# MODUL 2

# Pengenalan Unsupervised Learning dan Implementasi Algoritma K-Means

## A. TUJUAN

1. Memahami dasar Unsupervised Learning
2. Memahami dan menerapkan algoritma K-means
3. Memahami dan menerapkan visualisasi data 2D menggunakan pyplot

## B. PETUNJUK

- Awali setiap aktivitas dengan do'a, semoga berkah dan mendapat kemudahan.
- Pahami tujuan, dasar teori, dan latihan-latihan pratikum dengan baik dan benar.
- Kerjakan tugas-tugas pratikum dengan baik, sabar dan jujur.
- Tanyakan kepada asisten/dosen apabila ada hal-hal yang kurang jelas

## C. DASAR TEORI

1. *Unsupervised Learning*

Berbeda dengan "Supervised Learning", Unsupervised Learning merupakan metode machine learning yang mempelajari informasi atau data tanpa menggunakan bantuan label. Contoh implementasi Unsupervised Learning yaitu ketika menemukan seekor kucing dan kelinci pertama kalinya, mesin tersebut akan mengkategorikannya tanpa sebutan nama aslinya melalui kesamaan dan perbedaan struktur tubuhnya diantara kedua hewan tersebut. Metode mesin ini terdapat teknik yang dinamakan Clustering.

Cara bekerjanya teknik clustering yaitu dengan mencari kesamaan (nilai) antar data di sebuah datasets, nilai data yang dekat dengan yang lain akan dibentuk satu kelompok. Teknik ini memiliki 3 cara implementasi yaitu: K-Means Clustering, Hierarchical Clustering dan Probabilistic Clustering.

Cek link berikut untuk detil
https://www.youtube.com/watch?v=IUn8k5zSI6g

2. *Data Visualization*

Sebelum lanjut membahas Teknik Clustering, Data Visualization (Visualisasi Data) sangat penting dalam di setiap membangun machine learning. Teknik tersebut berperan sebagai mempermudah memahami datasets dalam bentuk visual context sehingga pola, trends dan korelasi yang awalnya tidak diketahui ditemukan.

Python telah menyediakan "graphing libraries" dengan penuh banyak fitur yang berbeda. Library yang sering digunakan untuk teknik visualisasi data yaitu Matplotlib (yang akan digunakan dalam praktikum ini), Pandas Visualization, Seaborn, ggplot, dan Plotly. Berikut ini langkah-langkah dalam menerapkan Matplotlib:

- Create/Load Datasets

!Cobalah!

```
#get datasets
from sklearn.datasets import make_blobs

# create dataset
X, y = make_blobs(
    n_samples=300, n_features=2,
    centers=4, cluster_std=0.60,
    random_state=0
    )
```

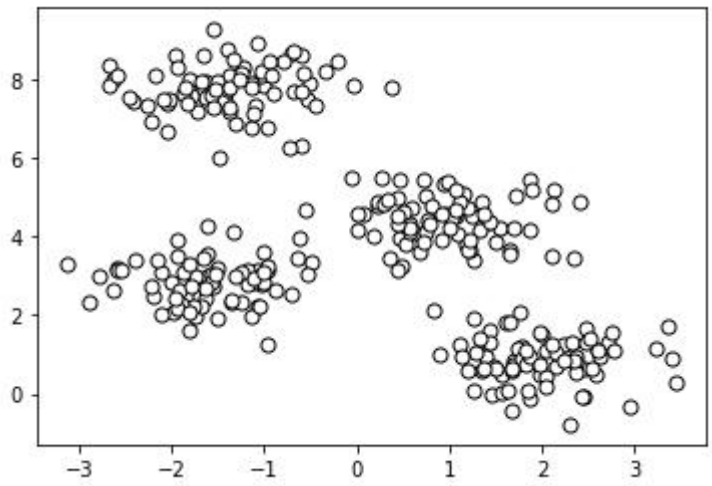
- Scatter the data

Dalam hal ini, data yang telah dibuat akan di "scatter" untuk mencari titik pada axis X1 dan X2.

!Cobalah!

```
import matplotlib.pyplot as plt
# Visualize Data
plt.scatter(
    X[:, 0], X[:, 1],
    c='white', marker='o',
    edgecolor='black', s=50
)
plt.show()
```

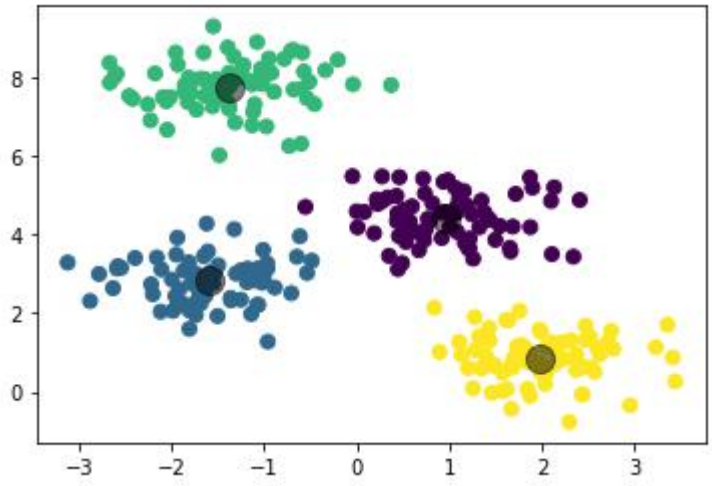
Hasilnya akan terbentuk seperti gambar di bawah ini:



Disaat menerapkan matplotlib.pyplot, anda akan membangun graphics mulai dari layer terbawah ke layer tertinggi lalu akan direnderer apabila memanggil fungsi show(). berikut contohnya seperti ini:
!Cobalah!

```
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=50, cmap='viridis') #Layer 0

centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5) #Layer 1
plt.show()
```




Untuk hendak memperdalamkan library matplotlib atau class pyplot ini, cek dokumentasinya di:
https://matplotlib.org/api/_as_gen/matplotlib.pyplot.html#module-matplotlib.pyplot

3. K-Means

Algoritma K-Means ini berperan sebagai membedakan setiap data pada datasets berbentuk sebagai kelompok berisi n data yang mirip karena nilainya. Metode ini akan membagi N satu set sampel X menjadi K kelompok-kelompok terpisah, masing-masing diuraikan dengan rata-rata sampel dalam cluster atau dinamakan sebagai "centroids". Berikut rumusnya berbentuk seperti di bawah ini:

$$\sum_{i=0}^{n} \min_{\mu_j \in C} (||x_i - \mu_j||^2)$$

K-means sering disebut sebagai algoritma Lloyd. Secara dasar, algoritma ini memiliki tiga langkah. Langkah pertama memilih centroid awal, dengan metode paling dasar adalah memilih sampel k dari dataset X. Setelah

inisialisasi, K-means terdiri dari perulangan di antara dua langkah lainnya. Langkah pertama menetapkan setiap sampel ke pusat massa terdekat. Langkah kedua menciptakan centroid baru dengan mengambil nilai rata-rata dari semua sampel yang ditugaskan untuk setiap centroid sebelumnya. Perbedaan antara centroid lama dan baru dihitung dan algoritma mengulangi dua langkah terakhir ini sampai nilai ini kurang dari ambang. Dalam makna lain, perulangan terus berlanjut sampai centroid tidak berubah secara signifikan.atau mencapai batas perulangan.

i. Implementasi di Scikit-Learn

Untuk menerapkan teknik K-Means clustering di Scikit-Learn, pertama yang dilakukan adalah import dengan cara "**from** sklearn.cluster **import** KMeans" untuk bisa mengakses fungsi nya. Berikut fungsi dan syntaxnya: dan **inisialisasi** import numpy as np

```
KMeans(n_clusters=8, init='k-means++', n_init=10,
max_iter=300, tol=0.0001, precompute_distances='auto',
verbose=0, random_state=None, copy_x=True, n_jobs=None,
algorithm='auto')
```

Berikut penjelasan parameter nya:

**n_clusters** : int, optional, default= 8

Jumlah cluster atau kelompok serta centroids bedasarkan value yang diberikan.

**init** : {'k-means++', 'random' or an ndarray}

Metode inisialisasi, defaultnya adalah 'k-means++'

'k-means++': memilih awalan *cluster centers* untuk k-mean clustering supaya mempercepat performansi konvergensi (data yang disekitar akan bergabung menjadi satu cluster)

**n_init** : int, default: 10

Jumlah waktu algoritma k-means akan dijalankan dengan *centroid seeds* yang berbeda. Hasil akhir akan menjadi output terbaik dari n_init berturut-turut berjalan dalam hal inersia.

**max_iter** : int, default: 300

Jumlah maksimum iterasi algoritma k-means untuk sekali jalan.

**tol** : float, default: 1e-4

Toleransi relatif berkaitan dengan inersia untuk menyatakan konvergensi

**precompute_distances** : {'auto', True, False}

Jarak precompute (lebih cepat tetapi membutuhkan lebih banyak memori).

‘auto’: jangan set precompute distance ke **auto** ketika n_samples * n_clusters> 12 juta. Ini sesuai dengan sekitar 100MB overhead per pekerjaan menggunakan presisi ganda.
‘True’: jarak selalu precompute
‘False’: tidak pernah menghitung jarak sebelumnya

**verbose** : int, default 0
verbosity mode

**random_state** : int, RandomState instance or None (default)
Menentukan generasi nomor acak untuk inisialisasi centroid.

**copy_x** : boolean, optional
Ketika jarak pra-komputasi lebih akurat secara numerik untuk memusatkan data terlebih dahulu. Jika copy_x True (default), maka data asli tidak dimodifikasi, memastikan X berdekatan C. Jika False, data asli dimodifikasi, dan dikembalikan sebelum fungsi kembali, tetapi perbedaan numerik kecil dapat diperkenalkan dengan mengurangi dan kemudian menambahkan rata-rata data, dalam hal ini juga tidak akan memastikan bahwa data bersebelahan C-yang dapat menyebabkan perlambatan yang signifikan.

**n_jobs** : int or None, optional (default=None)
Jumlah pekerjaan yang digunakan untuk perhitungan. Ini bekerja dengan menghitung masing-masing n_init berjalan secara paralel.

**algorithm** : “auto”, “full” or “elkan”, default=”auto”
Algoritma K-means untuk digunakan. Algoritma EM-style klasik adalah "penuh". Variasi "elkan" lebih efisien dengan menggunakan ketimpangan segitiga, tetapi saat ini tidak mendukung data jarang. "auto" akan memilih "elkan" untuk data padat dan "full" untuk data jarang.

Apabila telah mengeksekusi fungsi di atas, maka dapat mengeksekusi metodenya:

```
fit (self, X[, y, sample_weight])
fit_predict (self, X[, y, sample_weight])
fit_transform (self, X[, y, sample_weight])
get_params (self[, deep])
predict (self, X[, sample_weight])
score (self, X[, y, sample_weight])
set_params (self, \*\*params)
transform (self, X)
```

ii. Contoh penggunaan algoritma K-Means

Implementasi K-Means secara sederhana menggunakan python sebagai berikut:

!Cobalah!

```
from sklearn.cluster import KMeans
X = np.array(
    [[1, 2], [1, 4], [1, 0],
    [10, 2], [10, 4], [10, 0]]
    )
kmeans = KMeans(n_clusters=3, random_state=0).fit(X)
#Label Cluster mengetahui urutan data berada di cluster mana
print(kmeans.labels_)
print("Prediksi lokasi data input [0, 0] dan [12, 3]: \n",
      kmeans.predict([[0, 0], [12, 3]])
     )
print("Lokasi centroids: ",
      kmeans.cluster_centers_
     )
```

Sehingga hasilnya seperti ini:

```
[1 1 1 0 0 2]
Prediksi lokasi data input [0, 0] dan [12, 3]:
 [1 0]
Lokasi centroids:  [[10.  3.]
 [ 1.  2.]
 [10.  0.]]
```

!Latihan!

Selanjutnya, gunakan datasets yang diperoleh dari fungsi make_blobs dan visualize hasil clusteringnya

```
kmeans = KMeans(n_clusters=4)
kmeans.fit(X) #datasets make_blobs
y_kmeans = kmeans.predict(X)
```

```
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=50, cmap='viridis') #Layer 0

centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5) #Layer 1
plt.show()
```

iii. Menghitung score clustering

Dalam menghitung score pada machine learning, menggunakan:

- Silhouette score

Teknik perhitungan dengan silhouette score menggunakan nilai jarak rata-rata (mean) *intra-cluster* (a) dan nilai jarak antara sebuah sample dan *nearest-cluster* yang bukan terkait dengannya . Perhitungan Silhouette Coefficient di sebuah sample yaitu (b – a) / max(a, b). sebagai catatan, silhouette berfungsi atau berjalan apabila jumlah label sebesar 2 <= n_labels <= n_samples – 1.

Nilai terbaik adalah 1 dan nilai terburuk adalah -1. Nilai di dekat 0 menunjukkan *overlapping cluster*. Nilai negatif umumnya menunjukkan bahwa sampel telah ditetapkan ke kluster yang salah, karena kluster yang berbeda lebih mirip.

- V measure score

  Pelabelan clusterr V-Measiure memberikan kebenaran mendasar. Nilai skor antara 0.0 dan 1.0. 1.0 mempresentasikan sebagai nilai terbaik

!Cobalah!

```
from sklearn import metrics
from sklearn.preprocessing import scale
#Menggunakan load_digits datasets
digits = load_digits()
X = scale(digits.data)
labels = digits.target
kmeans_digits = KMeans(n_clusters=10, n_init=10)
kmeans_digits.fit(X)

print("Silhouette Score: ",
      metrics.silhouette_score(data, estimator.labels_,
                                       metric='euclidean',
                                       sample_size=300)
     )
print("V Measure Score: ",
      metrics.v_measure_score(labels, estimator.labels_)
     )
```

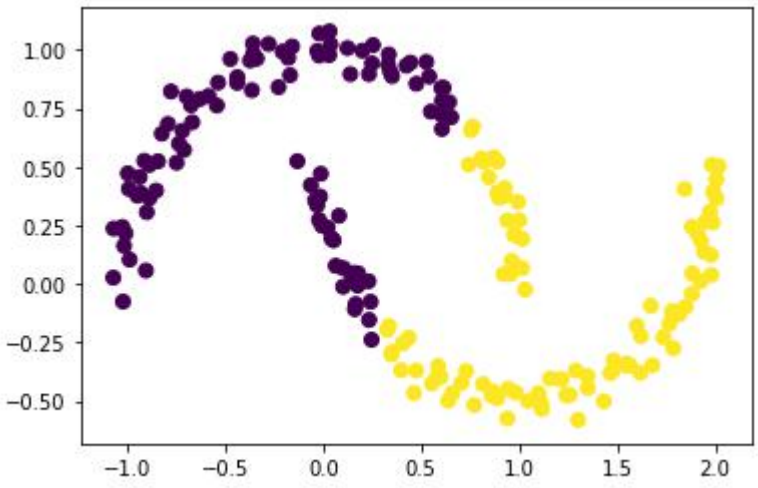
iv. Kernelized K-Means:

Terkadang menggunakan metode clustering K-Means sangat tidak efektif apabila datasets cluster nya memiliki bentuk geometri yang sangat kompleks karena perbatasan antara cluster K-Means selalu berbentuk linear. Berikut bentuk geometri sebuah datasets dan hasil clustering menggunakan k-means biasa.

!Cobalah!

```
from sklearn.datasets import make_moons
moon_X, moon_y = make_moons(200, noise=0.05, random_state=0)
```

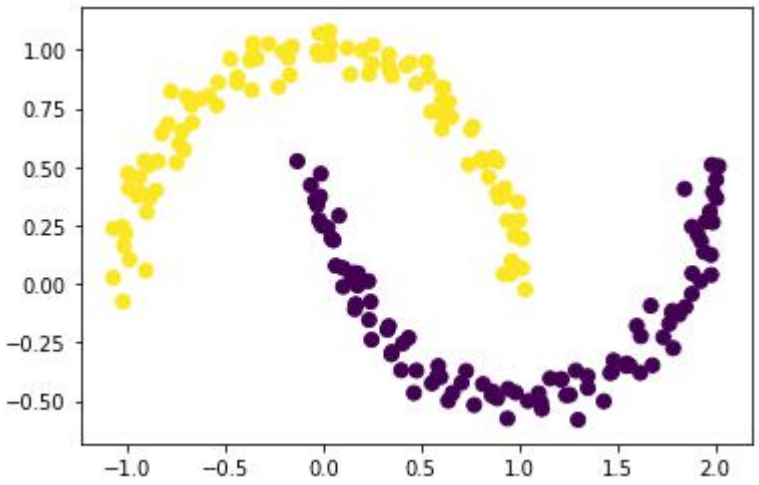
Lalu hasil graphic nya setelah di fit dan predict:




Maka scikit-learn menyediakan SpectralClustering estimator menerapkan nearest-neighbors untuk komputasi representasi *higher-dimensional* sebuah data, lalu menentukan labelnya menggunakan k-means.

!Cobalah!

```
from sklearn.cluster import SpectralClustering
model = SpectralClustering(n_clusters=2, affinity='nearest_neighbors',
                           assign_labels='kmeans')
labels = model.fit_predict(moon_X)
plt.scatter(moon_X[:, 0], moon_X[:, 1], c=labels,
            s=50, cmap='viridis');
```




## D. TUGAS

1. Carilah datasets sederhana (nilai mata kuliah, tiap kelompok harus berbeda) yang terdapat hasil klasifikasi berjumlah minimal 3 label, datasets yang akan diuji atau dilatih sebesar 30 + (2 digit NIM terakhir).
2. Sebelum melakukan *fit* dan *predict* cari korelasi antar pasangan sebuah attribut yang tersedia dalam datasets, pilih pasangan atribut yang

memiliki nilai korelasi paling tinggi (mendekati nilai positif 1), dan scatter plot nya mirip dengan plot figures di bawah ini:

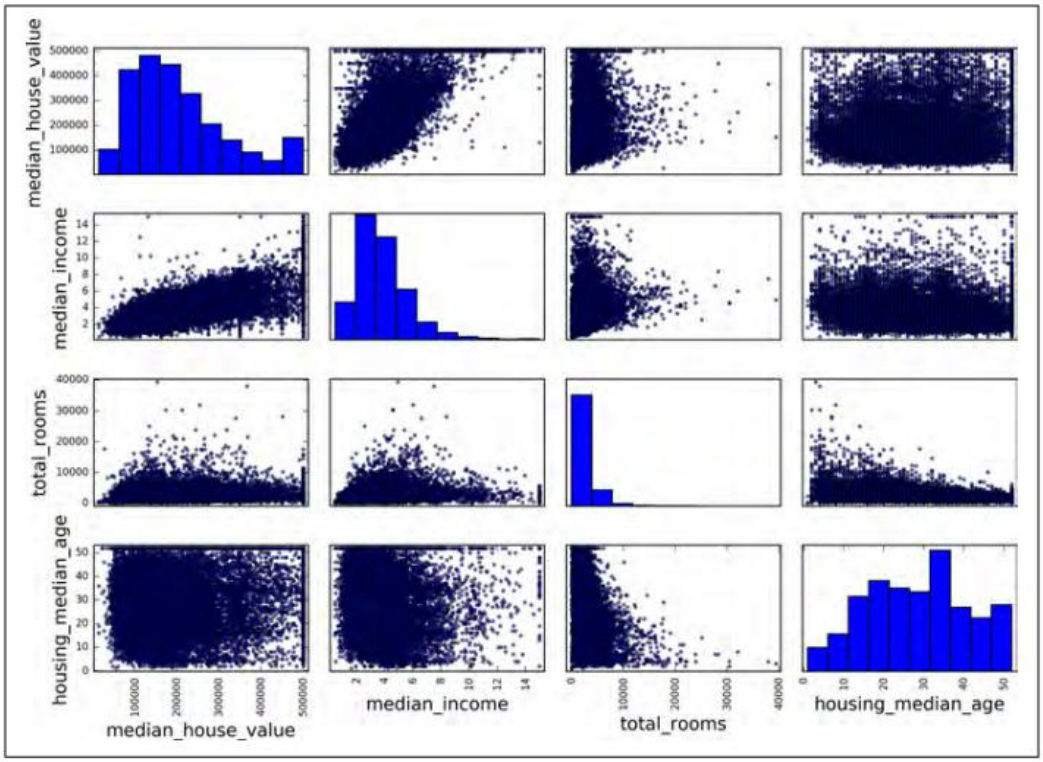



*Figure 2-15. Scatter matrix*

3. Fit dan predict datasets menggunakan model K-Means dengan metode ‘kmeans++’ dan ‘random’. Cek hasil scorenya menggunakan ‘Silhouette score’