

**LAPORAN UJIAN AKHIR SEMESTER
MATA KULIAH ALGORITMA EVOLUSI
ALGORITMA *FIREFLY* & K-MEANS**

Dosen Pengampu: Saiful Nur Budiman, S.Kom., M.Kom



Kelompok 2:

Virganda Rimba Asmara	22104410010
Khariratul Istiqlaliyah	22104410026
Afiana Septi Laili	22104410032
Nimas Ayu Anggun K	22104410051
Muhamad Gus Amix K	22104410054

**UNIVERSITAS ISLAM BALITAR
FAKULTAS TEKNIK DAN INFORMATIKA
PROGRAM STUDI TEKNIK INFORMATIKA
Januari 2026**

Kerjakan:

- a) Buatkan sampel perhitungan manualnya di **excel**.
- b) Selesaikan kasus tersebut dengan bahasa pemrograman python. Coding di upload di github.
- c) Buat Laporan dimana jelaskan dulu peran FA untuk optimasi K-Means. Jelaskan juga langkahnya serta hasil penjelasan coding. (hitungan sampel boleh di screenshot dimasukkan di laporan). Pada halaman cover tulis nama anggota kalian. Laporan diupload di github dalam bentuk .pdf.
- d) Buat video pembelajarannya mengenai hasil pekerjaan kalian. Setiap mahasiswa harus on-cam menjelaskan kontribusinya dan menunjukkan hasilnya. Durasi video maksimal 15 menit. Video diunggah di **youtube**.
- e) Link github dan link youtube di submit di Edlink.

PERANAN ALGORITMA *FIREFLY* UNTUK OPTIMASI K-MEANS

K-Means merupakan algoritma yang termasuk *unsupervised learning* untuk pengelompokan data yang mengelompokkan titik data tidak berlabel ke dalam kelompok atau klaster tertentu. Data pelatihan yang digunakan tidak berlabel artinya, titik data tidak memiliki struktur klasifikasi yang ditentukan. K-Means merupakan algoritma pengelompokan berbasis *centroid* berulang yang membagi kumpulan data menjadi kelompok serupa berdasarkan jarak antara *centroid* mereka.

Sementara itu, algoritma *Firefly* merupakan algoritma optimasi metaheuristik yang mensimulasikan daya tarik antar kunang-kunang berdasarkan kecerahan untuk menemukan solusi optimal dari masalah yang kompleks. Kecerahan yang dimaksud mencerminkan kualitas solusi dan tertarik pada yang lebih terang.

K-Means dapat dioptimasi dengan algoritma *Firefly* karena proses utama dari K-Means pada dasarnya adalah masalah optimasi, yaitu meminimalkan nilai dari *Sum of Squared Error* (SSE) antara data dan *centroid* klasternya. Algoritma *Firefly* dirancang menyelesaikan masalah optimasi global dengan cara melacak ruang solusi secara lebih luas dan tidak bergantung pada titik awal tertentu. Sementara K-Means cenderung bersifat optimasi lokal dan sangat dipengaruhi oleh inisialisasi *centroid* awalnya, sehingga sering terjebak pada solusi lokal saja dan kurang optimal.

Dengan dimanfaatkannya algoritma *Firefly*, pencarian pada *centroid* awal dilakukan secara global berdasarkan kualitas solusinya (nilai SSE). Solusi yang lebih baik akan menarik solusi lain untuk mendekat, sehingga kombinasi ini membuat *Firefly* mampu menemukan posisi dari *centroid* yang lebih baik sebelum proses K-Means dijalankan. Dampaknya K-Means akan bekerja lebih stabil, konvergen lebih cepat, dan menghasilkan kualitas klaster yang lebih optimal.

LANGKAH-LANGKAH PERHITUNGAN EXCEL

Data yang digunakan dalam tugas ini terdiri dari 12 data dengan 4 atribut, yaitu A1, A2, A3, dan A4, sebagaimana ditampilkan pada soal dan kami tulis ulang di tabel bawah ini. Data tersebut merupakan data numerik yang digunakan sebagai dasar dalam proses perhitungan *clustering*. Setiap baris merepresentasikan satu data, sedangkan setiap kolom menunjukkan nilai atribut yang dimiliki oleh data tersebut. Seluruh data ini digunakan pada tahap perhitungan jarak, penentuan *cluster*, serta pembaruan *centroid* pada proses selanjutnya.

DATA ASLI	DATA AWAL	1		
Data	A1	A2	A3	A4
D1	1	2	1	2
D2	1	1	2	2
D3	2	2	1	1
D4	2	1	2	1
D5	8	8	9	8
D6	9	8	8	9
D7	8	9	8	8
D8	9	9	9	8
D9	4	5	4	5
D10	5	4	5	4
D11	4	4	5	5
D12	5	5	4	4

Selanjutnya, pada metode *Firefly*, satu *Firefly* merepresentasikan satu solusi klastering. Setiap *Firefly* terdiri dari tiga *centroid*, yaitu C1, C2, dan C3, di mana setiap *centroid* memiliki empat atribut (A1, A2, A3, dan A4). Nilai-nilai *centroid* ini merupakan posisi awal pusat klaster yang digunakan dalam proses perhitungan. Pada penelitian ini digunakan dua *Firefly*, yaitu *Firefly* 1 dan *Firefly* 2. Masing-masing *Firefly* menyimpan posisi *centroid* C1, C2, dan C3 dalam ruang berdimensi empat sesuai dengan jumlah atribut data. Posisi *centroid* pada kedua *Firefly* ini berfungsi sebagai *centroid* awal yang selanjutnya digunakan untuk menghitung jarak data dan mengevaluasi kualitas klastering berdasarkan nilai SSE.

Firefly 1

H	I	J	K	L
Tabel Centroid Awal Firefly (1)			2	
Cluster	A1	A2	A3	A4
C1	1	2	1	2
C2	8	8	8	8
C3	4	5	4	5

Firefly 2

N	O	P	Q	R
FIREFLY2	3			
Cluster	A1	A2	A3	A4
C1	2	1	2	1
C2	9	9	9	8
C3	5	4	5	4

Langkah pertama adalah menghitung jarak setiap data terhadap nilai *centroid* pada 1 *Firefly*. Jarak yang digunakan adalah jarak *Eclidean* yang rumusnya menghitung selisih nilai atribut pada data dengan nilai atribut pada *centroid Firefly*, kemudian dijumlahkan, lalu diakarkan. Setiap data akan diperoleh 3 nilai jarak, masing-masing terhadap C1, C2, C3.

Berikut adalah rumus yang digunakan untuk menghitung jarak:

$$d_{ij} = \sqrt{(A1_i - A1_{Cj})^2 + (A2_i - A2_{Cj})^2 + (A3_i - A3_{Cj})^2 + (A4_i - A4_{Cj})^2}$$

Dengan:

- d_{ij} = jarak data ke-*i* terhadap *centroid* ke-*j*
- $A1_i, \dots, A4_i$ = nilai atribut data
- $A1_{Cj}, \dots, A4_{Cj}$ = nilai atribut *centroid*

Hasil rumus tersebut adalah sebagai berikut.

T	U	V	W	X	Y	Z	AA	AB	AC	AD
JARAK ECLUIDEAN		F1	MINIMUM	4	JARAK ECLUIDEAN		F2	MINIMUM		
d_C1	d_C2	d_C3	Min	SSE	d_C1	d_C2	d_C3	Min	SSE	
0	13.0384	6	0	0	2	14.595	6.3246	2	4	
1.4142	13.0384	6.1644	1.4142	2	1.4142	14.595	6.1644	1.4142	2	
1.4142	13.0384	6.1644	1.4142	2	1.4142	14.526	6.1644	1.4142	2	
2	13.0384	6.3246	2	4	13.6015	1	7.6811	1	1	
13.6015	1	7.6811	1	1	14.0712	1.4142	8.1240	1.4142	2	
14.0712	1.4142	8.1240	1.4142	2	13.5277	1	7.5498	1	1	
13.5277	1	7.5498	1	1	14.5945	1.7321	8.1240	1.7321	3	
14.5945	1.7321	8.6603	1.7321	3	13.6015	1.4142	7.6811	1.4142	2	
6	7.0711	0	0	0	14.5258	0	8.5440	0	0	
6.3246	7.0711	2	2	4	6.3246	8.6603	2	2	4	
6.1644	7.0711	1.4142	1.4142	2	6	8.5440	0	0	0	
6.1644	7.0711	1.4142	1.4142	2	6.1644	8.6603	1.4142	1.4142	2	
					6.1644	8.5440	1.4142	1.4142	2	
							SSE=	23	23	
Tabel Jarak Data ke Centroid (Firefly Awal)										

Sementara itu, untuk mendapatkan nilai SSE, rumus yang digunakan adalah penjumlahan seluruh nilai minimum dari jarak *Eclidean* yang didapatkan. Rumusnya adalah sebagai berikut:

$$SSE = \sum_{i=1}^n \min ||x_i - c_k||^2$$

Dengan:

- x_i = data ke- i
- c_k = pusat (*centroid*) klaster ke- k
- $||x_i - c_k||^2$ = jarak *Eclidean* kuadrat antara data dan pusat klaster
- n = jumlah seluruh data

Selanjutnya, setelah proses perhitungan jarak data terhadap masing-masing *centroid* selesai, dilakukan pembaruan posisi *centroid* menggunakan algoritma *Firefly*. Tahap ini bertujuan untuk memperbaiki posisi *centroid* agar menghasilkan kualitas *clustering* yang lebih baik, yang ditunjukkan oleh nilai SSE yang lebih kecil. Pada proses ini, setiap *centroid* dianggap sebagai sebuah *firefly*. *Centroid* dengan kualitas lebih rendah akan bergerak mendekati *centroid* lain yang memiliki kualitas lebih baik. Pergerakan ini dilakukan secara bertahap dengan mempertimbangkan tingkat ketertarikan antar *centroid* serta faktor acak untuk menghindari solusi lokal.

Proses pembaruan *centroid* mengikuti persamaan berikut:

$$X_{baru} = X_{lama} + \beta(X_{tujuan} - X_{lama}) + \alpha\varepsilon$$

Dengan:

- X_{lama} = centroid awal
- X_{tujuan} = centroid tujuan
- β = nilai *attractiveness*

- α = faktor randomisasi
- ε = Nilai acak

Hasil dari perhitungan tersebut sebagai berikut.

5				
Tabel Centroid Baru (Hasil Update Firefly Algorithm)				
Cluster	A1	A2	A3	A4
C1	1.5	1.5	1.6	1.5
C2	8.6	8.6	8.6	7.9
C3	4.4	4.5	4.5	4.5

Tabel tersebut menampilkan hasil perhitungan yang telah dilakukan berdasarkan nilai *random* yang digunakan dalam proses perhitungannya. Nilai *random* ini dapat berubah setiap kali proses dijalankan, baik akibat pelaksanaan langkah selanjutnya maupun perubahan di langkah sebelumnya. Oleh karenanya hasil perhitungan yang diperoleh bersifat tidak tetap. Namun demikian, variasi nilai tersebut tidak memberikan pengaruh signifikan terhadap hasil akhirnya, karena pola atau kecenderungan utama perhitungan tetap konsisten.

Langkah selanjutnya adalah perhitungan *centroid* yang kedua. Berikut penjelasannya dan rumus-rumus yang digunakan.

Tabel Perhitungan Jarak Data ke Centroid Baru (Hasil Firefly)					Tabel Perhitungan Jarak Data ke Centroid K-Means					
d_C1new	d_C2new	d_C3new	Minnew	6	d_C1_KM	d_C1_KM	d_C3_KM	CLSTR	SSE_data	7
1.0490	13.8863	6.0103	1.0490	1.1004	1.0490	13.8863	6.0103	C1	1.0490	1.1004
0.9457	13.8873	6.0032	0.9457	0.8944	0.9457	13.8873	6.0032	C1	0.9457	0.8944
1.0602	13.8414	6.0212	1.0602	1.1240	1.0602	13.8414	6.0212	C1	1.0602	1.1240
0.9581	13.8424	6.0141	0.9581	0.9180	0.9581	13.8424	6.0141	C1	0.9581	0.9180
13.4623	0.9179	7.6154	0.9179	0.8425	13.4623	0.9179	7.6154	C2	0.9179	0.8425
13.9746	1.4000	8.1410	1.4000	1.9601	13.9746	1.4000	8.1410	C2	1.4000	1.9601
13.4699	0.9028	7.6210	0.9028	0.8151	13.4699	0.9028	7.6210	C2	0.9028	0.8151
14.4646	0.7554	8.6166	0.7554	0.5706	14.4646	0.7554	8.6166	C2	0.7554	0.5706
6.0298	7.9384	0.9944	0.9944	0.9888	6.0298	7.9384	0.9944	C3	0.9944	0.9888
6.0147	7.8613	1.0168	1.0168	1.0339	6.0147	7.8613	1.0168	C3	1.0168	1.0339
6.0127	7.9401	0.9502	0.9502	0.9028	6.0127	7.9401	0.9502	C3	0.9502	0.9028
6.0318	7.8596	1.0582	1.0582	1.1199	6.0318	7.8596	1.0582	C3	1.0582	1.1199
SSE=		12.2704			SSE=		12.2704		12.2704	

Tabel di atas menunjukkan perhitungan jarak data ke *centroid* baru hasil algoritma *Firefly*. Perhitungan ini dilakukan untuk mengecek apakah posisi *centroid* yang telah diperbarui oleh *Firefly* menghasilkan pembagian *cluster* yang lebih baik atau sudah stabil. Jarak dihitung menggunakan rumus *Euclidean* yang sama, namun *centroid* yang digunakan adalah *centroid* hasil update *Firefly*, bukan *centroid* awal. Data kemudian dibandingkan kembali untuk melihat kemungkinan perubahan *cluster*. Berdasarkan hasil perhitungan, tidak terjadi perubahan *cluster* sehingga dapat disimpulkan bahwa hasil *clustering* sudah konvergen.

$$d = \sqrt{(X_1 - C_1)^2 + (X_2 - C_2)^2 + (X_3 - C_3)^2 + (X_4 - C_4)^2}$$

Setelah mendapatkan *centroid* yang pertama, selanjutnya adalah menghitung nilai jarak *centroid* yang kedua. Hasil dari perhitungan menunjukkan tabel di bawah ini.

AL	AM	AN	AO	AP	AQ
Tabel Perhitungan Jarak Data ke Centroid K-Means					
d_C1_KM	d_C1_KM	d_C3_KM	CLSTR	SSE_data	7
1.0490	13.8863	6.0103	C1	1.0490	1.1004
0.9457	13.8873	6.0032	C1	0.9457	0.8944
1.0602	13.8414	6.0212	C1	1.0602	1.1240
0.9581	13.8424	6.0141	C1	0.9581	0.9180
13.4623	0.9179	7.6154	C2	0.9179	0.8425
13.9746	1.4000	8.1410	C2	1.4000	1.9601
13.4699	0.9028	7.6210	C2	0.9028	0.8151
14.4646	0.7554	8.6166	C2	0.7554	0.5706
6.0298	7.9384	0.9944	C3	0.9944	0.9888
6.0147	7.8613	1.0168	C3	1.0168	1.0339
6.0127	7.9401	0.9502	C3	0.9502	0.9028
6.0318	7.8596	1.0582	C3	1.0582	1.1199
		SSE=		12.2704	12.2704

Tabel tersebut menunjukkan proses perhitungan jarak setiap data terhadap *centroid* hasil *K-Means*. Tujuan dari tabel ini adalah untuk mengetahui seberapa dekat masing-masing data dengan setiap *centroid*, sehingga dapat ditentukan data tersebut masuk ke *cluster* yang mana. Pada tabel ini terdapat tiga kolom jarak, yaitu d_C1_KM, d_C2_KM, dan d_C3_KM. Kolom-kolom tersebut menunjukkan jarak data ke *centroid cluster 1*, *cluster 2*, dan *cluster 3*. Perhitungan jarak dilakukan menggunakan rumus jarak *Euclidean*.

$$d = \sqrt{(X_1 - C_1)^2 + (X_2 - C_2)^2 + (X_3 - C_3)^2 + (X_4 - C_4)^2}$$

Rumus ini digunakan karena data memiliki beberapa atribut, sehingga jarak dihitung dengan mengukur selisih setiap atribut data terhadap atribut *centroid*, kemudian dijumlahkan dan diakarkan. Rumus tersebut kemudian diterapkan ke dalam *Excel* dengan cara menghitung selisih nilai setiap atribut data dengan atribut *centroid*, menguadratkan selisih tersebut, menjumlahkannya, lalu diakarkan. Hasil dari perhitungan ini ditampilkan pada kolom d_C1_KM, d_C2_KM, dan d_C3_KM.

Setelah jarak ke setiap *centroid* diperoleh, langkah selanjutnya adalah menentukan *cluster* untuk setiap data. Penentuan *cluster* ditampilkan pada kolom CLSTR. Data dimasukkan ke dalam *cluster* yang memiliki nilai jarak paling kecil. Sebagai contoh, jika nilai d_C1_KM lebih kecil dibandingkan d_C2_KM dan d_C3_KM, maka data tersebut dimasukkan ke dalam

cluster C1. Kolom SSE_data menunjukkan nilai kontribusi SSE dari setiap data. Nilai ini diperoleh dari kuadrat jarak data ke *centroid cluster* yang dipilih. Nilai SSE_data digunakan untuk mengetahui seberapa besar kesalahan data terhadap *centroid clusternya*. Rumus yang digunakan sama seperti sebelumnya, namun disederhanakan seperti berikut.

$$SSE = \sum d^2$$

Artinya, semua jarak data ke *centroid* yang terpilih dikuadratkan kemudian dijumlahkan. Pada tabel ini, nilai (SSE_data) dari seluruh data dijumlahkan sehingga diperoleh nilai SSE total sebesar 12,28 yang ditampilkan pada bagian bawah tabel. Nilai SSE ini digunakan sebagai ukuran kualitas *clustering*. Semakin kecil nilai SSE, maka hasil pengelompokan data semakin baik karena data semakin dekat dengan centroid clusternya masing-masing.

PENJELASAN CODING

KMEANS_FF.PY

```
import numpy as np
from utils import Utils

class FireflyClustering:

    def __init__(self, X, firefly1, firefly2, alpha=0.2, beta=0.5):
        self.X = X
        self.firefly1 = firefly1
        self.firefly2 = firefly2
        self.alpha = alpha
        self.beta = beta

    def iterasi_1(self):
        print("\n" + "="* 95)
        print(f"{'ITERASI 1 (FIREFLY AWAL)':^95}")
        print("=".* 95)
```

```

print("\nCentroid Firefly 1:")
print(self.firefly1)
sse1 = Utils.hitung_sse_dan_tabel(
    self.X, self.firefly1, "HASIL FIREFLY 1 - ITERASI 1"
)

print("\nCentroid Firefly 2:")
print(self.firefly2)
sse2 = Utils.hitung_sse_dan_tabel(
    self.X, self.firefly2, "HASIL FIREFLY 2 - ITERASI 1"
)

return sse1, sse2

def update_firefly(self, sse1, sse2):
    print("\n" + "="* 95)
    print(f"{'EVALUASI BRIGHTNESS':^95}")
    print("=".* 95)

    if sse1 <= sse2:
        bright, dim = self.firefly1, self.firefly2
        dim_id = 2
        print("Firefly 1 → TERANG")
        print("Firefly 2 → REDUP")
    else:
        bright, dim = self.firefly2, self.firefly1
        dim_id = 1
        print("Firefly 2 → TERANG")
        print("Firefly 1 → REDUP")

    dim_new = dim + self.beta * (bright - dim) \
        + self.alpha * (np.random.rand(dim.shape) - 0.5)

```

```

dim_new = np.round(dim_new, 3)

if dim_id == 1:
    self.firefly1 = dim_new
else:
    self.firefly2 = dim_new

def iterasi_2(self):
    print("\n" + "=" * 95)
    print(f'{ITERASI 2 (SETELAH UPDATE)':'^95}')
    print("=" * 95)

    print("\nCentroid Firefly 1:")
    print(self.firefly1)
    sse1 = Utils.hitung_sse_dan_tabel(
        self.X, self.firefly1, "HASIL FIREFLY 1 - ITERASI 2"
    )

    print("\nCentroid Firefly 2:")
    print(self.firefly2)
    sse2 = Utils.hitung_sse_dan_tabel(
        self.X, self.firefly2, "HASIL FIREFLY 2 - ITERASI 2"
    )

    return sse1, sse2

def kesimpulan(self, sse1, sse2):
    print("\n" + "=" * 95)
    print(f'{KESIMPULAN':'^95}')
    print("=" * 95)

best = "Firefly 1" if sse1 <= sse2 else "Firefly 2"

```

```
best_sse = min(sse1, sse2)

print(f"""
Firefly terbaik adalah {best}
karena memiliki SSE paling kecil ({best_sse:.3f}).

Firefly Algorithm mengoptimasi centroid
dengan memilih brightness tertinggi (SSE minimum).
""")
```

Dari *syntax* di atas dapat dilihat alur program yaitu inisialisasi data dan *centroid* awal, kemudian menghitung nilai SSE awal (Iterasi 1), selanjutnya menentukan *Firefly* yang terang dan redup, memperbarui *centroid* *Firefly* yang redup, menghitung ulang SSE (Iterasi 2), terakhir adalah mengambil solusi terbaik.

Kode pada `*kmeans_ff.py*` ini berfungsi sebagai modul inti implementasi algoritma *Firefly* klastering yang bertugas mengelola proses evaluasi dan pembaruan solusi klastering berbasis *centroid*. Dengan memanfaatkan prinsip algoritma *Firefly* yang setiap kunang-kunangnya merepresentasikan solusi kandidat berupa sekumpulan *centroid*, dan kualitas solusi diukur dengan nilai SSE. Semakin kecil SSE, semakin baik kualitas pengelompokan hasil.

Di bagian awal, *library* Numpy diimpor untuk mendukung operasi numerik dan manipulasi *array* yang diperlukan dalam proses perhitungan jarak, pembaruan *centroid*, dan pembangkitan nilai acak. Kelas `*Utils*` diimpor sebagai modul pendukung yang menyediakan fungsi perhitungan SSE sekaligus penyajian tabel hasil klastering. Fungsi ini dipisahkan untuk tujuan proses perhitungan utama tetap terstruktur dan mudah dipahami.

Kelas `*FireflyClustering*` diartikan sebagai kerangka utama algoritma. Melalui metode inisialisasi `(*__init__*)`, kelas ini menerima dataset, dua *Firefly* awal sebagai solusi awal, serta parameter `alpha` dan `beta`. Parameter `alpha` mengontrol tingkat keacakan dalam pembaruan posisi *Firefly* untuk mendukung eksplorasi solusinya. Sedangkan parameter `beta` mengatur tingkat ketertarikan *Firefly* redup terhadap *Firefly* yang lebih terang. Seluruh parameter ini disimpan sebagai atribut kelas untuk digunakan pada setiap tahap iterasi.

Pembaruan solusi dilakukan pada fungsi `update_firefly()`, yang merupakan inti dari algoritma *Firefly*. Fungsi ini membandingkan nilai SSE dari kedua *Firefly* untuk menentukan tingkat *brightness*. *Firefly* dengan nilai SSE lebih besar dianggap lebih redup dan posisi diperbarui dengan bergerak mendekati *Firefly* yang lebih terang. Proses pembaruan dilakukan dengan mengombinasikan arah pergerakan menuju solusi yang terbaik dan komponen acak yang dikendalikan oleh parameter `alpha` tadi. Sehingga algoritma mampu menyeimbangkan eksplorasi dan eksplorasi solusi.

Setelah proses pembaruan *centroid* dilakukan, fungsi `iterasi_20` dijalankan untuk mengevaluasi lagi hasil klastering menggunakan posisi *Firefly* yang telah diperbarui sebelumnya. Pada tahap ini, nilai SSE dihitung ulang untuk masing-masing *Firefly* guna mengukur peningkatan kualitas solusi dibandingkan dengan iterasi sebelumnya. Hasil evaluasi ini akan menunjukkan efektivitas mekanisme algoritma *Firefly* dalam memperbaiki hasil klastering.

Kemudian di tahap akhir dari proses ini dilakukan melalui fungsi `kesimpulan()`, yang bertugas menentukan solusi klastering terbaik berdasarkan nilai SSE terkecil pada iterasi terakhir. *Firefly* dengan nilai SSE paling rendah akan dipilih sebagai solusi optimal, karena merepresentasikan pembagian klaster dengan tingkat kesalahan minimum.

MAIN.PY

```
import numpy as np
from Kmeans_ff import FireflyClustering

np.set_printoptions(precision=3, suppress=True)
np.random.seed(1)

# DATASET
X = np.array([
    [1,2,1,2],[1,1,2,2],[2,2,1,1],[2,1,2,1],
    [8,8,9,8],[9,8,8,9],[8,9,8,8],[9,9,9,8],
    [4,5,4,5],[5,4,5,4],[4,4,5,5],[5,5,4,4]
])

# INISIAL FIREFLY
```

```

firefly1 = np.array([[1,2,1,2],[8,8,8,8],[4,5,4,5]])
firefly2 = np.array([[2,1,2,1],[9,9,8,8],[5,4,5,4]])

# JALANKAN MODEL
model = FireflyClustering(X, firefly1, firefly2)

sse1_i1, sse2_i1 = model.iterasi_1()
model.update_firefly(sse1_i1, sse2_i1)
sse1_i2, sse2_i2 = model.iterasi_2()
model.kesimpulan(sse1_i2, sse2_i2)

```

File `main.py` berfungsi sebagai pengendali utama (driver program) yang menjalankan proses clustering menggunakan algoritma *Firefly* Clustering berbasis K-Means. Pada bagian awal, *library NumPy* diimpor untuk mendukung operasi numerik dan manipulasi array, sedangkan kelas `FireflyClustering` diimpor dari file `Kmeans_ff` sebagai modul inti algoritma. Pengaturan `set_printoptions` digunakan agar hasil perhitungan ditampilkan dengan tiga angka desimal tanpa notasi ilmiah, sedangkan `random.seed(1)` bertujuan untuk menjaga konsistensi hasil eksekusi. Dataset `X` didefinisikan sebagai kumpulan data numerik dua dimensi yang terdiri dari 12 data dengan 4 atribut. Dataset ini menjadi input utama proses clustering dan merepresentasikan data yang akan dikelompokkan berdasarkan kedekatan jarak terhadap *centroid*.

Selanjutnya, dua variabel `firefly1` dan `firefly2` diinisialisasi sebagai solusi awal firefly, di mana masing-masing berisi tiga centroid yang menandakan jumlah cluster yang akan dibentuk. Perbedaan posisi centroid awal ini digunakan untuk membandingkan kualitas solusi berdasarkan nilai kesalahan clustering.

Objek `model` dibuat dari kelas `FireflyClustering` dengan memasukkan dataset dan dua firefly awal. Objek ini bertugas mengatur seluruh proses clustering, evaluasi solusi, serta pembaruan posisi *centroid*. Pemanggilan fungsi `iterasi_1()` digunakan untuk menjalankan proses clustering pada iterasi pertama dan menghasilkan nilai *Sum of Squared Error (SSE)* untuk masing-masing firefly sebagai ukuran kualitas pengelompokan. Fungsi `update_firefly()` berfungsi untuk memperbarui posisi centroid berdasarkan perbandingan nilai SSE hasil iterasi

pertama, di mana *firefly* dengan kualitas lebih rendah akan bergerak mendekati *firefly* dengan kualitas lebih baik. Setelah pembaruan dilakukan, fungsi `iterasi_20` dijalankan untuk mengevaluasi kembali hasil clustering menggunakan posisi centroid yang telah diperbarui.

Terakhir, fungsi `kesimpulan()` digunakan untuk menentukan solusi clustering terbaik berdasarkan nilai SSE terkecil pada iterasi akhir. Hasil ini merepresentasikan pembagian cluster yang paling optimal sesuai dengan data yang digunakan. Dengan demikian, `main.py` berperan sebagai pengatur alur eksekusi algoritma tanpa mengandung detail implementasi internal perhitungan.

UTILS.PY

```
import numpy as np

class Utils:

    @staticmethod
    def euclidean(a, b):
        return np.sqrt(np.sum((a - b) ** 2))

    @staticmethod
    def hitung_sse_dan_tabel(X, centroids, label):
        print("\n" + "-" * 95)
        print(f"{'label':^95}")
        print("-" * 95)
        print(f"{'Data':<6} {'C1':>12} {'C2':>12} {'C3':>12}" +
              f"{'Cluster':>12} {"Jarak^2":>12}")
        print("-" * 95)

        sse = 0
        for i, x in enumerate(X):
            jarak = [round(Utils.euclidean(x, c), 4) for c in centroids]
            idx = np.argmin(jarak)
            jarak2 = round(jarak[idx] ** 2, 4)
```

```

sse += jarak2

print(f'D{i+1:<5}'
      f'{jarak[0]:>12.4f}'
      f'{jarak[1]:>12.4f}'
      f'{jarak[2]:>12.4f}'
      f'{idx+1:>12}'
      f'{jarak2:>12.4f}')

print("-" * 95)
print(f'TOTAL SSE:>70} = {sse:.3f}')
print("-" * 95)

return sse

```

Penjelasan :

Pada bagian awal kode digunakan *library numpy* yang disingkat menjadi np. *library* ini membantu dalam melakukan perhitungan matematika seperti penjumlahan, pengurangan, pemangkatan, dan akar kuadrat. Penggunaan numpy memudahkan perhitungan jarak antar data dengan centroid.

“Class Utils” berfungsi sebagai class bantu yang menyediakan fungsi-fungsi pendukung dalam proses clustering. Class ini dibuat agar perhitungan jarak dan nilai SSE dapat digunakan kembali di bagian program lain tanpa harus menuliskan ulang kode yang sama. Seluruh fungsi di dalam class ini menggunakan decorator **@staticmethod** agar dapat dipanggil langsung tanpa membuat objek dari *class Utils*.

Pada Fungsi “**euclidean(a, b)**” digunakan untuk menghitung jarak Euclidean antara satu data (a) dan satu *centroid* (b). Pada fungsi ini, (a - b) menunjukkan selisih antara data dan centroid pada setiap atribut. Selisih tersebut kemudian dipangkatkan dengan angka 2, yang berarti setiap selisih dikuadratkan untuk menghilangkan nilai negatif dan mengikuti rumus jarak *euclidean*. Hasil kuadrat tersebut dijumlahkan menggunakan “**np.sum()**”, lalu diambil akar kuadratnya dengan “**np.sqrt()**” untuk memperoleh jarak sebenarnya. Nilai jarak ini menunjukkan tingkat kedekatan data terhadap *centroid*.

Fungsi “`hitung_sse_dan_tabel(X, centroids, label)`” digunakan untuk menghitung nilai SSE sekaligus menampilkan hasil perhitungan dalam bentuk tabel. Pada awal fungsi, ditampilkan judul dan garis pembatas dengan panjang 95 karakter yang bertujuan untuk merapikan tampilan output. Header tabel ditampilkan dengan pengaturan lebar kolom tertentu agar setiap informasi seperti data, jarak ke *centroid*, *cluster*, dan jarak kuadrat dapat terbaca dengan jelas.

Nilai SSE diinisialisasi dengan angka **0** karena SSE merupakan hasil penjumlahan jarak kuadrat dari seluruh data. Selanjutnya dilakukan perulangan terhadap setiap data menggunakan “`enumerate(X)`”, di mana i adalah indeks data dan x adalah nilai data tersebut. Indeks data ditambah **1** saat ditampilkan agar penomoran dimulai dari satu dan lebih mudah dipahami. Angka 4 pada “`round(..., 4)`” berarti hasil jarak dibulatkan hingga 4 angka di belakang koma agar lebih rapi dan mudah dibaca.

Setelah jarak dihitung, data akan dimasukkan ke dalam *cluster* yang memiliki jarak paling kecil. Jarak terdekat tersebut kemudian dikuadratkan sesuai dengan konsep SSE, dan hasilnya ditambahkan ke dalam total SSE. Proses ini dilakukan untuk seluruh data sehingga diperoleh nilai SSE keseluruhan.

Nilai SSE yang dihasilkan oleh fungsi ini dikembalikan sebagai *output*. Nilai tersebut digunakan untuk menilai kualitas *clustering*, di mana semakin kecil nilai SSE maka hasil pengelompokan data dianggap semakin baik.

HASIL RUNNING TERMINAL

Gambar pertama adalah tampilan awal dari program saat dijalankan pada terminal. Di gambar ini diperlihatkan Iterasi 1 (*Firefly Awal*) dengan *centroid firefly* 1 ([1,2,1,2];[8,8,8,8];[4,5,4,5]). Lalu terdapat hasil *Firefly* 1 dari Iterasi ke-1 dengan total nilai SSE 23.

```

PS E:\KULIAH\SEMESTER 7\ALGORITMA EVOLUSI\uasss\kmeans_firefly_algorithm> python main.py
=====
ITERASI 1 (FIREFLY AWAL)
=====

Centroid Firefly 1:
[[1 2 1 2]
 [8 8 8 8]
 [4 5 4 5]]

-----
HASIL FIREFLY 1 - ITERASI 1
-----
Data      C1        C2        C3        Cluster    Jarak^2
-----
D1      0.0000    13.0384   6.0000     1       0.0000
D2      1.4142    13.0384   6.1644     1       2.0000
D3      1.4142    13.0384   6.1644     1       2.0000
D4      2.0000    13.0384   6.3246     1       4.0000
D5      13.6015   1.0000    7.6811     2       1.0000
D6      14.0712   1.4142    8.1240     2       2.0000
D7      13.5277   1.0000    7.5498     2       1.0000
D8      14.5945   1.7321    8.6603     2       3.0002
D9      6.0000    7.0711    0.0000     3       0.0000
D10     6.3246    7.0711    2.0000     3       4.0000
D11     6.1644    7.0711    1.4142     3       2.0000
D12     6.1644    7.0711    1.4142     3       2.0000
-----
TOTAL SSE = 23.000

```

Gambar kedua merupakan lanjutan dari iterasi ke-1 dari *Firefly 2*. *Centroid Firefly 2* yaitu ([2,1,2,1];[9,9,8,9];[5,4,5,4]) kemudian hasilnya seperti berikut dengan total nilai SSE 23.

```

Centroid Firefly 2:
[[2 1 2 1]
 [9 9 8 8]
 [5 4 5 4]]

-----
HASIL FIREFLY 2 - ITERASI 1
-----
Data      C1        C2        C3        Cluster    Jarak^2
-----
D1      2.0000    14.0712   6.3246     1       4.0000
D2      1.4142    14.1421   6.1644     1       2.0000
D3      1.4142    14.0000   6.1644     1       2.0000
D4      0.0000    14.0712   6.0000     1       0.0000
D5      13.5277   1.7321    7.5498     2       3.0002
D6      14.0712   1.4142    8.1240     2       2.0000
D7      13.6015   1.0000    7.6811     2       1.0000
D8      14.5258   1.0000    8.5440     2       1.0000
D9      6.3246    8.1240    2.0000     3       4.0000
D10     6.0000    8.1240    0.0000     3       0.0000
D11     6.1644    8.2462    1.4142     3       2.0000
D12     6.1644    8.0000    1.4142     3       2.0000
-----
TOTAL SSE = 23.000
-----
EVALUASI BRIGHTNESS
=====
```

Gambar ketiga terdapat Evaluasi *Brightness* dari Iterasi ke-1 yang menunjukkan bahwa *Firefly 1* itu yang terang sementara *Firefly 2* yang redup. Karena nilai SSE dari kedua *Firefly* sama, maka dipilih salah satunya, sehingga *Firefly 1*-lah yang terang. Kemudian, pada Gambar ketiga juga terdapat Iterasi ke-2 (setelah *update*) dengan *centroid* yaitu ([1,2,1,2];[8,8,8,8],[4,5,4,5]) dan hasil seperti gambar berikut serta total nilai SSE adalah 23.

```

=====
EVALUASI BRIGHTNESS
=====
Firefly 1 → TERANG
Firefly 2 → REDUP

=====
ITERASI 2 (SETELAH UPDATE)
=====

Centroid Firefly 1:
[[1 2 1 2]
 [8 8 8 8]
 [4 5 4 5]]

=====
HASIL FIREFLY 1 - ITERASI 2
=====
Data      C1        C2        C3        Cluster    Jarak^2
-----
D1      0.0000    13.0384   6.0000     1       0.0000
D2      1.4142    13.0384   6.1644     1       2.0000
D3      1.4142    13.0384   6.1644     1       2.0000
D4      2.0000    13.0384   6.3246     1       4.0000
D5      13.6015   1.0000    7.6811     2       1.0000
D6      14.0712   1.4142    8.1240     2       2.0000
D7      13.5277   1.0000    7.5498     2       1.0000
D8      14.5945   1.7321    8.6603     2       3.0002
D9      6.0000    7.0711    0.0000     3       0.0000
D10     6.3246    7.0711    2.0000     3       4.0000
D11     6.1644    7.0711    1.4142     3       2.0000
D12     6.1644    7.0711    1.4142     3       2.0000
-----
TOTAL SSE = 23.000

```

Gambar keempat merupakan *Firefly 2* dengan *centroid* yaitu $([1.483, 1.544, 1.4, 1.46] ; [8.429, 8.418, 7.937, 7.969] ; [4.479, 4.508, 4.484, 4.537])$. Hasil dari Iterasi ke-2 seperti pada gambar berikut dengan total nilai SSE adalah 13.444.

```

Centroid Firefly 2:
[[1.483 1.544 1.4  1.46 ]
 [8.429 8.418 7.937 7.969]
 [4.479 4.508 4.484 4.537]]

=====
HASIL FIREFLY 2 - ITERASI 2
=====
Data      C1        C2        C3        Cluster    Jarak^2
-----
D1      0.9449    13.4213   6.0881     1       0.8928
D2      1.0867    13.4571   6.0841     1       1.1809
D3      0.9202    13.3870   6.0897     1       0.8468
D4      1.0653    13.4229   6.0936     1       1.1349
D5      13.5898   1.2205    7.5484     2       1.4896
D6      14.0924   1.2521    8.0569     2       1.5678
D7      13.5792   0.7264    7.5452     2       0.5277
D8      14.5818   1.3400    8.5442     2       1.7956
D9      6.1295    7.4575    0.9592     3       0.9201
D10     6.1492    7.4603    1.0412     3       1.0841
D11     6.1530    7.5217    0.9839     3       0.9681
D12     6.1258    7.3955    1.0179     3       1.0361
-----
TOTAL SSE = 13.444
=====

=====
KESIMPULAN
=====
```

Gambar kelima adalah kesimpulan dari program yang telah dijalankan.

```

=====
KESIMPULAN
=====

Firefly terbaik adalah Firefly 2
karena memiliki SSE paling kecil (13.444).

Firefly Algorithm mengoptimasi centroid
dengan memilih brightness tertinggi (SSE minimum).

```

Terdapat perbedaan yang terjadi pada nilai SSE antara Excel dan Python disebabkan oleh nilai acak (*random*) yang dihasilkan oleh fungsi RAND() di Excel dan np.random.rand() di Python. Hal ini dikarenakan algoritma *Firefly* bersifat stokastik (acak atau tidak bisa diprediksi) sehingga hasil numeriknya tidak sama walaupun parameter dan rumusnya identik.

KESIMPULAN LAPORAN

Algoritma *Firefly* digunakan untuk mengoptimasi proses klastering dari K-Means dengan memperbaiki posisi *centroid* secara iteratif sehingga nilai SSE dapat diminimalkan. Berdasarkan hasil perhitungan yang diperoleh, penerapan *Firefly* mampu meningkatkan kualitas klaster dibandingkan K-Means dengan inisialisasi *centroid* secara konvensional, karena proses pencarian solusi dilakukan melalui mekanisme eksplorasi dan eksloitasi yang lebih optimal. Perbedaan nilai hasil perhitungan antara implementasi Excel dan Python terjadi akibat adanya komponen acak (*randomness*) dari algoritma *Firefly* yang menyebabkan variasi nilai pada setiap eksekusi. Walau demikian, perbedaan tersebut tidak memberikan dampak yang signifikan pada pola klaster yang terbentuk maupun pada kesimpulan akhirnya.