomerative show moderate success, matching their theoretical ability to handle some non-linearity.
  - **Poor performance from KMeans, MiniBatchKMeans, and GMM aligns perfectly with their assumption violations.**

```
=== Dataset: Moons ===
KMeans: Silhouette=0.435, CH=383.6, DB=0.921, ARI=0.283, NMI=0.352
MiniBatchKMeans: Silhouette=0.447, CH=385.2, DB=0.911, ARI=0.305, NMI=0.359
DBSCAN: Silhouette=0.383, CH=254.8, DB=1.032, ARI=1.000, NMI=1.000
Agglomerative: Silhouette=0.420, CH=317.4, DB=0.906, ARI=0.397, NMI=0.506
Spectral: Silhouette=0.394, CH=269.8, DB=0.884, ARI=0.499, NMI=0.667
GMM: Silhouette=0.407, CH=347.2, DB=0.892, ARI=0.202, NMI=0.305
```



- **Iris:**
  - Data Characteristics:
    - **Real-world data with overlapping** clusters, unequal sizes, and shapes.
    - **Only Setosa is clearly separated**; Versicolor and Virginica overlap.

- Expected Outcomes:
  - **GMM** should perform **best**.
    - Can handle **overlapping**, elliptical clusters using full covariance matrices.
    - **Soft clustering is useful when species boundaries are fuzzy**.
  - **Agglomerative** should perform **moderately well**.
    - More flexible than KMeans with shape and size.
    - Good for imbalanced clusters.
  - **KMeans and MiniBatchKMeans** should perform **moderately to poorly**.
    - Assume **spherical, equally sized clusters** ,assumptions is not met.
  - **Spectral Clustering** should perform **moderately**.
    - Can capture some **non-linear structure** but lacks a strong signal in this dataset.
  - **DBSCAN** should perform **poorly**.
    - There is **no strong density separation** between clusters.
    - Tends to merge clusters or label many points as noise.

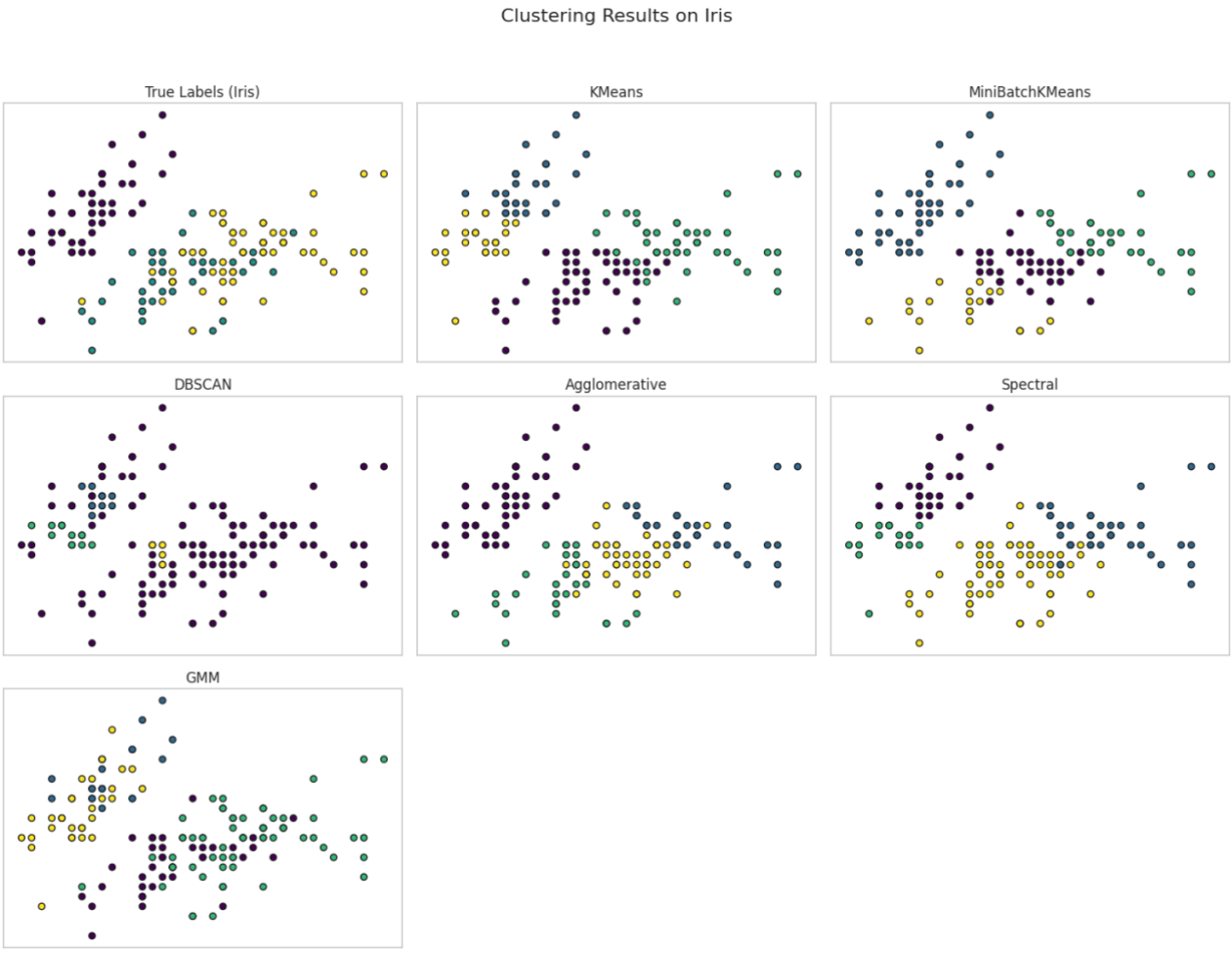
- Why Results Match Expectations:

  - **Algorithms that can model shape and overlap (like GMM) perform best ,exactly as theory suggests.**
  - **KMeans and DBSCAN perform poorly due to incorrect assumptions about cluster shape and density.**

- The **results met expectations**.

  - **GMM outperforms others**, showing flexibility for elliptical and overlapping clusters.
  - **Moderate performance from Agglomerative and spectral methods** fits their flexible assumptions.
  - **Poor performance of KMeans and DBSCAN** aligns with their rigid assumptions not being met in the dataset.

```
=== Dataset: Iris ===
KMeans: Silhouette=0.387, CH=207.3, DB=0.870, ARI=0.473, NMI=0.597
MiniBatchKMeans: Silhouette=0.414, CH=206.1, DB=0.922, ARI=0.571, NMI=0.619
DBSCAN: Silhouette=-0.194, CH=16.9, DB=2.142, ARI=0.088, NMI=0.278
Agglomerative: Silhouette=0.401, CH=201.3, DB=0.979, ARI=0.588, NMI=0.663
Spectral: Silhouette=0.389, CH=198.0, DB=0.837, ARI=0.539, NMI=0.651
GMM: Silhouette=0.231, CH=138.9, DB=1.322, ARI=0.782, NMI=0.820
```



Task 2: Experiment with existing clustering algorithms and my own implemetations

a. Clustering algorithm and parameters:

- KMeans(n_clusters=4, random_state=0)
- AgglomerativeClustering(n_clusters=4, linkage='ward')
- custom_kmeans(n_clusters=4)
- custom_agglomerative(n_clusters=4)

b. **Results:**

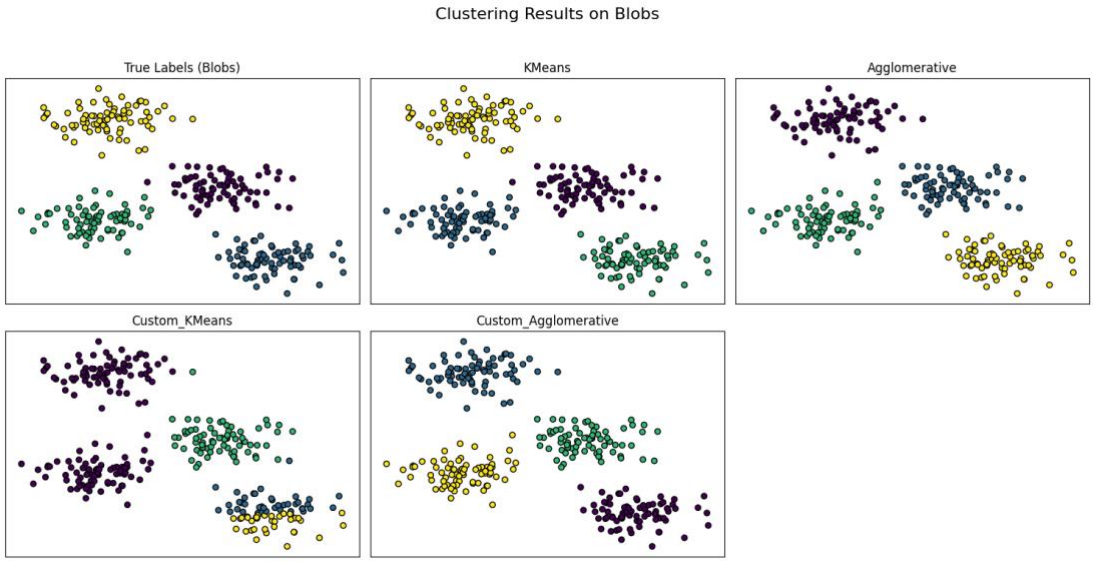
```
--- Clustering Evaluation Results ---
```

Dataset	Algorithm	Silhouette	Calinski-Harabasz	Davies-Bouldin	ARI	NMI
Blobs	KMeans	0.656923	958.008789	0.461370	1.000000	1.000000
Blobs	Agglomerative	0.656559	957.423275	0.459937	0.991081	0.987214
Blobs	Custom_Agglomerative	0.656559	957.423275	0.459937	0.991081	0.987214
Blobs	Custom_KMeans	0.340741	215.068941	1.358407	0.604882	0.760239
Iris	Agglomerative	0.400636	201.251454	0.978821	0.587941	0.663414
Iris	Custom_KMeans	0.413613	206.074734	0.921677	0.571359	0.618694
Iris	Custom_Agglomerative	0.406746	104.291513	0.502445	0.552203	0.700554
Iris	KMeans	0.386941	207.265914	0.869814	0.472818	0.597298
Moons	Custom_Agglomerative	-0.089924	88.382411	6.673883	0.986755	0.971937
Moons	Agglomerative	0.420235	317.404307	0.906065	0.396999	0.506214
Moons	KMeans	0.435171	383.619623	0.921077	0.283118	0.351712
Moons	Custom_KMeans	0.435171	383.619623	0.921077	0.283118	0.351712

c. **Analysis:**

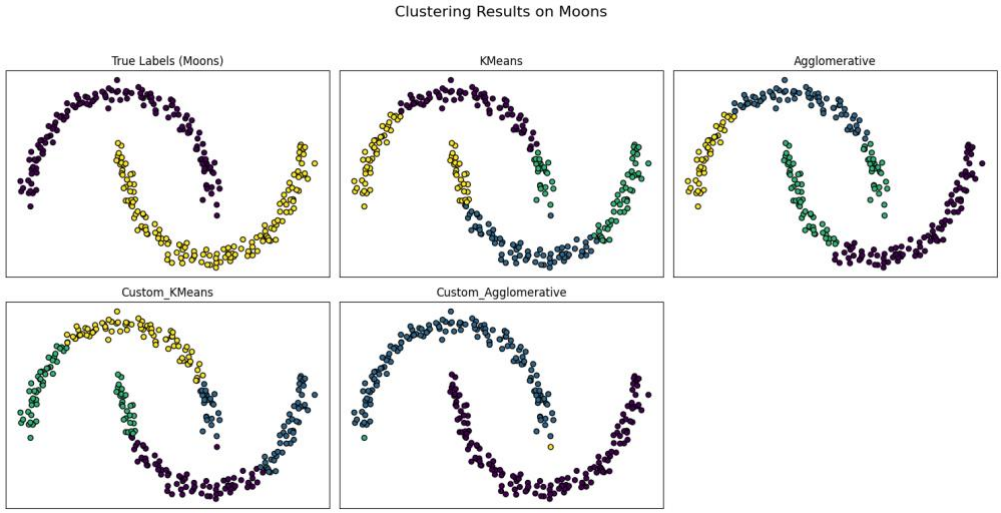
- **Blobs:**
  - KMeans and Agglomerative perform perfectly or near-perfectly, which aligns with expectations: **make_blobs generates spherical, well-separated clusters, ideal for KMeans and Ward-linkage Agglomerative clustering**.
  - **Custom Agglomerative matches the library version** , as expected, since both likely use similar strategies (s**ingle or average linkage approximating Ward's method**).
  - Custom KMeans underperforms, which is notable:
    - Likely due to **poor centroid initialization** (random without k-means), causing it to converge to suboptimal local minima.
    - The much higher Davies-Bouldin index (1.36 vs 0.46) indicates **less compact and well-separated clusters**.
    - ARI of 0.60 shows **many points were misclassified compared to ground truth.**
  - The results match expectations. **Well-separated clusters favor centroid-based algorithms**.

```
=== Dataset: Blobs ===
KMeans: Silhouette=0.657, CH=958.0, DB=0.461, ARI=1.000, NMI=1.000
Agglomerative: Silhouette=0.657, CH=957.4, DB=0.460, ARI=0.991, NMI=0.987
Custom_KMeans: Silhouette=0.341, CH=215.1, DB=1.358, ARI=0.605, NMI=0.760
Custom_Agglomerative: Silhouette=0.657, CH=957.4, DB=0.460, ARI=0.991, NMI=0.987
```



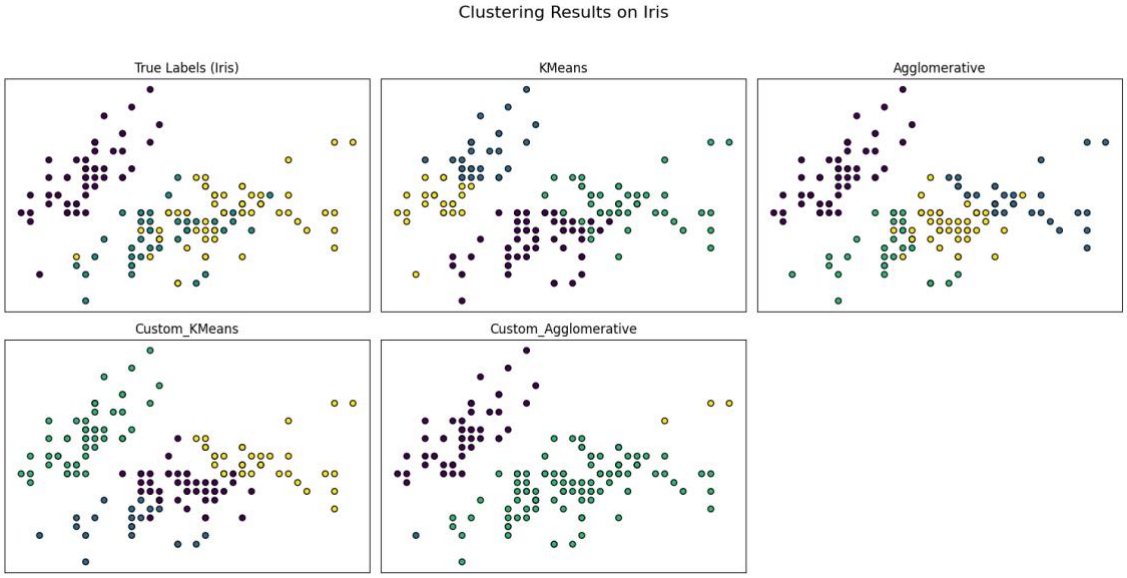
- **Moons:**
  - Moons dataset is non-linearly separable and curved. **KMeans fails, which is expected, it assumes convex, spherical clusters.**
  - **Custom Agglomerative performs best in ARI and NMI**, despite poor Silhouette (-0.09) and DB (6.67).
    - This suggests that, **although the clusters aren't compact** (Silhouette ↓), **they still align well with the labels** (ARI/NMI ↑).
    - **Custom Agglomerative uses single linkage**, which is good for **chaining and capturing complex shapes** . Single-linkage Agglomerative clustering (custom one) works very well on non-convex data.
  - **Scikit-learn Agglomerative (Ward) doesn't perform well** — Ward linkage **assumes Euclidean space-based compactness, not suited for curved shapes**.
  - KMeans and Custom KMeans perform identically , as expected, since their output is the same, but poorly on external metrics.

```
=== Dataset: Moons ===
KMeans: Silhouette=0.435, CH=383.6, DB=0.921, ARI=0.283, NMI=0.352
Agglomerative: Silhouette=0.420, CH=317.4, DB=0.906, ARI=0.397, NMI=0.506
Custom_KMeans: Silhouette=0.435, CH=383.6, DB=0.921, ARI=0.283, NMI=0.352
Custom_Agglomerative: Silhouette=-0.090, CH=88.4, DB=6.674, ARI=0.987, NMI=0.972
```



- **Iris (n_ckusters=4):**
  - All algorithms perform moderately.
  - Iris contains 3 natural clusters, but I've set n_clusters=4, which introduces noise into metrics.(Iris actually contains 3 species; setting n_clusters=4 artificially splits them.)
  - **Custom KMeans slightly outperforms sklearn KMeans in Silhouette, but might be coincidence depending on initialization.**
  - **Custom Agglomerative gets lower CH (Calinski-Harabasz) score but better NMI**, meaning it **matched the structure more evenly across clusters**, even if clusters are not as tight.
  - Results are aligned with expectations. **Algorithms struggle mildly due to the mismatch between true class count (3) and clustering target (4).** Agglomerative slightly better at respecting the natural groupings.

```
=== Dataset: Iris ===
KMeans: Silhouette=0.387, CH=207.3, DB=0.870, ARI=0.473, NMI=0.597
Agglomerative: Silhouette=0.401, CH=201.3, DB=0.979, ARI=0.588, NMI=0.663
Custom_KMeans: Silhouette=0.414, CH=206.1, DB=0.922, ARI=0.571, NMI=0.619
Custom_Agglomerative: Silhouette=0.407, CH=104.3, DB=0.502, ARI=0.552, NMI=0.701
```



- **Iris (n_ckusters=3):**
  - All algorithms perform reasonably well,
  - The Iris dataset contains 3 natural clusters corresponding to the three iris species (setosa, versicolor, virginica). **Setting n_clusters=3 aligns correctly with the ground truth, allowing more meaningful external evaluations like ARI and NMI.**
  - **KMeans (both custom and sklearn) is effective in assigning labels that mostly match ground truth.**
  - **Custom KMeans performs best in ARI (0.645)**, indicating it assigned labels closest to the true classes. Its Silhouette score (0.457) is nearly identical to sklearn's KMeans, suggesting similar intra-cluster compactness.
  - **Sklearn KMeans shows slightly higher CH score (241.9) and Silhouette (0.460)**, which may reflect better separation between clusters, **possibly due to k-means initialization**. However, it has slightly lower ARI and NMI, indicating less accurate label matching.
  - **Custom Agglomerative better captures the informational structure of the data** (highest NMI), despite lower CH.
  - **Custom Agglomerative shows the highest Silhouette (0.505) and lowest Davies-Bouldin index (0.493), meaning clusters are well-separated and internally coherent.** However, its CH score is much lower (131.5), which suggests **less between-cluster dispersion** ,often expected with non-Ward linkage, which doesn't prioritize global structure.It also has the highest NMI (0.720), indicating that **even if the clusters aren't tight (low CH), their structure aligns well with the actual classes.**
  - **Sklearn Agglomerative** gives balanced results , without leading in any metric. **Likely uses Ward linkage, which tends to create compact clusters ,good for CH but less flexible in complex overlaps.**

```
=== Dataset: Iris ===
KMeans: Silhouette=0.460, CH=241.9, DB=0.834, ARI=0.620, NMI=0.659
Agglomerative: Silhouette=0.447, CH=222.7, DB=0.803, ARI=0.615, NMI=0.675
Custom_KMeans: Silhouette=0.457, CH=239.5, DB=0.828, ARI=0.645, NMI=0.661
Custom_Agglomerative: Silhouette=0.505, CH=131.5, DB=0.493, ARI=0.558, NMI=0.720
```

Appendix (Code)

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs, make_moons, load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans, MiniBatchKMeans, DBSCAN,
AgglomerativeClustering, SpectralClustering
from sklearn.mixture import GaussianMixture
from sklearn.metrics import (
    silhouette_score, calinski_harabasz_score, davies_bouldin_score,
    adjusted_rand_score, normalized_mutual_info_score
)

# 1. Define datasets
datasets = {
    "Blobs": make_blobs(n_samples=300, centers=4, cluster_std=0.60,
random_state=0),
    "Moons": make_moons(n_samples=300, noise=0.05, random_state=0),
    "Iris": (load_iris().data, load_iris().target)
}

# 2. Define clustering algorithms
clustering_algorithms = {
    "KMeans": lambda X: KMeans(n_clusters=4, random_state=0).fit_predict(X),
    "MiniBatchKMeans": lambda X: MiniBatchKMeans(n_clusters=4,
random_state=0).fit_predict(X),
    "DBSCAN": lambda X: DBSCAN(eps=0.3, min_samples=5).fit_predict(X),
    "Agglomerative": lambda X: AgglomerativeClustering(n_clusters=4,
linkage='ward').fit_predict(X),
    "Spectral": lambda X: SpectralClustering(n_clusters=4,
affinity='nearest_neighbors', n_neighbors=10, random_state=0).fit_predict(X),
    "GMM": lambda X: GaussianMixture(n_components=4, random_state=0).fit_predict(X)
}

results = []

# 3. Run clustering, evaluation, and visualization
for ds_name, (X, y_true) in datasets.items():
    print(f"\n=== Dataset: {ds_name} ===")

    # Scale features for better clustering performance
```

```
    X = StandardScaler().fit_transform(X)

    n_algorithms = len(clustering_algorithms)
    n_plots = n_algorithms + 1  # +1 for true labels
    cols = 3
    rows = (n_plots + cols - 1) // cols  # ceiling division

    fig, axs = plt.subplots(rows, cols, figsize=(cols * 5, rows * 4))
    axs = axs.flatten()

    # Plot true labels
    axs[0].scatter(X[:, 0], X[:, 1], c=y_true, cmap='viridis', s=30, edgecolor='k')
    axs[0].set_title(f"True Labels ({ds_name})")
    axs[0].set_xticks([])
    axs[0].set_yticks([])

    for i, (name, algorithm) in enumerate(clustering_algorithms.items(), start=1):
        y_pred = algorithm(X)

        # Plot clustering results
        axs[i].scatter(X[:, 0], X[:, 1], c=y_pred, cmap='viridis', s=30,
edgecolor='k')
        axs[i].set_title(name)
        axs[i].set_xticks([])
        axs[i].set_yticks([])

        # Compute internal evaluation metrics
        try:
            sil = silhouette_score(X, y_pred)
            ch = calinski_harabasz_score(X, y_pred)
            db = davies_bouldin_score(X, y_pred)
        except Exception:
            sil = ch = db = np.nan

        # Compute external evaluation metrics
        try:
            ari = adjusted_rand_score(y_true, y_pred)
            nmi = normalized_mutual_info_score(y_true, y_pred)
        except Exception:
            ari = nmi = np.nan
```

```
        print(f"{name}: Silhouette={sil:.3f}, CH={ch:.1f}, DB={db:.3f},
ARI={ari:.3f}, NMI={nmi:.3f}")

        # Save results for summary
        results.append({
            "Dataset": ds_name,
            "Algorithm": name,
            "Silhouette": sil,
            "Calinski-Harabasz": ch,
            "Davies-Bouldin": db,
            "ARI": ari,
            "NMI": nmi
        })

    # Hide any extra subplots
    for j in range(i + 1, len(axs)):
        fig.delaxes(axs[j])

    plt.suptitle(f"Clustering Results on {ds_name}", fontsize=16)
    plt.tight_layout(rect=[0, 0.03, 1, 0.95])
    plt.show()

# 4. Display Results Summary
results_df = pd.DataFrame(results)
results_df_sorted = results_df.sort_values(by=["Dataset", "ARI"], ascending=[True,
False])

print("\n--- Clustering Evaluation Results ---\n")
print(results_df_sorted.to_string(index=False))
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs, make_moons, load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans, MiniBatchKMeans, DBSCAN,
AgglomerativeClustering, SpectralClustering
from sklearn.mixture import GaussianMixture
from sklearn.metrics import (
    silhouette_score, calinski_harabasz_score, davies_bouldin_score,
    adjusted_rand_score, normalized_mutual_info_score
```

```
)
from scipy.spatial.distance import pdist, squareform

# ==========================
# Custom KMeans Implementation
# ==========================
def custom_kmeans(X, n_clusters=4, max_iter=100, tol=1e-4):
    np.random.seed(0)
    idx = np.random.choice(len(X), n_clusters, replace=False)
    centroids = X[idx]

    for _ in range(max_iter):
        distances = np.linalg.norm(X[:, np.newaxis] - centroids, axis=2)
        labels = np.argmin(distances, axis=1)

        new_centroids = np.array([
            X[labels == i].mean(axis=0) if np.any(labels == i) else centroids[i]
            for i in range(n_clusters)
        ])

        if np.allclose(centroids, new_centroids, atol=tol):
            break
        centroids = new_centroids

    return labels

# ==========================
# Custom Agglomerative Clustering (Single Linkage)
# ==========================
def custom_agglomerative(X, n_clusters=4):
    N = len(X)
    clusters = [{i} for i in range(N)]
    distances = squareform(pdist(X))
    np.fill_diagonal(distances, np.inf)

    while len(clusters) > n_clusters:
        i, j = np.unravel_index(np.argmin(distances), distances.shape)
        new_cluster = clusters[i].union(clusters[j])
        clusters[i] = new_cluster
        del clusters[j]

        distances = np.delete(distances, j, axis=0)
```

```
        distances = np.delete(distances, j, axis=1)

        for k in range(len(clusters)):
            if k != i:
                dists = [np.linalg.norm(X[p1] - X[p2]) for p1 in clusters[i] for p2
in clusters[k]]
                distances[i, k] = distances[k, i] = np.min(dists)

        distances[i, i] = np.inf

    labels = np.zeros(N, dtype=int)
    for label, cluster in enumerate(clusters):
        for index in cluster:
            labels[index] = label
    return labels

# ==========================
# Define Datasets
# ==========================
datasets = {
    "Blobs": make_blobs(n_samples=300, centers=4, cluster_std=0.60,
random_state=0),
    "Moons": make_moons(n_samples=300, noise=0.05, random_state=0),
    "Iris": (load_iris().data, load_iris().target)
}

# ==========================
# Clustering Algorithms
# ==========================
clustering_algorithms = {
    "KMeans": lambda X: KMeans(n_clusters=4, random_state=0).fit_predict(X),
    "Agglomerative": lambda X: AgglomerativeClustering(n_clusters=4,
linkage='ward').fit_predict(X),
    "Custom_KMeans": lambda X: custom_kmeans(X, n_clusters=4),
    "Custom_Agglomerative": lambda X: custom_agglomerative(X, n_clusters=4)
}

# ==========================
# Run Clustering and Evaluation
# ==========================
results = []
```

```
for ds_name, (X, y_true) in datasets.items():
    print(f"\n=== Dataset: {ds_name} ===")

    X = StandardScaler().fit_transform(X)
    n_algorithms = len(clustering_algorithms)
    n_plots = n_algorithms + 1  # +1 for true labels
    cols = 3
    rows = (n_plots + cols - 1) // cols

    fig, axs = plt.subplots(rows, cols, figsize=(cols * 5, rows * 4))
    axs = axs.flatten()

    axs[0].scatter(X[:, 0], X[:, 1], c=y_true, cmap='viridis', s=30, edgecolor='k')
    axs[0].set_title(f"True Labels ({ds_name})")
    axs[0].set_xticks([])
    axs[0].set_yticks([])

    for i, (name, algorithm) in enumerate(clustering_algorithms.items(), start=1):
        try:
            y_pred = algorithm(X)
        except Exception as e:
            print(f"{name}: failed with error: {e}")
            y_pred = np.full(X.shape[0], -1)  # Dummy label to avoid crash

        axs[i].scatter(X[:, 0], X[:, 1], c=y_pred, cmap='viridis', s=30,
edgecolor='k')
        axs[i].set_title(name)
        axs[i].set_xticks([])
        axs[i].set_yticks([])

        try:
            sil = silhouette_score(X, y_pred)
            ch = calinski_harabasz_score(X, y_pred)
            db = davies_bouldin_score(X, y_pred)
        except:
            sil = ch = db = np.nan

        try:
            ari = adjusted_rand_score(y_true, y_pred)
            nmi = normalized_mutual_info_score(y_true, y_pred)
        except:
            ari = nmi = np.nan
```

```
        print(f"{name}: Silhouette={sil:.3f}, CH={ch:.1f}, DB={db:.3f},
ARI={ari:.3f}, NMI={nmi:.3f}")

        results.append({
            "Dataset": ds_name,
            "Algorithm": name,
            "Silhouette": sil,
            "Calinski-Harabasz": ch,
            "Davies-Bouldin": db,
            "ARI": ari,
            "NMI": nmi
        })

    for j in range(i + 1, len(axs)):
        fig.delaxes(axs[j])

    plt.suptitle(f"Clustering Results on {ds_name}", fontsize=16)
    plt.tight_layout(rect=[0, 0.03, 1, 0.95])
    plt.show()

# ============================
# Display Results Summary
# ============================
results_df = pd.DataFrame(results)
results_df_sorted = results_df.sort_values(by=["Dataset", "ARI"], ascending=[True,
False])

print("\n--- Clustering Evaluation Results ---\n")
print(results_df_sorted.to_string(index=False))
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs, make_moons, load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans, MiniBatchKMeans, DBSCAN,
AgglomerativeClustering, SpectralClustering
from sklearn.mixture import GaussianMixture
from sklearn.metrics import (
    silhouette_score, calinski_harabasz_score, davies_bouldin_score,
```

```
    adjusted_rand_score, normalized_mutual_info_score
)
from scipy.spatial.distance import pdist, squareform

# ==========================
# Custom KMeans Implementation
# ==========================
def custom_kmeans(X, n_clusters=4, max_iter=100, tol=1e-4):
    np.random.seed(0)
    idx = np.random.choice(len(X), n_clusters, replace=False)
    centroids = X[idx]

    for _ in range(max_iter):
        distances = np.linalg.norm(X[:, np.newaxis] - centroids, axis=2)
        labels = np.argmin(distances, axis=1)

        new_centroids = np.array([
            X[labels == i].mean(axis=0) if np.any(labels == i) else centroids[i]
            for i in range(n_clusters)
        ])

        if np.allclose(centroids, new_centroids, atol=tol):
            break
        centroids = new_centroids

    return labels

# ==========================
# Custom Agglomerative Clustering (Single Linkage)
# ==========================
def custom_agglomerative(X, n_clusters=3):
    N = len(X)
    clusters = [{i} for i in range(N)]
    distances = squareform(pdist(X))
    np.fill_diagonal(distances, np.inf)

    while len(clusters) > n_clusters:
        i, j = np.unravel_index(np.argmin(distances), distances.shape)
        new_cluster = clusters[i].union(clusters[j])
        clusters[i] = new_cluster
        del clusters[j]
```

```
        distances = np.delete(distances, j, axis=0)
        distances = np.delete(distances, j, axis=1)

        for k in range(len(clusters)):
            if k != i:
                dists = [np.linalg.norm(X[p1] - X[p2]) for p1 in clusters[i] for p2
in clusters[k]]
                distances[i, k] = distances[k, i] = np.min(dists)

        distances[i, i] = np.inf

    labels = np.zeros(N, dtype=int)
    for label, cluster in enumerate(clusters):
        for index in cluster:
            labels[index] = label
    return labels

# ==========================
# Define Datasets
# ==========================
datasets = {

    "Iris": (load_iris().data, load_iris().target)
}

# ==========================
# Clustering Algorithms
# ==========================
clustering_algorithms = {
    "KMeans": lambda X: KMeans(n_clusters=3, random_state=0).fit_predict(X),
    "Agglomerative": lambda X: AgglomerativeClustering(n_clusters=3,
linkage='ward').fit_predict(X),
    "Custom_KMeans": lambda X: custom_kmeans(X, n_clusters=3),
    "Custom_Agglomerative": lambda X: custom_agglomerative(X, n_clusters=3)
}

# ==========================
# Run Clustering and Evaluation
# ==========================
results = []

for ds_name, (X, y_true) in datasets.items():
```

```
    print(f"\n=== Dataset: {ds_name} ===")

    X = StandardScaler().fit_transform(X)
    n_algorithms = len(clustering_algorithms)
    n_plots = n_algorithms + 1  # +1 for true labels
    cols = 3
    rows = (n_plots + cols - 1) // cols

    fig, axs = plt.subplots(rows, cols, figsize=(cols * 5, rows * 4))
    axs = axs.flatten()

    axs[0].scatter(X[:, 0], X[:, 1], c=y_true, cmap='viridis', s=30, edgecolor='k')
    axs[0].set_title(f"True Labels ({ds_name})")
    axs[0].set_xticks([])
    axs[0].set_yticks([])

    for i, (name, algorithm) in enumerate(clustering_algorithms.items(), start=1):
        try:
            y_pred = algorithm(X)
        except Exception as e:
            print(f"{name}: failed with error: {e}")
            y_pred = np.full(X.shape[0], -1)  # Dummy label to avoid crash

        axs[i].scatter(X[:, 0], X[:, 1], c=y_pred, cmap='viridis', s=30,
edgecolor='k')
        axs[i].set_title(name)
        axs[i].set_xticks([])
        axs[i].set_yticks([])

        try:
            sil = silhouette_score(X, y_pred)
            ch = calinski_harabasz_score(X, y_pred)
            db = davies_bouldin_score(X, y_pred)
        except:
            sil = ch = db = np.nan

        try:
            ari = adjusted_rand_score(y_true, y_pred)
            nmi = normalized_mutual_info_score(y_true, y_pred)
        except:
            ari = nmi = np.nan
```

```
        print(f"{name}: Silhouette={sil:.3f}, CH={ch:.1f}, DB={db:.3f},
ARI={ari:.3f}, NMI={nmi:.3f}")

        results.append({
            "Dataset": ds_name,
            "Algorithm": name,
            "Silhouette": sil,
            "Calinski-Harabasz": ch,
            "Davies-Bouldin": db,
            "ARI": ari,
            "NMI": nmi
        })

    for j in range(i + 1, len(axs)):
        fig.delaxes(axs[j])

    plt.suptitle(f"Clustering Results on {ds_name}", fontsize=16)
    plt.tight_layout(rect=[0, 0.03, 1, 0.95])
    plt.show()

# ============================
# Display Results Summary
# ============================
results_df = pd.DataFrame(results)
results_df_sorted = results_df.sort_values(by=["Dataset", "ARI"], ascending=[True,
False])

print("\n--- Clustering Evaluation Results ---\n")
print(results_df_sorted.to_string(index=False))
```