**PRACTICAL NO.: 01**

**Design an Expert system using AIML**

**AIM: To design an expert system for responding the patient query for identifying the flu.**

**PYTHON CODE:-**

!pip install python-aiml

import aiml

def main():

kernel = aiml.Kernel()

kernel.learn("flu.aiml")

print("Welcome to the flu diag center")

while True:

user\_input = input("You: ").strip().lower()

if user\_input == 'exit':

print("Goodbye!")

break

response = kernel.respond(user\_input)

print("Expert System: " + response)

if \_\_name\_\_ == "\_\_main\_\_":

main ()

**The flu.aiml file:-**

<aiml>

<category>

<pattern>WELCOME</pattern>

<template>Welcome to the flu diag center. Please enter your symptoms.</template>

</category>

<category>

<pattern>FEVER</pattern>

<template>No flu.</template>

</category>

<category>

<pattern>COUGH</pattern>

<template>No flu.</template>

</category>

<category>

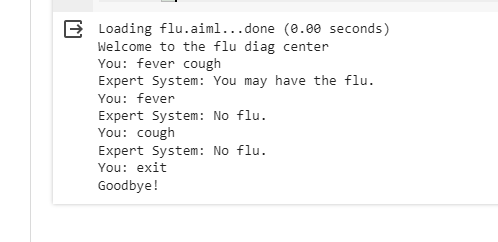
<pattern>FEVER COUGH</pattern>

<template>You may have the flu.</template>

</category>

</aiml>

**OUTPUT:-**



**PRACTICAL NO.: 02**

**Design a bot using AIML**

**AIM: To design a bot using AIML**

**PYTHON CODE:-**

!pip install python-aiml

import aiml

kernel = aiml.Kernel()

kernel.learn("bot.aiml")

while True:

user\_input = input("You: ")

if user\_input.lower() == "exit":

print("Bot: Goodbye!")

break

bot\_response = kernel.respond(user\_input)

print("Bot:", bot\_response)

bot\_response = kernel.respond(user\_input)

print("Chatbot:", bot\_response)

**The bot.aiml file:-**

<aiml>

<category>

<pattern>HI</pattern>

<template>Hello there!</template>

</category>

<category>

<pattern>HELLO</pattern>

<template>Hi! How can I assist you today?</template>

</category>

<category>

<pattern>WHAT IS THE WEATHER LIKE TODAY</pattern>

<template>Sorry, I don't have access to real-time weather information.</template>

</category>

<category>

<pattern>WHAT IS YOUR NAME</pattern>

<template>My name is SmileBOT.</template>

</category>

<category>

<pattern>TELL ME A JOKE</pattern>

<template>I failed math so many times at school, I cant even count.</template>

</category>

<category>

<pattern>WHAT IS THE MEANING OF LIFE</pattern>

<template>The meaning of life is a profound philosophical question, and the answer may vary from

person to person.</template>

</category>

<category>

<pattern>WHO WON THE FOOTBALL WORLD CUP IN 2023</pattern>

<template>ARGENTINA WON THE WORLD CUP.</template>

</category>

<category>

<pattern>TELL ME ABOUT YOURSELF</pattern>

<template>I am a friendly chatbot created to assist and provide information to users.</template>

</category>

<category>

<pattern>WHAT IS THE CAPITAL OF TURKEY</pattern>

<template>The capital of Turkey is Ankara.</template>

</category>

<category>

<pattern>DO YOU LIKE ICE CREAM</pattern>

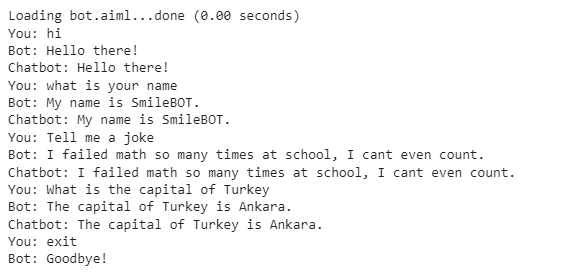
<template>Unfortunately, I am just a computer program and do not have the ability to taste or enjoy

food.</template>

</category>

</aiml>

**OUTPUT:-**



**PRACTICAL NO.: 03**

**Implement Bayes Theorem using Python**

**AIM: To implement Bayes Theorem using Python.**

**PYTHON CODE:-**

def bayes\_theorem(prior\_prob, likelihood, evidence):

# Calculate the posterior probability using Bayes' theorem

posterior\_prob = (likelihood \* prior\_prob) / evidence

return posterior\_prob

if \_\_name\_\_ == "\_\_main\_\_":

# Given values

prior\_prob = 0.01 # Prior probability of having cancer

likelihood\_cancer = 0.95 # Likelihood of getting a positive test result given cancer

likelihood\_no\_cancer = 0.10 # Likelihood of getting a positive test result given no cancer

# Calculate the evidence using the law of total probability

evidence = (likelihood\_cancer \* prior\_prob) + (likelihood\_no\_cancer \* (1 - prior\_prob))

# Calculate the posterior probability

posterior\_prob = bayes\_theorem(prior\_prob, likelihood\_cancer, evidence)

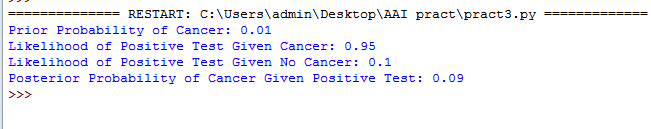
print("Prior Probability of Cancer:", prior\_prob)

print("Likelihood of Positive Test Given Cancer:", likelihood\_cancer)

print("Likelihood of Positive Test Given No Cancer:", likelihood\_no\_cancer)

print("Posterior Probability of Cancer Given Positive Test:", round(posterior\_prob, 2))

**OUTPUT:-**



**PRACTICAL NO.: 04**

**Implement Conditional Probability and Joint Probability using Python.**

**AIM: To implement Conditional Probability and Joint probability using Python.**

**Conditional Probability:-**

**PYTHON CODE:-**

def get\_valid\_probability\_input(prompt):

while True:

try:

probability = float(input(prompt))

if 0 <= probability <= 1:

return probability # Valid input, return the probability

else:

print("Probability must be between 0 and 1. Please try again.")

except ValueError:

print("Invalid input. Please enter a valid probability between 0 and 1.")

# Get the probability values from the user

P\_B = get\_valid\_probability\_input("Enter the probability of event B (0 to 1): ")

P\_A\_and\_B = get\_valid\_probability\_input("Enter the probability of events A and B (0 to 1): ")

# Calculate conditional probability P(A|B)

P\_A\_given\_B = P\_A\_and\_B / P\_B

# Check if the calculated conditional probability is greater than 1

if P\_A\_given\_B > 1:

print("Inconsistent result. Please check the inputs again.")

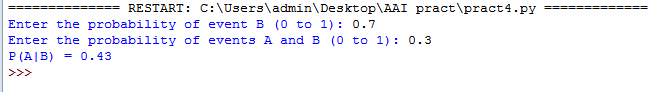
else:

# Format the result to two decimal places

formatted\_result = "{:.2f}".format(P\_A\_given\_B)

print("P(A|B) =", formatted\_result)

**OUTPUT:-**



**Joint Probability:-**

**PYTHON CODE:-**

# Get the probability values from the user

while True:

try:

P\_A = float(input("Enter the probability of event A (0 to 1): "))

if 0 <= P\_A <= 1:

break # Valid input, exit the loop

else:

print("Probability must be between 0 and 1. Please try again.")

except ValueError:

print("Invalid input. Please enter a valid probability between 0 and 1.")

while True:

try:

P\_B = float(input("Enter the probability of event B (0 to 1): "))

if 0 <= P\_B <= 1:

break # Valid input, exit the loop

else:

print("Probability must be between 0 and 1. Please try again.")

except ValueError:

print("Invalid input. Please enter a valid probability between 0 and 1.")

# Calculate joint probability P(A and B)

P\_A\_and\_B = P\_A \* P\_B

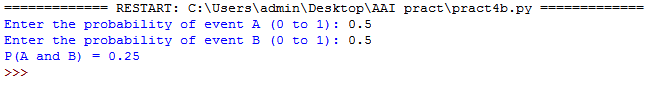
# Format the result to two decimal places

formatted\_result = "{:.2f}".format(P\_A\_and\_B)

# Print the result

print("P(A and B) =", formatted\_result)

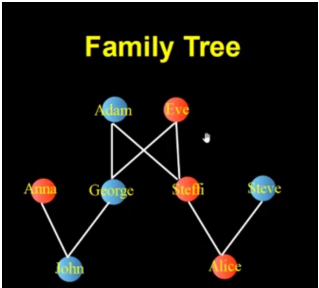
**OUTPUT:-**



**PRACTICAL NO.: 05**

**Write a program for to implement Rule based system.**

**AIM: To write a program to implement Rule based system (family tree).**

****

**PROLOG CODE:-**

male(adam).

male(george).

male(steve).

male(john).

female(alice).

female(anna).

female(eve).

female(steffi).

parent(adam,george).

parent(eve,george).

parent(adam,steffi).

parent(eve,steffi).

parent(george,john).

parent(anna,john).

parent(steffi,alice).

parent(steve,alice).

mother(X,Y) :- parent(X,Y), female(X).

father(X,Y) :- parent(X,Y), male(X).

sister(X,Y) :- parent(Z,X), parent(Z,Y), female(X),X\==Y.

brother(X,Y) :- parent(Z,X), parent(Z,Y), male(X),X\==Y.

grandfather(X,Z) :- father(X,Y), parent(Y,Z).

grandmother(X,Z) :- mother(X,Y), parent(Y,Z).

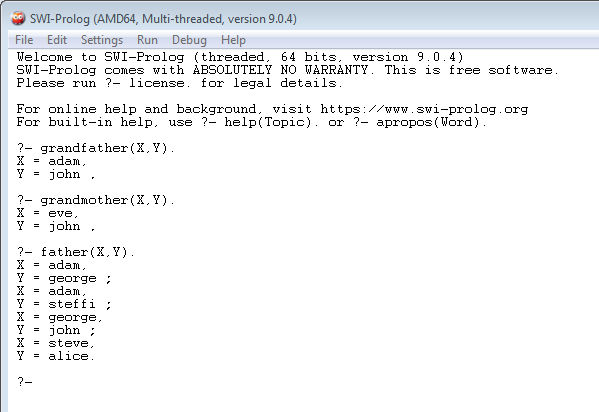
siblings(X,Y) :- (brother(X,Y);sister(X,Y)),X\==Y.

uncle(X,Y) :- parent(Z,Y),brother(X,Z).

aunty(X,Y) :- parent(Z,Y),sister(X,Z).

cousin(X,Y) :- parent(A,X),parent(B,Y),siblings(A,B).

**OUTPUT:-**



**PRACTICAL NO.: 06**

**Design a Fuzzy based application using Python**

**AIM: To Design a Fuzzy based application using Python.**

**PYTHON CODE:-**

!pip install numpy

!pip install scikit-fuzzy

#import the library pip install numpy

#import the library pip install scikit-fuzzy

import numpy as np

import skfuzzy as fuzz

from skfuzzy import control as ctrl

# Create fuzzy variables and their ranges

quality = ctrl.Antecedent(np.arange(0, 11, 1), 'quality')

service = ctrl.Antecedent(np.arange(0, 11, 1), 'service')

tip = ctrl.Consequent(np.arange(0, 26, 1), 'tip')

# Define membership functions for quality and service

quality['poor'] = fuzz.trimf(quality.universe, [0, 0, 5])

quality['average'] = fuzz.trimf(quality.universe, [0, 5, 10])

quality['excellent'] = fuzz.trimf(quality.universe, [5, 10, 10])

service['poor'] = fuzz.trimf(service.universe, [0, 0, 5])

service['average'] = fuzz.trimf(service.universe, [0, 5, 10])

service['excellent'] = fuzz.trimf(service.universe, [5, 10, 10])

# Define membership functions for the tip

tip['low'] = fuzz.trimf(tip.universe, [0, 0, 13])

tip['medium'] = fuzz.trimf(tip.universe, [0, 13, 25])

tip['high'] = fuzz.trimf(tip.universe, [13, 25, 25])

# Define fuzzy rules

rule1 = ctrl.Rule(quality['poor'] | service['poor'], tip['low'])

rule2 = ctrl.Rule(service['average'], tip['medium'])

rule3 = ctrl.Rule(quality['excellent'] | service['excellent'], tip['high'])

# Create the fuzzy control system

tipping\_ctrl = ctrl.ControlSystem([rule1, rule2, rule3])

# Create a simulation

tipping = ctrl.ControlSystemSimulation(tipping\_ctrl)

# Input values for quality and service

tipping.input['quality'] = 6.5

tipping.input['service'] = 9.8

# Compute the tipping result

tipping.compute()

# Print the output tip value

print("Recommended tip:", tipping.output['tip'])

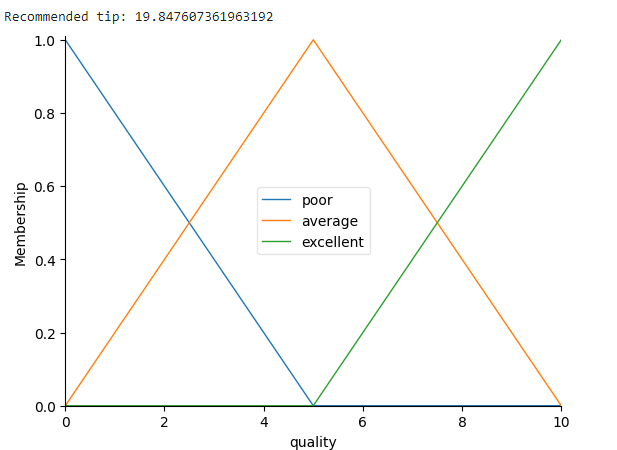
# Visualize the membership functions and the output

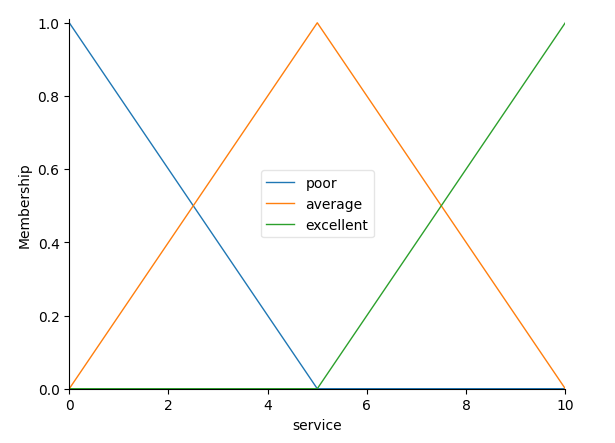
quality.view()

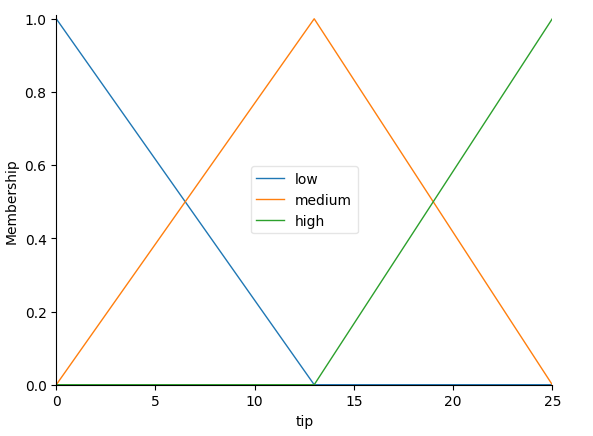
service.view()

tip.view()

**OUTPUT:-**







**PRACTICAL NO.: 07**

**Implement supervise and un-supervised learning model using python.**

**AIM: Write an application to simulate supervised and un-supervised learning model.**

**Supervised Learning Model:-**

**PYTHON CODE:-**

!pip install numpy

!pip install matplotlib

!pip install pandas

!pip install scipy

!pip install seaborn

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from sklearn.linear\_model import LogisticRegression

from sklearn import datasets

# Importing the dataset

dataset = pd.read\_csv("iris.csv")

dataset.describe()

# Splitting the dataset into the Training set and Test set

X = dataset.iloc[:, [0,1,2, 3]].values

y = dataset.iloc[:, 4].values

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 0)

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

# Fitting Logistic Regression to the Training set

classifier = LogisticRegression(random\_state = 0, solver='lbfgs', multi\_class='auto')

classifier.fit(X\_train, y\_train)

# Predicting the Test set results

y\_pred = classifier.predict(X\_test)

# Predict probabilities

probs\_y=classifier.predict\_proba(X\_test)

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

print(cm)

# Plot confusion matrix

import seaborn as sns

import pandas as pd

#confusion matrix sns heatmap

ax = plt.axes()

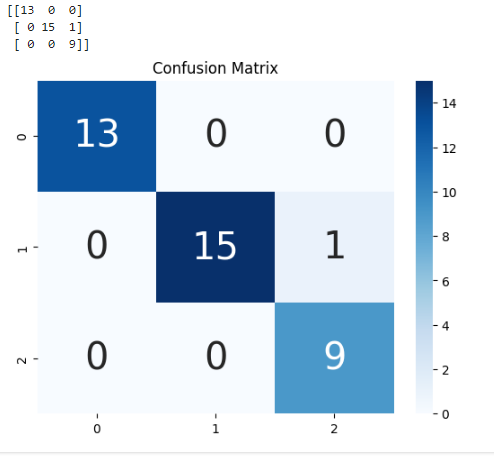
df\_cm = cm

sns.heatmap(df\_cm, annot=True, annot\_kws={"size": 30}, fmt='d',cmap="Blues", ax = ax )

ax.set\_title('Confusion Matrix')

plt.show()

**OUTPUT:-**

****

**Un-Supervised Learning Model:-**

**PYTHON CODE:-**

!pip install matplotlib

!pip install pandas

!pip install numpy

!pip install scipy

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

customer\_data = pd.read\_csv('Mall\_Customers.csv')

customer\_data.shape

customer\_data.head()

data = customer\_data.iloc[:, 3:5].values

import scipy.cluster.hierarchy as shc

plt.figure(figsize=(10, 7))

plt.title("Customer Dendograilocms")

dend = shc.dendrogram(shc.linkage(data, method='ward'))

from sklearn.cluster import AgglomerativeClustering

cluster = AgglomerativeClustering(n\_clusters=5, affinity='euclidean', linkage='ward')

cluster.fit\_predict(data)

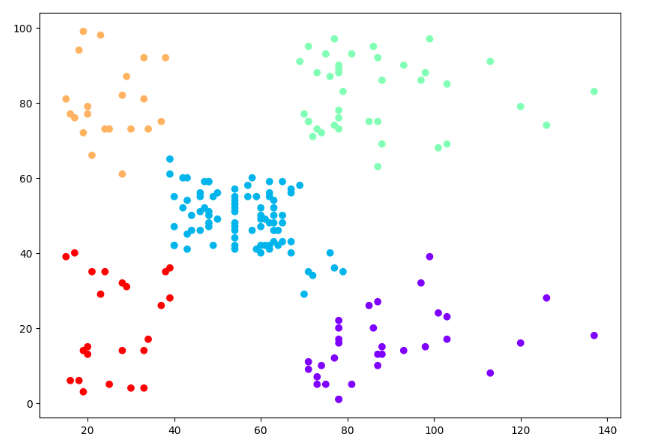
plt.figure(figsize=(10, 7))

plt.scatter(data[:,0], data[:,1], c=cluster.labels\_, cmap='rainbow')

plt.show()

**OUTPUT:-**

****

****

**PRACTICAL NO.: 08**

**Design Clustering Algorithm using Python**

**AIM: Write an application to implement Clustering Algorithm using python.**

**PYTHON CODE:-**

!pip install numpy

!pip install matplotlib

!pip install scikit-learn

# synthetic classification dataset

from numpy import where

from sklearn.datasets import make\_classification

from matplotlib import pyplot

# define dataset

X, y = make\_classification(n\_samples=1000, n\_features=2, n\_informative=2, n\_redundant=0, n\_clusters\_per\_class=1, random\_state=4)

# create scatter plot for samples from each class

for class\_value in range(2):

# get row indexes for samples with this class

row\_ix = where(y == class\_value)

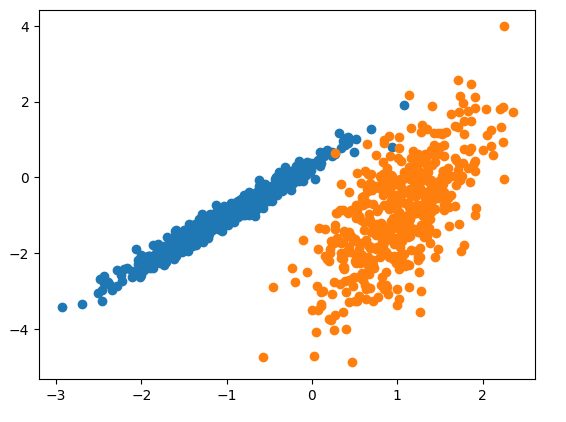
# create scatter of these samples

pyplot.scatter(X[row\_ix, 0], X[row\_ix, 1])

# show the plot

pyplot.show()

**OUTPUT:-**

****