

# Laporan Praktikum Kecerdasan Buatan Logika Fuzzy

Dosen Pengampu Bu Entin



Wahyu Ikbil Maulana

3323600056

D4 SDT B

Politeknik Elektronika Negeri Surabaya

# Percobaan 1

```
In [ ]: import numpy as np
import math
```

```
In [ ]: class Berat:
    """
    Variabel titik-titik range pada model fuzzy,
    agar mudah untuk mengganti nilai ketika ada perubahan
    """
    titik1 = 0
    titik2 = 40
    titik3 = 45
    titik4 = 50
    titik5 = 55
    titik6 = 60
    titik7 = 65
    titik8 = 80
    titik9 = 85

    def __init__(self, berat):
        self.berat = berat

    def sangatkurus(self):
        if(self.berat >= self.titik1 and self.berat <= self.titik2):
            return 1
        elif (self.berat > self.titik2 and self.berat < self.titik3):
            return (float)(self.titik3 - self.berat) / (self.titik3 - self.titik2)
        else:
            return 0

    def kurus(self):
        if (self.berat > self.titik2 and self.berat < self.titik3):
            return (float)(self.berat - self.titik2) / (self.titik3 - self.titik2)
        elif (self.berat >= self.titik3 and self.berat <= self.titik4):
            return 1
        elif (self.berat > self.titik4 and self.berat < self.titik5):
            return (float)(self.titik5 - self.berat) / (self.titik5 - self.titik4)
        else:
            return 0

    def biasa(self):
        if (self.berat > self.titik4 and self.berat < self.titik5):
            return (float)(self.berat - self.titik4) / (self.titik5 - self.titik4)
        elif (self.berat >= self.titik5 and self.berat <= self.titik6):
            return 1
        elif (self.berat > self.titik6 and self.berat < self.titik7):
            return (float)(self.titik7 - self.berat) / (self.titik7 - self.titik6)
        else:
            return 0


    def f_berat(self):
        if (self.berat > self.titik6 and self.berat < self.titik7):
            return (float)(self.berat - self.titik6) / (self.titik7 - self.titik6)
        elif (self.berat >= self.titik7 and self.berat <= self.titik8):
            return 1
```

```

        elif (self.berat > self.titik8 and self.berat < self.titik9):
            return (float)(self.titik9 - self.berat) / (self.titik9 - self.titik8)
        else:
            return 0

    def sangatberat(self):
        if (self.berat > self.titik8 and self.berat < self.titik9):
            return (float)(self.berat - self.titik8) / (self.titik9 - self.titik8)
        elif (self.berat >= self.titik9):
            return 1
        else:
            return 0

```

 **Analisis :** Kelas "Berat" yang di definisikan berguna untuk mengimplementasikan model fuzzy yang berguna untuk mengklasifikasikan kategori berat badan nanti. Di dalam kelas "Berat" ada beberapa Fungsi keanggotaan, yaitu:

- `sangatkurus()`: berguna untuk menghitung keanggotaan berat badan dalam kategori "sangat kurus".
- `kurus()`: berguna untuk menghitung keanggotaan berat badan dalam kategori "kurus".
- `biasa()`: berguna untuk menghitung keanggotaan berat badan dalam kategori "biasa".
- `f_berat()`: berguna untuk menghitung keanggotaan berat badan dalam kategori "berat".
- `sangatberat()`: berguna untuk menghitung keanggotaan berat badan dalam kategori "sangat berat".

Setiap fungsi keanggotaan ini memerlukan nilai berat badan sebagai input dan mengembalikan nilai keanggotaan dalam rentang [0, 1]. Jika nilai berat badan berada di luar rentang yang ditentukan, nilai keanggotaan yang dihasilkan adalah 0 atau 1 tergantung pada kategori berat badan yang paling ekstrim.

```

In [ ]: class Berat:
        ...
        Variabel titik-titik range pada model fuzzy,
        agar mudah untuk mengganti nilai ketika ada perubahan
        ...
        titik1 = 0
        titik2 = 40
        titik3 = 45
        titik4 = 50

```

```

titik5 = 55
titik6 = 60
titik7 = 65
titik8 = 80
titik9 = 85

def __init__(self, berat):
    self.berat = berat

def sangatkurus(self):
    if(self.berat >= self.titik1 and self.berat <= self.titik2):
        return 1
    elif (self.berat > self.titik2 and self.berat < self.titik3):
        return (float)(self.titik3 - self.berat) / (self.titik3 - self.titik2)
    else:
        return 0


def kurus(self):
    if (self.berat > self.titik2 and self.berat < self.titik3):
        return (float)(self.berat - self.titik2) / (self.titik3 - self.titik2)
    elif (self.berat >= self.titik3 and self.berat <= self.titik4):
        return 1
    elif (self.berat > self.titik4 and self.berat < self.titik5):
        return (float)(self.titik5 - self.berat) / (self.titik5 - self.titik4)
    else:
        return 0

def biasa(self):
    if (self.berat > self.titik4 and self.berat < self.titik5):
        return (float)(self.berat - self.titik4) / (self.titik5 - self.titik4)
    elif (self.berat >= self.titik5 and self.berat <= self.titik6):
        return 1
    elif (self.berat > self.titik6 and self.berat < self.titik7):
        return (float)(self.titik7 - self.berat) / (self.titik7 - self.titik6)
    else:
        return 0

def f_berat(self):
    if (self.berat > self.titik6 and self.berat < self.titik7):
        return (float)(self.berat - self.titik6) / (self.titik7 - self.titik6)
    elif (self.berat >= self.titik7 and self.berat <= self.titik8):
        return 1
    elif (self.berat > self.titik8 and self.berat < self.titik9):
        return (float)(self.titik9 - self.berat) / (self.titik9 - self.titik8)
    else:
        return 0

def sangatberat(self):
    if (self.berat > self.titik8 and self.berat < self.titik9):
        return (float)(self.berat - self.titik8) / (self.titik9 - self.titik8)
    elif (self.berat >= self.titik9):
        return 1
    else:
        return 0

```

 **Analisis :** Sedangkan di kelas "Tinggi" ini berguna sebagai implementasi dari model fuzzy yang nantinya untuk mengklasifikasikan kategori tinggi badan. seperti

pada kelas "Berat", pada kelas "Tinggi" ini juga memiliki beberapa fungsi yang mewakili tingkatan dari tinggi badan, yaitu:

- `sangatpendek()`: Berguna untuk menghitung keanggotaan tinggi badan dalam kategori "sangat pendek".
- `pendek()`: Berguna untuk menghitung keanggotaan tinggi badan dalam kategori "pendek".
- `sedang()`: Berguna untuk menghitung keanggotaan tinggi badan dalam kategori "sedang".
- `f_tinggi()`: Berguna untuk menghitung keanggotaan tinggi badan dalam kategori "tinggi".
- `sangattinggi()`: Berguna untuk menghitung keanggotaan tinggi badan dalam kategori "sangat tinggi".

Nilai tinggi badan juga diperlukan untuk setiap fungsi keanggotaan ini, yang mengembalikan nilai keanggotaan dalam rentang [0, 1]. Jika nilai tinggi badan berada di luar rentang yang ditentukan, nilai keanggotaan yang dihasilkan akan menjadi 0 atau 1, tergantung pada kategori tinggi badan yang paling ekstrim.

```
In [ ]: status = ""
u_Status = np.empty(25, dtype=object)
z_Status = ["Sangat Sehat", "Sehat", "Agak Sehat", "Tidak Sehat", "Tidak Sehat",
            "Sehat", "Sangat Sehat", "Sehat", "Agak Sehat", "Tidak Sehat",
            "Agak Sehat", "Sangat Sehat", "Sangat Sehat", "Agak Sehat", "Tidak S
            "Tidak Sehat", "Sehat", "Sangat Sehat", "Sehat", "Tidak Sehat",
            "Tidak Sehat", "Agak Sehat", "Sangat Sehat", "Sehat", "Agak Sehat"]

# RULE
def hitung_u():
    u_Status[0] = min(tinggi_test.sangatpendek(), berat_test.sangatkurus()) # S
    u_Status[1] = min(tinggi_test.sangatpendek(), berat_test.kurus()) # Sehat
    u_Status[2] = min(tinggi_test.sangatpendek(), berat_test.biasa()) # Agak Se
    u_Status[3] = min(tinggi_test.sangatpendek(), berat_test.f_berat()) # Tidak S
    u_Status[4] = min(tinggi_test.sangatpendek(), berat_test.sangatberat()) # T
    u_Status[5] = min(tinggi_test.pendek(), berat_test.sangatkurus()) # Sehat
    u_Status[6] = min(tinggi_test.pendek(), berat_test.kurus()) # Sangat Sehat
    u_Status[7] = min(tinggi_test.pendek(), berat_test.biasa()) # Sehat
    u_Status[8] = min(tinggi_test.pendek(), berat_test.f_berat()) # Agak Sehat
    u_Status[9] = min(tinggi_test.pendek(), berat_test.sangatberat()) # Tidak S
    u_Status[10] = min(tinggi_test.sedang(), berat_test.sangatkurus()) # Agak S
    u_Status[11] = min(tinggi_test.sedang(), berat_test.kurus()) # Sangat Sehat
    u_Status[12] = min(tinggi_test.sedang(), berat_test.biasa()) # Sangat Sehat
    u_Status[13] = min(tinggi_test.sedang(), berat_test.f_berat()) # Agak Sehat
    u_Status[14] = min(tinggi_test.sedang(), berat_test.sangatberat()) # Tidak
    u_Status[15] = min(tinggi_test.f_tinggi(), berat_test.sangatkurus()) # Tida
    u_Status[16] = min(tinggi_test.f_tinggi(), berat_test.kurus()) # Sehat
```

```

u_Status[17] = min(tinggi_test.f_tinggi(), berat_test.biasa()) # Sangat Sehat
u_Status[18] = min(tinggi_test.f_tinggi(), berat_test.f_berat()) # Sehat
u_Status[19] = min(tinggi_test.f_tinggi(), berat_test.sangatberat()) # Tidak Sehat
u_Status[20] = min(tinggi_test.sangattinggi(), berat_test.sangatkurus()) # Sangat Kurus
u_Status[21] = min(tinggi_test.sangattinggi(), berat_test.kurus()) # Agak Kurus
u_Status[22] = min(tinggi_test.sangattinggi(), berat_test.biasa()) # Sangat Sehat
u_Status[23] = min(tinggi_test.sangattinggi(), berat_test.f_berat()) # Sangat Sehat
u_Status[24] = min(tinggi_test.sangattinggi(), berat_test.sangatberat()) # Sangat Sehat

# Max Method
def hitung_z():
    global status
    max_value = 0
    for i in range(len(u_Status)):
        if max_value < u_Status[i]:
            max_value = u_Status[i]
            status = z_Status[i]
    return max_value


b = int(input("Masukkan Berat Badan: "))
t = int(input("Masukkan Tinggi Badan: "))
# Assuming Berat and Tinggi classes are defined elsewhere
berat_test = Berat(b)
tinggi_test = Tinggi(t)

hitung_u()

for index, value in enumerate(u_Status):
    print(f"u_ke- {index}, Value: {value}")

bobot = hitung_z()
print("Bobot: ", bobot, "dengan status kesehatan: ", status)

```

 **Analisis :** Fungsi dari kode tersebut merupakan main code untuk menjalankan kode kode sebelumnya. Kode ini berguna untuk mengklasifikasikan status kesehatan berdasarkan berat badan dan tinggi badan.

- Pertama, kelas "Tinggi" dan "Berat" mendefinisikan fungsi keanggotaan

untuk variabel input, yang merupakan tinggi dan berat badan. Kemudian, nilai input dari pengguna digunakan untuk membuat objek dari kedua kelas tersebut.

- Fungsi "hitung\_u()" untuk menghitung derajat keanggotaan setiap

aturan fuzzy menggunakan metode min sebagai implikasi. Derajat keanggotaan untuk setiap aturan disimpan dalam array u\_Status.

- Selanjutnya, fungsi "hitung\_z()" digunakan untuk menentukan nilai

keanggotaan tertinggi (max) dari semua aturan yang telah dihitung sebelumnya. Status kesehatan ditentukan berdasarkan nilai keanggotaan tertinggi tersebut.

Selanjutnya, semua nilai keanggotaan aturan, nilai bobot, dan nilai status kesehatan dicetak untuk ditunjukkan kepada user.


```
In [ ]: import numpy as np
import math

class Permintaan:
    """
    variabel titik-titik range pada model fuzzy,
    agar mudah untuk mengganti nilai ketika ada perubahan
    """
    titik1 = 0
    titik2 = 1000
    titik3 = 5000

    def __init__(self, permintaan):
        self.permintaan = permintaan

    def turun(self):
        if (self.permintaan >= self.titik1 and self.permintaan <= self.titik2):
            return 1.0
        elif (self.permintaan > self.titik2 and self.permintaan < self.titik3):
            return float(self.titik3 - self.permintaan) / (self.titik3 - self.titik2)
        else:
            return 0.0

    def naik(self):
        if (self.permintaan > self.titik2 and self.permintaan < self.titik3):
            return float(self.permintaan - self.titik2) / (self.titik3 - self.titik2)
        elif (self.permintaan >= self.titik3):
            return 1.0
        else:
            return 0.0
```

 **Alasan :** Pada kelas "Permintaan" ini berguna untuk mengklasifikasikan tingkat permintaan dengan mengimplementasikan model fuzzy. Di dalam kelas ini ada dua fungsi, yang setiap fungsinya ini memiliki titik-titik range yang digunakan untuk menghitung nilai keanggotaannya berdasarkan nilai permintaan yang diberikan. Fungsi tersebut adalah:

- turun(): Berguna untuk menghitung keanggotaan tingkat permintaan

dalam kategori "turun".

- naik(): Berguna untuk menghitung keanggotaan tingkat permintaan

dalam kategori "naik".

Nanti Nilai permintaan juga diperlukan untuk setiap fungsi tersebut, yang mengembalikan nilai fungsi dalam rentang [0, 1]. Jika nilai permintaan berada di luar rentang yang ditentukan, nilai Fungsi yang dihasilkan akan menjadi 0 atau 1, tergantung pada kategori permintaan tersebut.

```
In [ ]: class Persediaan:
```

```

variabel titik-titik range pada model fuzzy,
agar mudah untuk mengganti nilai ketika ada perubahan
'''


titik1 = 0
titik2 = 100
titik3 = 600

def __init__(self, persediaan):
    self.persediaan = persediaan

def sedikit(self):
    if (self.persediaan >= self.titik1 and self.persediaan <= self.titik2):
        return 1.0
    elif (self.persediaan > self.titik2 and self.persediaan < self.titik3):
        return float(self.titik3 - self.persediaan) / (self.titik3 - self.titik2)
    else:
        return 0.0

def banyak(self):
    if (self.persediaan > self.titik2 and self.persediaan < self.titik3):
        return float(self.persediaan - self.titik2) / (self.titik3 - self.titik2)
    elif (self.persediaan >= self.titik3):
        return 1.0
    else:
        return 0.0

```

 **Alasan :** Pada kelas "Persediaan" ini kita akan mendefinisikan pengklasifikasian Tingkat persediaan barang. Disini ada 2 fungsi yang merupakan anggota dari kelas "Persediaan". Fungsi dari kedua anggota ini adalah untuk menghitung nilai keanggotaannya berdasarkan nilai persediaan yang diberikan. Kedua fungsi tersebut adalah:

- sedikit(): untuk menghitung keanggotaan tingkat persediaan dalam kategori "sedikit".
- banyak(): untuk menghitung keanggotaan tingkat persediaan dalam kategori "banyak".

Kelas "Persediaan" ini juga membutuhkan nilai persediaan sebagai input dan mengembalikan nilai keanggotaan dalam rentang [0, 1]. Jika nilai persediaan berada di luar rentang yang ditentukan, nilai Fungsi yang dihasilkan akan menjadi 0 atau 1, tergantung pada kategori persediaan tersebut.

```

In [ ]: class Produksi:
        titik1 = 0
        titik2 = 2000
        titik3 = 7000

        def __init__(self, produksi):
            self.persediaan = produksi

```




```

@staticmethod
def berkurang(*args):
    if len(args) == 0:
        if Produksi.produksi >= Produksi.titik1 and Produksi.produksi <= Pro
            return 1.0
        elif Produksi.produksi > Produksi.titik2 and Produksi.produksi < Pro
            return float(Produksi.titik3 - Produksi.produksi) / (Produksi.ti
        else:
            return 0.0
    else:
        return float(Produksi.titik3 - (args[0] * (Produksi.titik3 - Produks

@staticmethod
def bertambah(*args):
    if len(args) == 0:
        if Produksi.produksi > Produksi.titik2 and Produksi.produksi < Produ
            return float(Produksi.produksi - Produksi.titik2) / (Produksi.ti
        elif Produksi.produksi >= Produksi.titik3:
            return 1.0
        else:
            return 0.0
    else:
        return float(Produksi.titik2 + (args[0] * (Produksi.titik3 - Produks

```

 **Analisis :** Pada kelas "Produksi" ini kegunaannya sama seperti kelas kelas yang telah kita analisis sebelumnya. Yaitu untuk mengklasifikasikan Tingkat produksi. Kelas ini memiliki dua fungsi yang sebagai anggota dari kelas "Produksi" Fungsi Fungsi ini juga memiliki titik-titik range yang digunakan untuk menghitung nilai keanggotaannya berdasarkan nilai produksi yang diberikan.

- berkurang(): Berguna untuk menghitung keanggotaan tingkat produksi

dalam kategori "berkurang".

- bertambah(): Berguna untuk menghitung keanggotaan tingkat produksi

dalam kategori "bertambah".berkurang():

Kelas "Produksi" ini juga membutuhkan nilai dari Produksi sebagai input dan mengembalikan nilai keanggotaan dalam rentang [0, 1]. Jika nilai produksi berada di luar rentang yang ditentukan, nilai Fungsi yang dihasilkan akan menjadi 0 atau 1, tergantung pada kategori produksi tersebut.

```

In [ ]: import numpy as np
import math

u_Produksi = np.empty(4, dtype=float)
zt_Produksi = np.empty(4, dtype=float)
zs_Produksi = np.empty(4, dtype=float)

def hitung_u():
    u_Produksi[0] = min(minta_test.turun(), sedia_test.banyak())
    u_Produksi[1] = min(minta_test.turun(), sedia_test.sedikit())

```

```

u_Produksi[2] = min(minta_test.naik(), sedia_test.banyak())
u_Produksi[3] = min(minta_test.naik(), sedia_test.sedikit())

# RULE 1 (Tsukamoto)
def hitung_zt():
    zt_Produksi[0] = Produksi.berkurang(u_Produksi[0])
    zt_Produksi[1] = Produksi.berkurang(u_Produksi[1])
    zt_Produksi[2] = Produksi.bertambah(u_Produksi[2])
    zt_Produksi[3] = Produksi.bertambah(u_Produksi[3])

# RULE 2 (Sugeno)
def hitung_zs():
    zs_Produksi[0] = minta_test.permintaan - sedia_test.persediaan
    zs_Produksi[1] = minta_test.permintaan
    zs_Produksi[2] = minta_test.permintaan
    zs_Produksi[3] = float(1.25 * minta_test.permintaan - sedia_test.persediaan)

def bobot():
    atas_zt = 0
    bawah_zt = 0
    atas_zs = 0
    bawah_zs = 0
    for i in range(len(u_Produksi)):
        atas_zt += (u_Produksi[i] * zt_Produksi[i])
        bawah_zt += u_Produksi[i]
        atas_zs += (u_Produksi[i] * zs_Produksi[i])
        bawah_zs += u_Produksi[i]
    # Handle division by zero error
    if bawah_zt == 0:
        t_jml_prod = 0
    else:
        t_jml_prod = float(atas_zt / bawah_zt)
    if bawah_zs == 0:
        s_jml_prod = 0
    else:
        s_jml_prod = float(atas_zs / bawah_zs)
    return t_jml_prod, s_jml_prod

class Permintaan:
    def __init__(self, permintaan):
        self.permintaan = permintaan

    def turun(self):
        pass # You need to implement this method

    def naik(self):
        pass # You need to implement this method

class Persediaan:
    def __init__(self, persediaan):
        self.persediaan = persediaan

    def banyak(self):
        pass # You need to implement this method

    def sedikit(self):
        pass # You need to implement this method

# Assuming Permintaan and Persediaan classes have the required methods

```

```


minta = int(input("Masukkan Permintaan: "))
sedia = int(input("Masukkan Persediaan: "))

minta_test = Permintaan(minta)
sedia_test = Persediaan(sedia)

hitung_u()
hitung_zt()
hitung_zs()
bobot_p = bobot()

print('Produksi (Tsukamoto): ', math.floor(bobot_p[0]))
print('Produksi (Sugeno): ', math.floor(bobot_p[1]))

```

 **Analisis :** Main code diatas berguna untuk menerapkan metode Tsukamoto dan Sugeno untuk menentukan produksi berdasarkan tingkat permintaan dan persediaan barang. Metode Tsukamoto:

1. Fungsi `hitung_u()` berfungsi untuk menghitung tingkat keanggotaan dari setiap aturan fuzzy.
2. Fungsi `hitung_zt()` menghitung bobot (kontribusi) dari setiap aturan menggunakan model Tsukamoto.
3. Bobot produksi (jumlah produksi) dihitung dengan menggunakan fungsi `bobot()`, yang menghasilkan rata-rata tertimbang dari bobot produksi sesuai aturan fuzzy yang telah dihitung sebelumnya.

Metode Sugeno:

1. Fungsi `hitung_zs()` menghitung bobot (kontribusi) dari setiap aturan menggunakan model Sugeno. Dalam kasus ini, bobotnya langsung dihitung berdasarkan persamaan yang telah ditentukan.
2. Bobot produksi (jumlah produksi) juga dihitung menggunakan fungsi `bobot()`, dengan menghasilkan rata-rata tertimbang dari bobot produksi sesuai aturan fuzzy yang telah dihitung sebelumnya. Setelahnya, setelah user memasukkan nilai permintaan dan persediaan, dilakukan perhitungan derajat keanggotaan, bobot produksi menggunakan kedua metode tersebut, dan hasilnya ditampilkan kepada user.

In [ ]: `import numpy as np`

```

class GPA:
    titik1 = 0
    titik2 = 2.2
    titik3 = 3.0
    titik4 = 3.8
    titik5 = 4.0

    def __init__(self, gpa):
        self.gpa = gpa

    def high(self):
        if self.gpa < self.titik4 and self.gpa > self.titik3:

```

```

        return (self.gpa - self.titik3) / (self.titik4 - self.titik3)
    elif self.gpa > self.titik3 and self.gpa < self.titik4:
        return (self.titik5 - self.gpa) / (self.titik5 - self.titik4)
    elif self.gpa >= self.titik4:
        return 1
    else:
        return 0

def medium(self):
    if self.gpa < self.titik3 and self.gpa > self.titik2:
        return (self.gpa - self.titik2) / (self.titik3 - self.titik2)
    elif self.gpa > self.titik2 and self.gpa < self.titik3:
        return (self.titik4 - self.gpa) / (self.titik4 - self.titik3)
    elif self.gpa >= self.titik3:
        return 1
    else:
        return 0

def low(self):
    if self.gpa > self.titik1 and self.gpa < self.titik2:
        return (self.titik3 - self.gpa) / (self.titik3 - self.titik2)
    elif self.gpa >= self.titik2 and self.gpa < self.titik3:
        return 1
    else:
        return 0

class GRE:
    titik1 = 0
    titik2 = 800
    titik3 = 1200
    titik4 = 1800

    def __init__(self, gre):
        self.gre = gre

    def high(self):
        if self.gre < self.titik4 and self.gre > self.titik3:
            return (self.gre - self.titik3) / (self.titik4 - self.titik3)
        elif self.gre > self.titik3 and self.gre < self.titik4:
            return (self.gre - self.titik3) / (self.titik4 - self.titik3)
        elif self.gre >= self.titik4:
            return 1
        else:
            return 0

    def medium(self):
        if self.gre < self.titik3 and self.gre > self.titik2:
            return (self.gre - self.titik2) / (self.titik3 - self.titik2)
        elif self.gre > self.titik2 and self.gre < self.titik3:
            return (self.titik4 - self.gre) / (self.titik4 - self.titik3)
        elif self.gre >= self.titik3:
            return 1
        else:
            return 0

    def low(self):
        if self.gre > self.titik1 and self.gre < self.titik2:
            return 1
        elif self.gre >= self.titik2 and self.gre < self.titik3:
            return (self.titik3 - self.gre) / (self.titik3 - self.titik2)

```

```

        else:
            return 0

class Centroid:
    poor_var = 65
    fair_var = 70
    good_var = 80
    very_good_var = 90
    excellent_var = 95

    def __init__(self, crisp):
        self.crisp = crisp

    def poor(self):
        if self.crisp > self.poor_var and self.crisp < self.fair_var:
            return (self.crisp - self.poor_var) / (self.fair_var - self.poor_var)
        elif self.crisp <= self.poor_var:
            return 1
        else:
            return 0

    def fair(self):
        if self.crisp < self.fair_var and self.crisp > self.poor_var:
            return (self.crisp - self.fair_var) / (self.good_var

```

Cell In [2], line 98

```

    return (self.crisp - self.fair_var) / (self.good_var

```

**SyntaxError:** incomplete input



**Analisis :** Kode ini mengimplementasikan dua metode untuk menentukan status evaluasi berdasarkan nilai GPA dan GRE: Metode Max dan Metode Centroid. Metode Max:

1. Fungsi `hitung_u()` : untuk menghitung derajat keanggotaan untuk setiap

status evaluasi berdasarkan aturan fuzzy yang telah ditentukan. 2. Fungsi

`hitung_z()` : untuk menghitung bobot (kontribusi) dari setiap aturan menggunakan metode Max. 3. Output: menampilkan bobot evaluasi dan status evaluasi berdasarkan metode Max. Metode Centroid:

1. Fungsi `hitung_crisp_centroid()` : untuk menghitung crisp centroid

berdasarkan derajat keanggotaan yang telah dihitung sebelumnya. 2. Fungsi

`hitung_centroid_status()` : untuk menghitung derajat keanggotaan untuk setiap status evaluasi berdasarkan crisp centroid. 3. Fungsi `hitung_centroid()` : untuk menentukan status evaluasi berdasarkan derajat keanggotaan yang telah dihitung menggunakan metode Centroid. 4. Output: menampilkan crisp centroid, derajat keanggotaan status evaluasi, dan status evaluasi berdasarkan metode Centroid. lalu, setelah menginputkan nilai GPA dan GRE dari user, dilakukan perhitungan derajat keanggotaan, bobot evaluasi, crisp centroid, derajat

keanggotaan status evaluasi, dan status evaluasi berdasarkan kedua metode tersebut. Terakhir akan menampilkan output kepada user

In [ ]: