CII2M3 INTRODUCTION OF ARTIFICIAL INTELLIGENCE

EVEN SEMESTER SESSION 2021/2022

ASSIGNMENT 2 - REASONING

Group No : 10

Section: IF-44-INT

Lecturer Name: Edward Ferdian

Group Member:

NAME	STUDENT ID
NUR SABRINA SYAZA BINTI ZAHAR HISHAM	1301213666
NURUL IMAN BINTI NORDIN	1301213669
ZIDAN RIZKY WIJAYA	1301200226

TABLE OF CONTENT

Youtube link: https://youtu.be/VdgJk_8mIyY

Readme link:

https://github.com/Sabrinasyaza/Programming-Assignment-2/blob/main/README.md

QUESTION:

Given bengkel.xlsx file which contains data of 100 auto mechanics in Bandung with 2 attributes:

Service quality (real number 1-100; with 100 means best quality) and
Price (real number 1-10, with 10 means most expensive).

Develop a Fuzzy Logic-based system to choose 10 best auto mechanics in Bandung.

The system reads the bengkel.xlsx as input file and outputs a file ranking.xlsx consists of 10 best auto-mechanics using their ID and score (defuzzification output).

1. PROBLEM DESCRIPTION

Fuzzy logic is a variable processing technique that allows multiple truth values to be processed by the same variable. Fuzzy logic aims to solve problems by using an open, inaccurate spectrum of facts and heuristics to provide a range of accurate conclusions and also making the best possible decision given the input.

We need to analyze and design a Fuzzy Logic-based system in order to choose 10 best auto mechanics in Bandung based on the input which are Service and Price in the bengkel.xlsx file.

2. AMOUNT AND LINGUISTIC NAMES FOR EACH INPUT ATTRIBUTE

The inputs for the auto-mechanics based on the bengkel.xlxs are Service and Price, while the output is the 10 best auto-mechanics score.

For the step 1, in order to design the linguistic names for each input attribute, we have specified each into three linguistics such as :

- Service : Bad, Average, Best
- Price : Cheap, Standard, Expensive

For the output, we designed also three linguistics which are :

- Score : Not Recommended (NotRec), Recommended (Rec), Very Recommended (VeryRec)

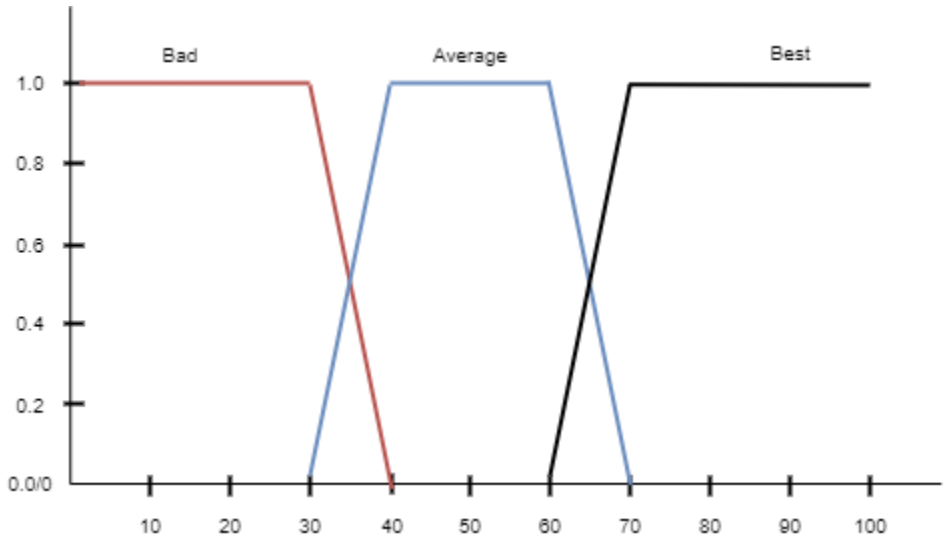


Figure 2.0: Service

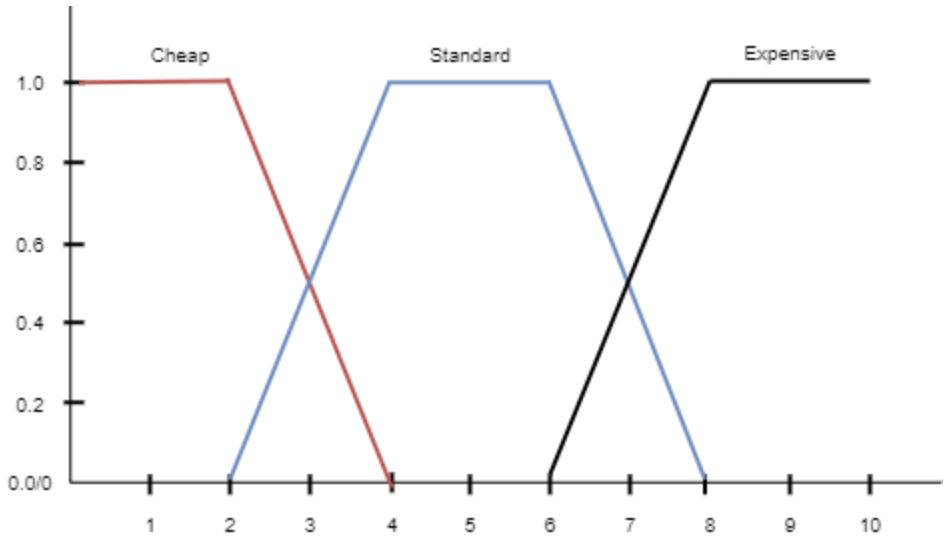


Figure 2.1: Price

```
y = []

for x in range(100):
    service_score = [0,0,0,0]
    price_score = [0,0,0,0]
    bad = average = best = cheap = standard = expensive = 0
```

3. SHAPE AND LIMIT OF THE INPUT MEMBERSHIP FUNCTION

For step 2, we have designed the shape and limit of the input membership function. The range of Service is [1,100] and for the Price is [1, 10].

Service

The membership function for Service (Bad), its range is between 1 to 30. From there, we determined that Service <= 30 is Bad and Service > 40 is Not Bad. Therefore, the range of fuzzy areas between Bad and Not Bad is [30, 40].

For the membership function for Service (Average), its range is between 40 to 60. Service <= 30 or Service > 70 is considered as Not Average.

The membership function for Service (Best), its range is between 60 to 100. Service > 70 is definitely Best and Service <= 60 is definitely Not Best. So, the range of its fuzzy area between Best and Not Best is [60, 70].

Price

The range of membership function for Price (Cheap) is between 1 to 2. From there, we determined that Price <= 2 is Cheap and Price > 4 is Not Cheap. Therefore, the range [2, 4] is the fuzzy area between Cheap and Not Cheap.

The membership function for Price (Standard), its range is between 4 to 6. Price <= 2 or Price > 8 is considered as Not Average.

The range of membership function for Price (Expensive) is between 6 to 10. Price > 8 is definitely Expensive and Price <= 6 is definitely Not Expensive. The range of its fuzzy area between Expensive and Not Expensive is [6, 8].

- Very Recommended (VeryRec)
- Recommended (Rec)
- Not Recommended (NotRec)

Fuzzy Rule

Price / Service	Cheap	Standard	Expensive
Bad	Not Recommended (NotRec)	Not Recommended (NotRec)	Not Recommended (NotRec)
Average	Recommended (Rec)	Recommended (Rec)	Not Recommended (NotRec)
Best	Very Recommended (VeryRec)	Very Recommended (VeryRec)	Recommended (Rec)

```
#service

if service[x] <= 30:
    bad = 1
    service_score[0] = bad

elif service[x] > 30 and service[x] < 40:
    bad = (40 - service[x]) / 10
    average = (service[x] - 30) / 10
    service_score[0] = bad
    service_score[1] = average

elif service[x] >= 40 and service[x] <= 60:
    average = 1
    service_score[1] = average

elif service[x] > 60 and service[x] < 70:
    average = (70 - service[x]) / 10
    best = (service[x] - 60)
    service_score[1] = average
    service_score[2] = best

elif service[x] >= 70 and service[x] <= 100:
    best = 1
    service_score[2] = best
```

```
#price

  if price[x] <= 2:
      cheap = 1
      price_score[0] = cheap

  elif price[x] > 2 and price[x] < 4:
      cheap = ((4 - price[x]) / 2)
      standard = ((price[x] - 2) / 2)
      price_score[0] = cheap
      price_score[1] = standard

  elif price[x] >= 4 and price[x] <= 6:
      standard = 1
      price_score[1] = standard

  elif price[x] > 6 and price[x] < 8:
      standard = ((8 - price[x]) / 2)
      expensive = ((price[x] - 6) / 2)
      price_score[1] = standard
      price_score[2] = expensive

  elif price[x] >= 8 and price[x] <= 10:
      expensive = 1
      price_score[2] = expensive
```

4. INFERENCE RULES

Inference rules are syntactical transform rules that may be used to infer a conclusion from a premise in order to construct an argument. If a set of rules is full, it may be used to infer any valid conclusion while never inferring an invalid conclusion if it is sound. Many of the rules in the following list are redundant and may be demonstrated with the other rules, therefore a solid and comprehensive set of rules does not need to include them all.

Using Clipping technique, the conjunction rule will get the minimum value of fuzzy input as the fuzzy output.

- Service score (Bad) = 0
- Service score (Average) = 1
- Service score (Best) = 2
- Price score (Cheap) = 0
- Price score (Standard) = 1
- Price Score (Expensive) = 2

```
#Not recommended

NotRec = []
if service_score[0] == bad and price_score[0] == cheap:
    NotRec.append(min(service_score[0], price_score[0]))
if service_score[0] == bad and price_score[1] == standard:
    NotRec.append(min(service_score[0], price_score[1]))
if service_score[0] == bad and price_score[2] == expensive:
    NotRec.append(min(service_score[0], price_score[2]))
if service_score[1] == average and price_score[2] == expensive:
    NotRec.append(min(service_score[1], price_score[2]))
NotRec_score = max(NotRec)

#Recommended

Rec = []
if service_score[1] == average and price_score[0] == cheap:
    Rec.append(min(service_score[1], price_score[0]))
if service_score[1] == average and price_score[1] == standard:
    Rec.append(min(service_score[1], price_score[1]))
if service_score[2] == best and price_score[2] == expensive:
    Rec.append(min(service_score[2], price_score[2]))
Rec_score = max(Rec)
```

```
#Very recommended

VeryRec = []
if service_score[2] == best and price_score[0] == cheap:
    VeryRec.append(min(service_score[2], price_score[0]))
if service_score[2] == best and price_score[1] == standard:
    VeryRec.append(min(service_score[2], price_score[1]))
VeryRec_score = max(VeryRec)
```

5. DEFUZZIFICATION METHOD

The process of transforming a fuzzified output into a single crisp value with regard to a fuzzy set is known as defuzzification. In FLC (Fuzzy Logic Controller), the defuzzified value reflects the action to be done in regulating the process. There are two common methods of defuzzification which are Center of Gravity (Mamdani-style) and Constant Defuzzification (Takagi-Sugeno-style).

In our coding, we have used Takagi-Sugeno-Style (Constant Defuzzification) as our method to choose 10 best auto mechanics in Bandung. We have chose a constant value to represent each output linguistic which are :

- Not Recommended (NotRec) : 25
- Recommended (Rec) : 50
- Very Recommended (VeryRec) : 100

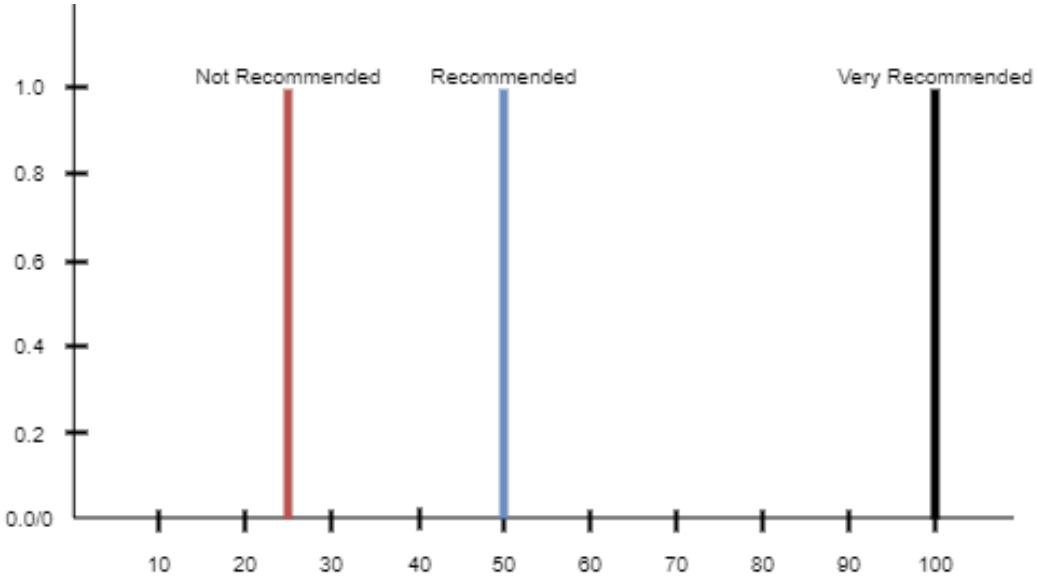
```
divider = NotRec_score + Rec_score + VeryRec_score
if divider == 0:
    z = 0
else:
    z = ((NotRec_score*25) + (Rec_score*50) + (VeryRec_score*100))/divider
print(id_code[x],z)
y.append([id_code[x],z])
```

6. SHAPE AND LIMIT OF THE OUTPUT MEMBERSHIP FUNCTION

The shape and limit of the output membership function is the maximum value from each linguistic.

The constant value to represent each output linguistic which are :

- Not Recommended (NotRec) : 25
- Recommended (Rec) : 50
- Very Recommended (VeryRec) : 100



THE CODE

In [6]:
```
import pandas as pd
bengkel = pd.read_excel("bengkel.xlsx")
bengkel
```

Out[6]:

	ID	Service	Price
0	1	58	7
1	2	54	1
2	3	98	2
3	4	52	4
4	5	11	4
...	...	...	...
95	96	30	1
96	97	25	3
97	98	27	10
98	99	8	6
99	100	11	8

100 rows × 3 columns

In [7]:
```
#Fuzzification

id_code = bengkel['ID']
service = bengkel['Service']
price = bengkel['Price']
```

In [10]:

```
y = []

for x in range(100):
    service_score = [0,0,0,0]
    price_score = [0,0,0,0]
    bad = average = best = cheap = standard = expensive = 0


    #service

    if service[x] <= 30:
        bad = 1
        service_score[0] = bad

    elif service[x] > 30 and service[x] < 40:
        bad = (40 - service[x]) / 10
        average = (service[x] - 30) / 10
        service_score[0] = bad
        service_score[1] = average

    elif service[x] >= 40 and service[x] <= 60:
        average = 1
        service_score[1] = average

    elif service[x] > 60 and service[x] < 70:
        average = (70 - service[x]) / 10
        best = (service[x] - 60)
        service_score[1] = average
        service_score[2] = best

    elif service[x] >= 70 and service[x] <= 100:
        best = 1
        service_score[2] = best
```

```
#price

  if price[x] <= 2:
      cheap = 1
      price_score[0] = cheap

  elif price[x] > 2 and price[x] < 4:
      cheap = ((4 - price[x]) / 2)
      standard = ((price[x] - 2) / 2)
      price_score[0] = cheap
      price_score[1] = standard

  elif price[x] >= 4 and price[x] <= 6:
      standard = 1
      price_score[1] = standard

  elif price[x] > 6 and price[x] < 8:
      standard = ((8 - price[x]) / 2)
      expensive = ((price[x] - 6) / 2)
      price_score[1] = standard
      price_score[2] = expensive

  elif price[x] >= 8 and price[x] <= 10:
      expensive = 1
      price_score[2] = expensive

  #Inference
```

```
#Not recommended

NotRec = []
if service_score[0] == bad and price_score[0] == cheap:
    NotRec.append(min(service_score[0], price_score[0]))
if service_score[0] == bad and price_score[1] == standard:
    NotRec.append(min(service_score[0], price_score[1]))
if service_score[0] == bad and price_score[2] == expensive:
    NotRec.append(min(service_score[0], price_score[2]))
if service_score[1] == average and price_score[2] == expensive:
    NotRec.append(min(service_score[1], price_score[2]))
NotRec_score = max(NotRec)

#Recommended

Rec = []
if service_score[1] == average and price_score[0] == cheap:
    Rec.append(min(service_score[1], price_score[0]))
if service_score[1] == average and price_score[1] == standard:
    Rec.append(min(service_score[1], price_score[1]))
if service_score[2] == best and price_score[2] == expensive:
    Rec.append(min(service_score[2], price_score[2]))
Rec_score = max(Rec)

#Very recommended

VeryRec = []
if service_score[2] == best and price_score[0] == cheap:
    VeryRec.append(min(service_score[2], price_score[0]))
if service_score[2] == best and price_score[1] == standard:
    VeryRec.append(min(service_score[2], price_score[1]))
VeryRec_score = max(VeryRec)
```

```
divider = NotRec_score + Rec_score + VeryRec_score
if divider == 0:
    z = 0
else:
    z = ((NotRec_score*25) + (Rec_score*50) + (VeryRec_score*100))/divider
print(id_code[x],z)
y.append([id_code[x],z])
```

1 37.5
2 50.0
3 100.0
4 50.0
5 25.0
6 25.0
7 38.15789473684211
8 25.0
9 50.0
10 25.0
11 25.0
12 37.5
13 100.0
14 25.0
15 100.0
16 100.0
17 100.0
18 25.0
19 50.0
20 25.0
21 50.0
22 50.0
23 25.0
24 50.0
25 38.15789473684211
26 25.0

27 25.0
28 50.0
29 25.0
30 25.0
31 50.0
32 50.0
33 25.0
34 100.0
35 25.0
36 30.0
37 27.5
38 25.0
39 37.5
40 25.0
41 32.5
42 50.0
43 36.11111111111111
44 79.41176470588235
45 25.0
46 45.0
47 25.0
48 81.25
49 25.0
50 50.0
51 25.0
52 100.0
53 25.0
54 40.625
55 37.5
56 50.0
57 25.0
58 27.5
59 25.0
60 100.0
61 50.0

```
62 29.166666666666668
63 75.0
64 37.5
65 25.0
66 25.0
67 25.0
68 50.0
69 50.0
70 50.0
71 45.833333333333336
72 25.0
73 25.0
74 50.0
75 76.31578947368422
76 50.0
77 25.0
78 25.0
79 50.0
80 25.0
81 25.0
82 25.0
83 47.72727272727273
84 25.0
85 25.0
86 25.0
87 50.0
88 25.0
89 50.0
90 25.0
91 100.0
92 100.0
93 25.0
94 25.0
95 39.705882352941174
96 25.0
97 25.0
98 25.0
99 25.0
100 25.0
```

In [11]:
```
final_result = sorted(y, key = lambda x:x[1], reverse = True)
xlsx_output = {"Best auto Mechanics": final_result[:10]}

result_xlsx = pd.DataFrame(xlsx_output, columns = ["Best auto Mechanics"])
result_xlsx.to_excel("Ranking.xlsx")
print(xlsx_output)
```

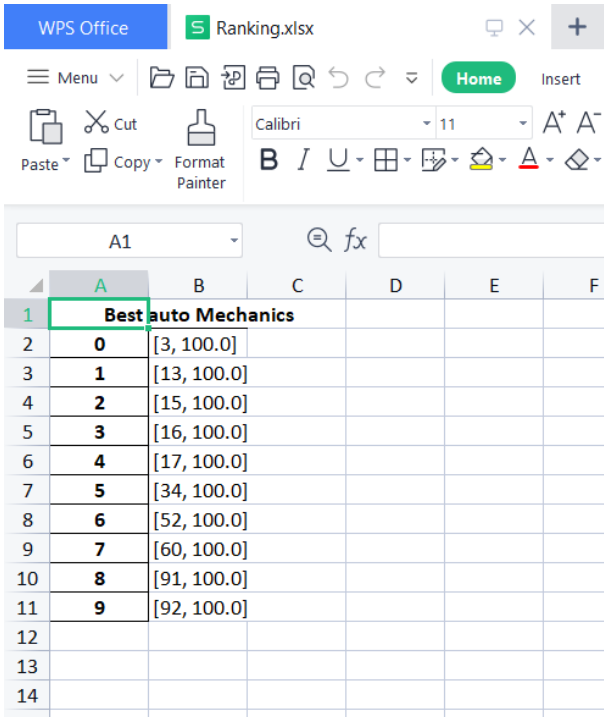
```
{'Best auto Mechanics': [[3, 100.0], [13, 100.0], [15, 100.0], [16, 100.0], [17, 100.0], [34, 100.0], [52, 100.0], [60, 100.0],
[91, 100.0], [92, 100.0]]}
```

Program Output

10 Best auto Mechanics in Bandung

```
{'Best auto Mechanics': [[3, 100.0], [13, 100.0], [15, 100.0], [16, 100.0], [17, 100.0], [34, 100.0], [52, 100.0], [60, 100.0],
[91, 100.0], [92, 100.0]]}
```

Output in Excel file



	A	B	C	D	E	F
1	**Best auto Mechanics**					
2	**0**	[3, 100.0]				
3	**1**	[13, 100.0]				
4	**2**	[15, 100.0]				
5	**3**	[16, 100.0]				
6	**4**	[17, 100.0]				
7	**5**	[34, 100.0]				
8	**6**	[52, 100.0]				
9	**7**	[60, 100.0]				
10	**8**	[91, 100.0]				
11	**9**	[92, 100.0]				
12						
13						
14						

TEAM MEMBERS CONTRIBUTION

NAME	TASKS
NUR SABRINA SYAZA BINTI ZAHAR HISHAM (1301213666)	- Source code - Report - Readme.txt - Presentation
NURUL IMAN BINTI NORDIN (1301213669)	- Source code - Report - Presentation - Compile Presentation Video
ZIDAN RIZKY WIJAYA (1301200226)	- Presentation