MODUL 9

Clustering & Outlier Detection

1. TUJUAN PRAKTIKUM

1) Menggunakan Local Outlier Factor (LOF) untuk mendeteksi anomali berdasarkan kedekatan lokal.
2) Menerapkan algoritma clustering berbasis partisi (K-Means) dan berbasis densitas (DBSCAN).
3) Menghitung dan menginterpretasikan internal evaluation metrics (Silhouette Score & Davies–Bouldin Index).

2. CLUSTERING

Clustering adalah metode unsupervised learning yang digunakan untuk mengelompokkan data berdasarkan kemiripannya tanpa menggunakan label. Dalam praktikum ini digunakan dua pendekatan:

1) **K-Means (Partition-Based)**

   K-Means merupakan algoritma partition-based yang bekerja dengan membagi data ke dalam sejumlah k cluster yang telah ditentukan sebelumnya. Prosesnya dilakukan dengan menempatkan centroid secara awal, lalu menghitung jarak Euclidean dari setiap titik data ke centroid terdekat, sehingga data akan dikelompokkan ke dalam cluster dengan jarak paling kecil. Algoritma ini sangat efektif untuk data yang memiliki bentuk cluster bulat dan kompak, namun memiliki kelemahan karena pengguna harus menentukan jumlah cluster (k) sejak awal, yang tidak selalu mudah ditentukan.

2) **DBSCAN (Density-Based)**

   DBSCAN adalah algoritma density-based yang membentuk cluster berdasarkan kepadatan titik data di sekitarnya. Algoritma ini mampu menghasilkan cluster dengan bentuk yang lebih beragam atau arbitrer, sehingga cocok digunakan pada data yang bersifat nonlinear. DBSCAN juga secara otomatis mengidentifikasi titik-titik yang memiliki kepadatan rendah sebagai noise atau outlier. Dengan kemampuannya mendeteksi cluster tidak beraturan tanpa perlu menentukan jumlah cluster di awal, DBSCAN menjadi alternatif yang kuat untuk data yang kompleks dan tidak terstruktur.

3. EVALUASI CLUSTERING

Evaluasi clustering dilakukan untuk menilai sejauh mana hasil pengelompokan yang terbentuk sudah mencerminkan struktur alami dalam data. Melalui metrik seperti Silhouette Score dan Davies–Bouldin Index, kualitas cluster dapat dianalisis secara objektif sehingga membantu menentukan apakah model telah menghasilkan pembagian kelompok yang optimal.

1) **Silhouette Score**

Silhouette Score adalah metrik yang digunakan untuk mengevaluasi kualitas cluster dengan melihat seberapa dekat sebuah titik dengan cluster miliknya dan seberapa jauh titik tersebut dari cluster lain. Dengan kata lain, skor ini menilai kekompakan dan pemisahan cluster dalam satu angka. Semakin tinggi Silhouette Score, semakin baik kualitas clustering karena cluster lebih rapat dan terpisah jelas.

Silhouette Score dihitung dari dua komponen:

- a(i): rata-rata jarak titik dengan anggota cluster yang sama (kekompakan cluster)
- b(i): jarak rata-rata titik ke cluster terdekat yang berbeda (pemisahan antar cluster)

Skor akhirnya:

- mendekati 1, artinya titik berada sangat dekat dengan cluster-nya dan jauh dari cluster lain (cluster sangat jelas)
- mendekati 0, artinya titik berada di area batas antar cluster (cluster tumpang tindih)
- mendekati –1, artinya titik salah klaster atau sangat dekat dengan cluster lain (cluster buruk)

2) **Davies–Bouldin Index (DBI)**

Davies–Bouldin Index mengukur kualitas cluster dengan melihat Dispersi cluster yaitu seberapa menyebar atau padat cluster tersebut dan Jarak antar cluster yaitu seberapa jauh cluster satu dengan lainnya. DBI dihitung sebagai rata-rata nilai "kemiripan" antar cluster. Setiap cluster "dibandingkan" dengan cluster lain: jika jarak antar cluster dekat dan dispersi melebar, maka nilainya tinggi dan dianggap buruk.

Interpretasi nilai, semakin kecil DBI maka semakin baik dan nilai DBI yang kecil berarti cluster saling jauh, serta cluster kompak atau tidak menyebar

4. LATIHAN PRAKTIKUM

Import Library dan Load Dataset

Dataset Iris Clustering

```
import pandas as pd
from pathlib import Path

df=pd.read_csv("iris_clusters.csv", delimiter=";")

df.head()
```

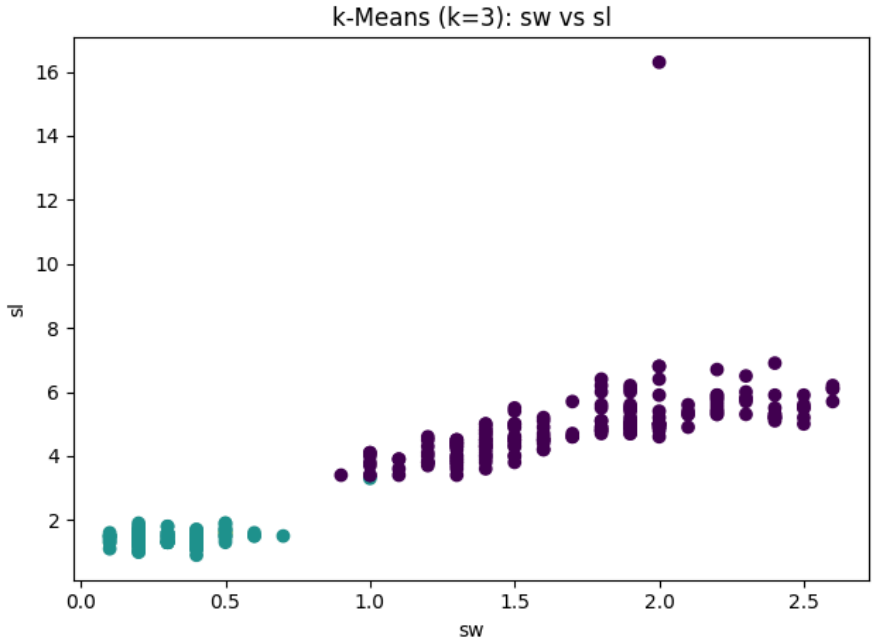
```
print("Shape (rows, cols):", df.shape)
print("Columns:", list(df.columns))
print("\nData types:")
print(df.dtypes)
print("\nData description:")
df.describe()
```

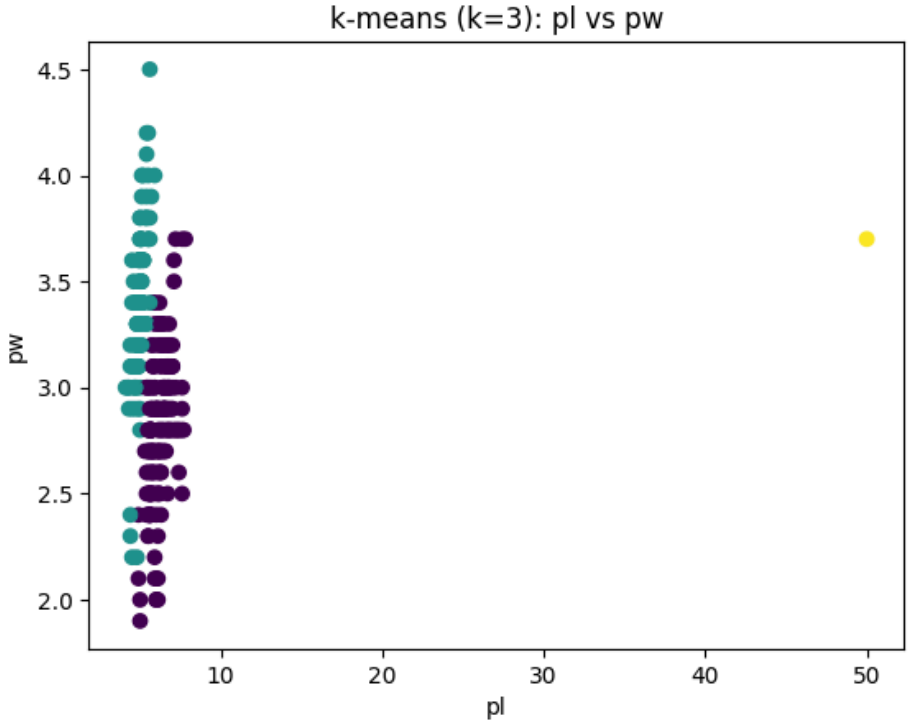
Cek Missing Value dan Duplikat

```
print("\nMissing values per column:")
print(df.isna().sum())

dup_count = df.duplicated().sum()
print("\nDuplicate rows:", dup_count)
```

Initial exploration

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

# Build the feature matrix
feature_cols = [c for c in df.columns if c.lower() not in {"id"}]
data = df[feature_cols].to_numpy()

# Run k-Means with k=3
kmeans = KMeans(n_clusters=3)
kmeans.fit(data)

# Inspect assignments and centroids
labels = kmeans.labels_
centroids = kmeans.cluster_centers_

print("Feature columns used:", feature_cols)
print("Cluster counts:", {i: (labels == i).sum() for i in range(3)})
print("\nCluster centroids (in feature space order above):")
print(centroids)

# Plot sw-sl
plt.figure()
plt.scatter(df["sw"], df["sl"], c=kmeans.labels_, cmap='viridis')
plt.xlabel("sw"); plt.ylabel("sl")
plt.title("k-Means (k=3): sw vs sl")
plt.tight_layout()
plt.show()

# Plot pw-pl
plt.scatter(df["pl"], df["pw"], c=kmeans.labels_, cmap="viridis")
plt.xlabel("pl"); plt.ylabel("pw")
plt.title("k-means (k=3): pl vs pw")
plt.show()
```



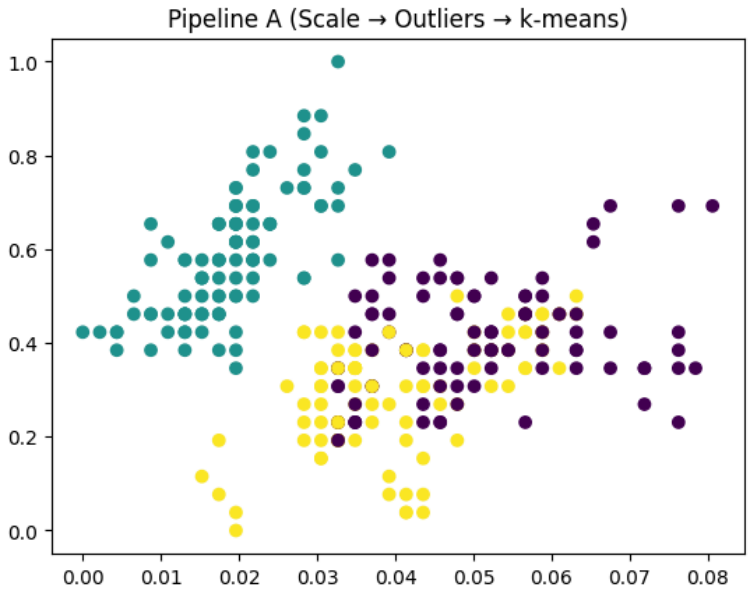


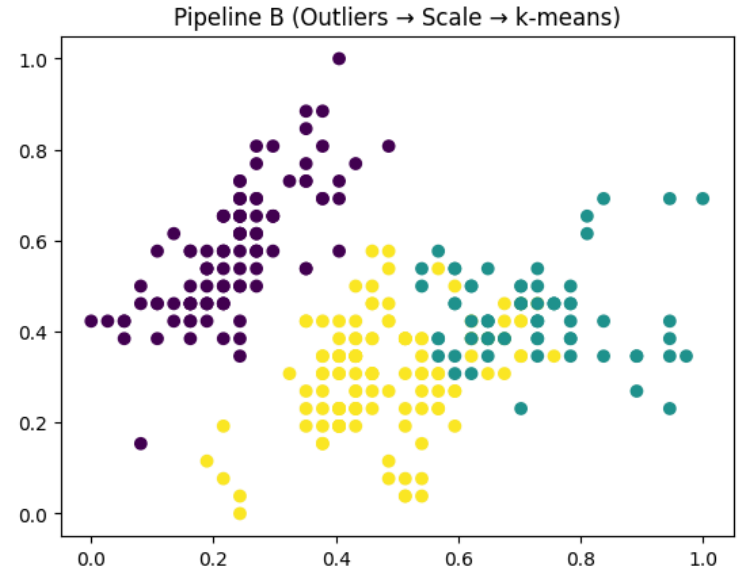
Pada percobaan awal menggunakan k-means (k=3), algoritma memang membentuk tiga cluster. Namun, salah satu cluster tersebut hanya berisi satu titik (sebuah outlier dengan nilai petal length yang sangat besar). Dua cluster lainnya berisi hampir semua bunga yang tersisa. Hal ini berarti bahwa tiga cluster tersebut tidak sesuai dengan tiga spesies Iris yang kita ketahui dari perkuliahan. Hasil ini menunjukkan bahwa outlier dan skala fitur sangat memengaruhi k-means, yang akan ditangani pada Task 2 dengan melakukan penanganan outlier dan menerapkan scaling.

Baseline k-means clustering

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.neighbors import LocalOutlierFactor
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

# features (drop id if present)
feature_cols = [c for c in df.columns if c.lower() != "id"]
X_raw = df[feature_cols].to_numpy()

# --- Pipeline A: Scale → Detect → Filter → k-means ---
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X_raw)

lof = LocalOutlierFactor(n_neighbors=10)
lof.fit(X_scaled)
scores = lof.negative_outlier_factor_
outliers_A = np.argsort(scores)[:5]   # 5 most outlier points
print("Pipeline A outlier indices:", outliers_A)

mask_A = np.ones(len(X_scaled), dtype=bool)
mask_A[outliers_A] = False
X_A = X_scaled[mask_A]

kmeans_A = KMeans(n_clusters=3, random_state=42)
labels_A = kmeans_A.fit_predict(X_A)
print("Pipeline A cluster counts:", np.bincount(labels_A))
print("Pipeline A centroids:\n", pd.DataFrame(kmeans_A.cluster_centers_, columns=feature_cols))

plt.scatter(X_A[:,0], X_A[:,1], c=labels_A, cmap="viridis")
plt.title("Pipeline A (Scale → Outliers → k-means)")
plt.show()
```

```
# --- Pipeline B: Detect → Filter → Scale → k-means ---
lof = LocalOutlierFactor(n_neighbors=10)
lof.fit(X_raw)
scores = lof.negative_outlier_factor_
outliers_B = np.argsort(scores)[:5]
print("Pipeline B outlier indices:", outliers_B)

mask_B = np.ones(len(X_raw), dtype=bool)
mask_B[outliers_B] = False
X_raw_clean = X_raw[mask_B]

scaler = MinMaxScaler()
X_B = scaler.fit_transform(X_raw_clean)

kmeans_B = KMeans(n_clusters=3, random_state=42)
labels_B = kmeans_B.fit_predict(X_B)
print("Pipeline B cluster counts:", np.bincount(labels_B))
print("Pipeline B centroids:\n", pd.DataFrame(kmeans_B.cluster_centers_, columns=feature_cols))

plt.scatter(X_B[:,0], X_B[:,1], c=labels_B, cmap="viridis")
plt.title("Pipeline B (Outliers → Scale → k-means)")
plt.show()
```

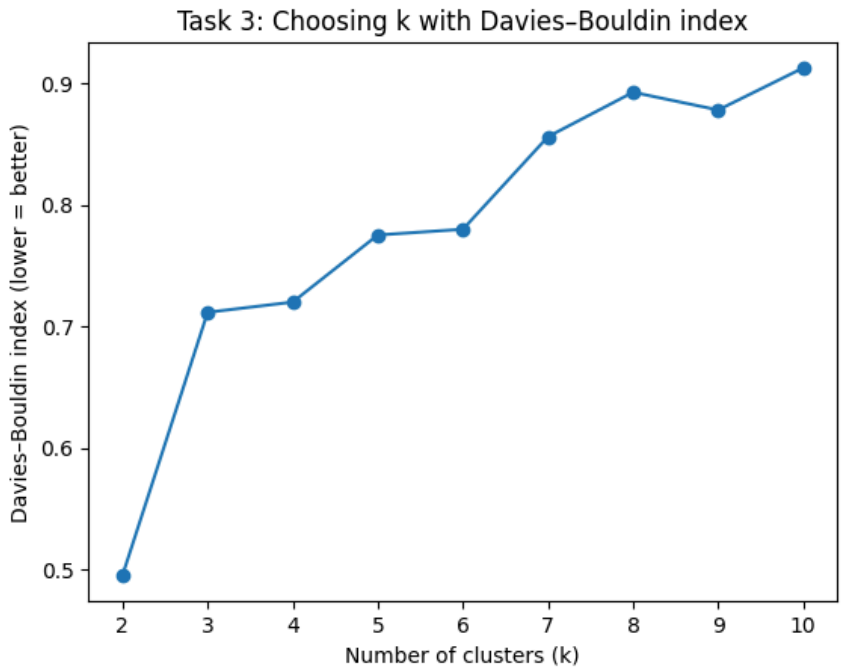


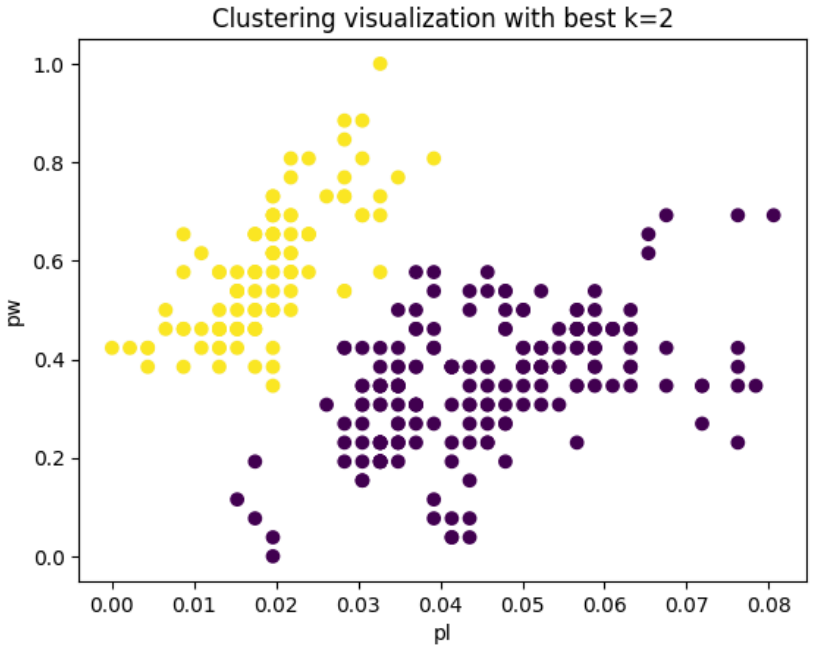


Di antara kedua pipeline tersebut, Pipeline A (melakukan scaling sebelum deteksi outlier) memberikan performa yang sedikit lebih baik. Cluster yang dihasilkan lebih seimbang ukurannya (98, 102, 95 dibandingkan 102, 69, 124), dan centroid-nya lebih terdistribusi secara merata. Pipeline B tetap berfungsi, tetapi menghasilkan satu cluster yang lebih kecil. Oleh karena itu, melakukan scaling sebelum deteksi outlier tampaknya memberikan hasil clustering yang lebih stabil untuk dataset ini.

Outlier detection (LOF) and scaling pipelines

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

# === TASK 3 ===
X_task3 = X_A   # scaled + outliers removed (from Task 2)

scores = []
k_values = range(2, 11)

for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42)
    labels = kmeans.fit_predict(X_task3)
    score = davies_bouldin_score(X_task3, labels)
    scores.append(score)
    print(f"k={k}, DB index={score:.4f}")

# best k
best_k = k_values[int(np.argmin(scores))]
print("\nBest k by Davies–Bouldin index:", best_k)

# plot DB curve
plt.plot(k_values, scores, marker="o")
plt.xlabel("Number of clusters (k)")
plt.ylabel("Davies–Bouldin index (lower = better)")
plt.title("Task 3: Choosing k with Davies–Bouldin index")
plt.show()

# visualize the clustering for best k
best_model = KMeans(n_clusters=best_k, random_state=42)
best_labels = best_model.fit_predict(X_task3)

plt.scatter(X_task3[:,0], X_task3[:,1], c=best_labels, cmap="viridis")
plt.title(f"Clustering visualization with best k={best_k}")
plt.xlabel(feature_cols[0]); plt.ylabel(feature_cols[1])
plt.show()
```

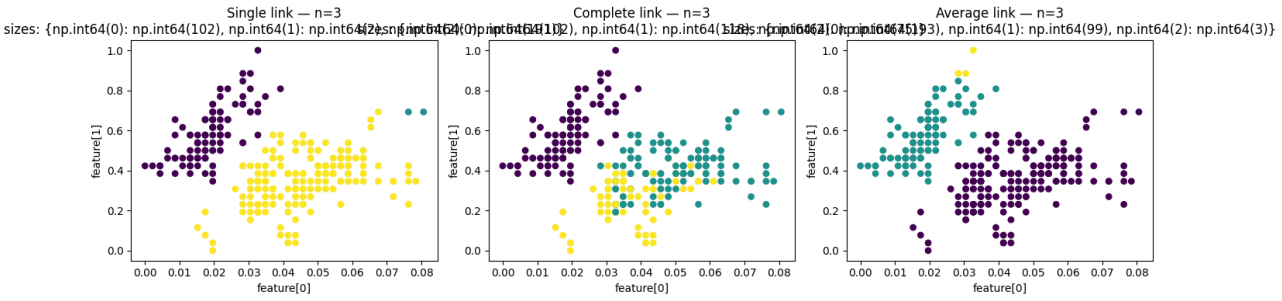




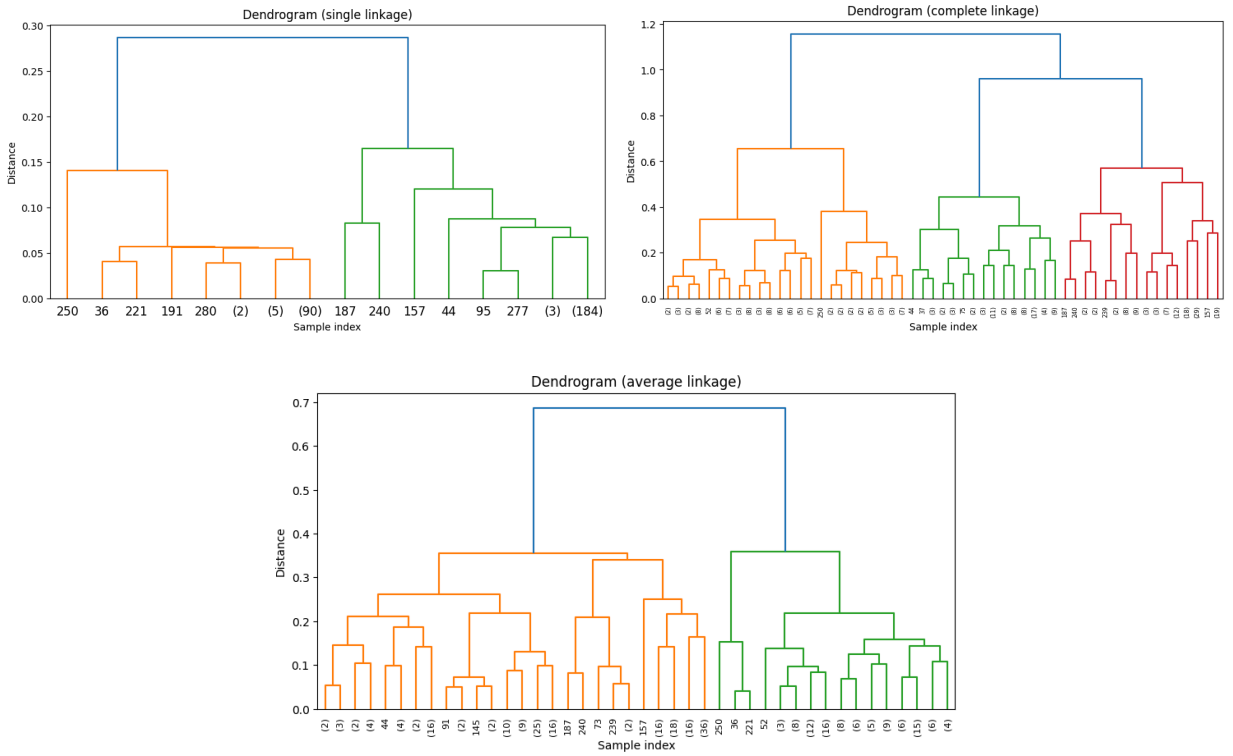
Skor Davies–Bouldin untuk k=2 hingga k=10 menunjukkan bahwa skor terendah berada pada k=2 (DB index = 0.4955). Ini berarti bahwa, menurut ukuran DB, dataset tersebut membentuk dua cluster yang kompak dan terpisah dengan baik. Visualisasi dengan k=2 juga mengonfirmasi bahwa bunga-bunga tersebut terbagi menjadi dua kelompok utama, bukan tiga. Hal ini berbeda dari asumsi awal kita tentang adanya tiga spesies, tetapi mencerminkan struktur yang ditemukan oleh k-means dan indeks DB dalam data.

Cluster validity (e.g. DB index / silhouette)

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

# Use cleaned + scaled data from Task 2
X_comp = X_A                    # already scaled [0,1], outliers removed
feat_x, feat_y = 0, 1           # which two scaled features to plot (0=first, 1=second)
methods = ["single", "complete", "average"]
n_clusters = 3

results = {}

plt.figure(figsize=(14, 4))
for i, m in enumerate(methods, 1):
    agg = AgglomerativeClustering(n_clusters=n_clusters, linkage=m)
    labels = agg.fit_predict(X_comp)
    unique, counts = np.unique(labels, return_counts=True)
    results[m] = dict(zip(unique, counts))

    # scatter for visual comparison
    ax = plt.subplot(1, 3, i)
    sc = ax.scatter(X_comp[:, feat_x], X_comp[:, feat_y], c=labels, s=28, cmap="viridis")
    ax.set_title(f"{m.capitalize()} link — n={n_clusters}\nsizes: {results[m]}")
    ax.set_xlabel(f"feature[{feat_x}]"); ax.set_ylabel(f"feature[{feat_y}]")

plt.tight_layout()
plt.show()

print("Cluster sizes (n_clusters=3):")
for m in methods:
    print(f"  {m:8s} -> {results[m]}")
```




```python
from scipy.cluster.hierarchy import linkage, dendrogram

def show_dendrogram(X, method="single", truncate_p=5):
    Z = linkage(X, method=method)
    plt.figure(figsize=(10, 5))
    dendrogram(Z, truncate_mode="level", p=truncate_p)
    plt.title(f"Dendrogram ({method} linkage)")
    plt.xlabel("Sample index"); plt.ylabel("Distance")
    plt.show()

# Example:
show_dendrogram(X_comp, method="single")
show_dendrogram(X_comp, method="complete")
show_dendrogram(X_comp, method="average")
```

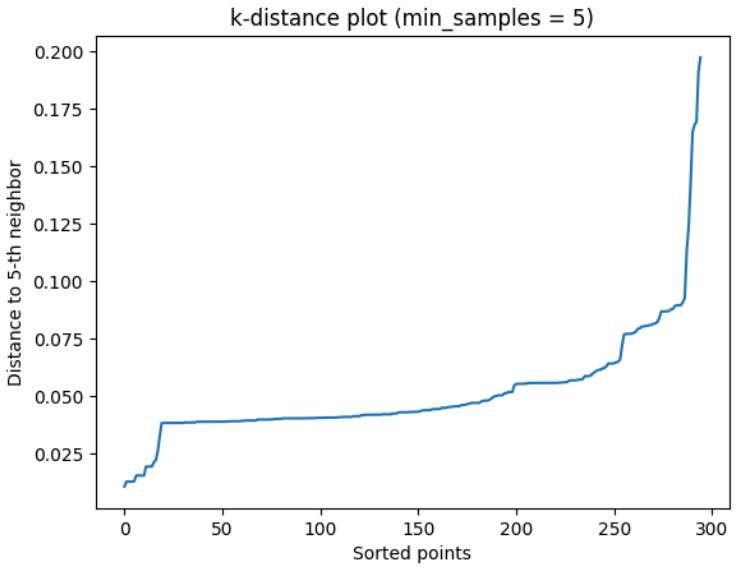


DBSCAN and density-based clustering

```
from sklearn.cluster import DBSCAN
import numpy as np

# use the cleaned+scaled data from Task 2 (Pipeline A)
data = X_A          # <- important: normalized matrix (min-max [0,1])

db = DBSCAN(eps=1.0, min_samples=5)
labels_db = db.fit_predict(data)

# counts
n_noise = np.sum(labels_db == -1)
n_clusters = len(set(labels_db)) - (1 if -1 in labels_db else 0)

print("DBSCAN with eps=1.0, min_samples=5")
print("Clusters (excluding noise):", n_clusters)
print("Noise points:", n_noise)
```

Dengan eps=1.0 dan min_samples=5, DBSCAN menempatkan semua observasi ke dalam satu cluster dan menemukan 0 titik noise. Hal ini terjadi karena dataset telah dinormalisasi ke rentang [0,1], sehingga nilai epsilon sebesar 1.0 menjadi sangat besar dan membuat semua titik dianggap sebagai tetangga. Akibatnya, seluruh dataset diperlakukan sebagai satu kelompok yang padat.

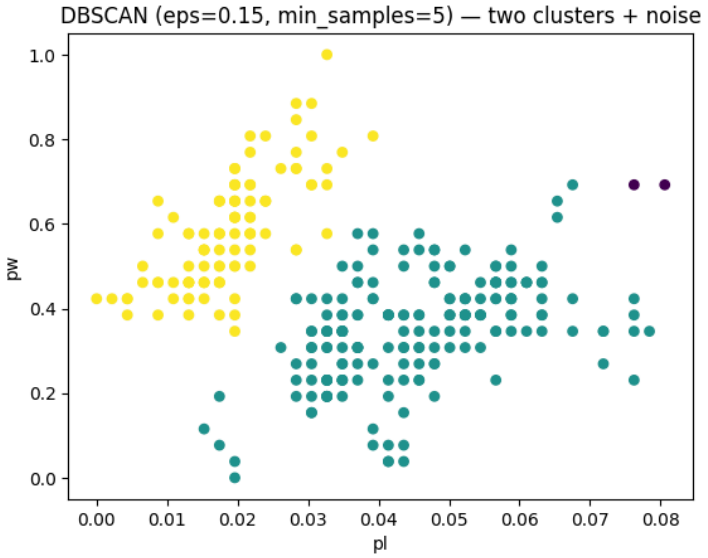
```python
# Plot k-distances
import matplotlib.pyplot as plt
import numpy as np
from sklearn.neighbors import NearestNeighbors

def k_distances2(X, n):
    neighbors = NearestNeighbors(n_neighbors=n)
    neighbors_fit = neighbors.fit(X)
    distances, indices = neighbors_fit.kneighbors(X)
    return distances, indices

# normalized dataset (from Task 2)
data = X_A
k = 5  # min_samples

distances, indices = k_distances2(data, k)
# sort the distance to the k-th neighbor for each point
kth = np.sort(distances[:, -1])   # use the k-th neighbor, not [:,1]
plt.plot(kth)
plt.xlabel("Sorted points")
plt.ylabel(f"Distance to {k}-th neighbor")
plt.title("k-distance plot (min_samples = 5)")
plt.show()
```




```python
# Try some eps values
candidates = [0.05, 0.08, 0.10, 0.12, 0.15]

for eps in candidates:
    model = DBSCAN(eps=eps, min_samples=5)
    labs = model.fit_predict(data)
    n_noise = np.sum(labs == -1)
    n_clusters = len(set(labs)) - (1 if -1 in labs else 0)
    print(f"eps={eps:.3f} -> clusters={n_clusters}, noise={n_noise}")
```

Rentang nilai ε sekitar 0.10–0.15 memberikan hasil yang paling seimbang, di mana DBSCAN mampu membentuk dua cluster utama dengan tambahan sejumlah kecil titik noise. Rentang ini menunjukkan parameter yang optimal untuk memisahkan struktur kepadatan dalam data tanpa menggabungkan seluruh observasi menjadi satu cluster atau menghasilkan terlalu banyak noise.

```python
# suppose eps_best is the one that gave clusters=2
eps_best = 0.15

best = DBSCAN(eps=eps_best, min_samples=5).fit_predict(data)

import matplotlib.pyplot as plt
plt.scatter(data[:,0], data[:,1], c=best, cmap="viridis", s=28)
plt.title(f"DBSCAN (eps={eps_best}, min_samples=5) — two clusters + noise")
plt.xlabel(feature_cols[0]); plt.ylabel(feature_cols[1])
plt.show()
```



Referensi