****

****

**SAKET GYANPEETH’S**

**SAKET COLLEGE OF ARTS, SCIENCE AND COMMERCE**

**(Permanently Affiliated to University of Mumbai)**

**NAAC Accredited B Grade**

Saket Vidyanagari Marg, Chinchpada Road, Katemanivali, Kalyan (East) – 421306, Dist. Thane (MAH)

Department of Information Technology

**CERTIFICATE**

**This is to certify that**

**SURAJ GIRJA GUPTA** of **MSc Information Technology Part-II** Class has satisfactory carried out the required practical in the subject.

**MACHINE LEARNING**

For the Academic year 2023 – 2024

**Practical In-Charge Head of Department External Examiner**

# INDEX

|  |  |  |
| --- | --- | --- |
| **Sr.no.** | **Practical** | **Page no.** |
| **1** | **Design an Expert system using AIML.** | **3-4** |
| **2** | **Design a bot using AIML.** | **5-7** |
| **3** | **Implement Bayes Theorem using Python.** | **8-9** |
| **4** | **Implement Conditional Probability And Joint Probability using Python.** | **10-12** |
| **5** | **Design a Fuzzy based application using Python.** | **13-17** |
| **6** | **Write an application to simulate supervised and un-supervised learning model.** | **18-22** |
| **7** | **Write an application to implement clustering algorithm.** | **23** |
| **8** | **Write an application to implement BFS algorithm.** | **24-25** |
| **9** | **Write an application to implement DFS algorithm.** | **26-27** |

# Practical: 1

**Aim: Design an Expert system using AIML.**

**Description:**

**What is an Expert System?**

An expert system is a computer program that is designed to solve complex problems and to provide decision-making ability like a human expert. It performs this by extracting knowledge from its knowledge base using the reasoning and inference rules according to the user queries.

**Code:**

import aiml

def main():

  kernel = aiml.Kernel()

  kernel.learn("basic\_chat.aiml")

  print("Welcome to Bot World")

  while True:

    input\_text = input(">Human: ").strip().upper()

    if input\_text == 'EXIT':

      print("Goodbye!")

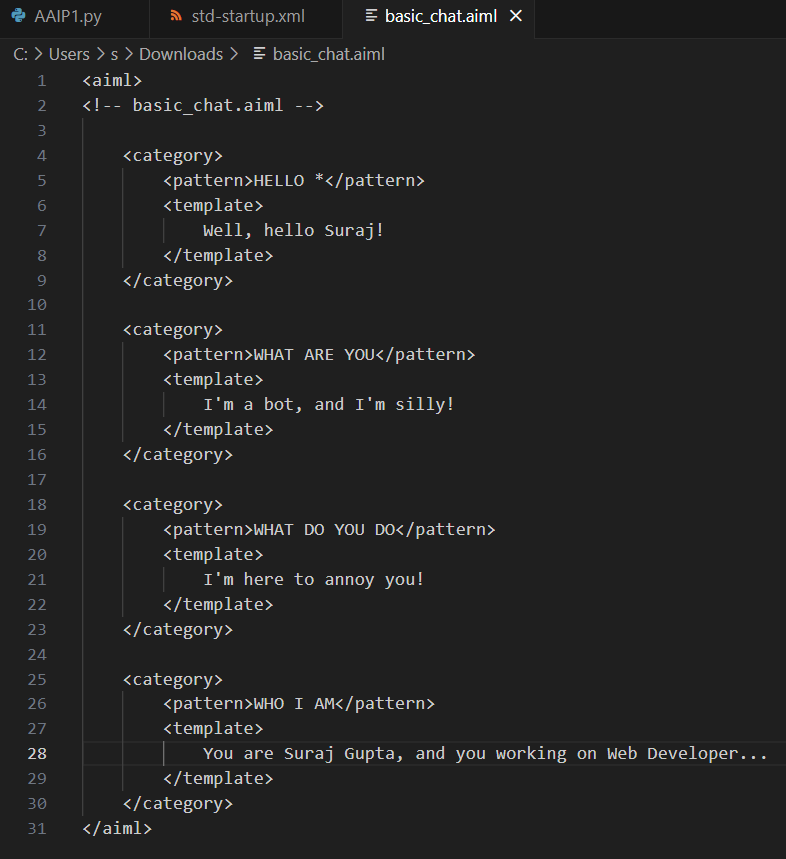
      break

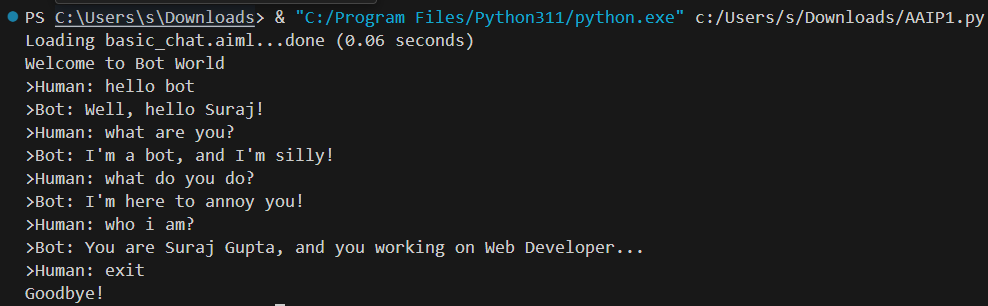
    response = kernel.respond(input\_text)

    print(">Bot: "+response)

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

  main()



**Output: **

# Practical: 2

**Aim: Design a bot using AIML.**

**Description:**

**What is AIML?**

AIML stands for Artificial Intelligence Modelling Language. AIML is an XML based markup language meant to create artificial intelligent applications. AIML makes it possible to create human interfaces while keeping the implementation simple to program, easy to understand and highly maintainable. This tutorial will teach you the basics of AIML. All the basic components of AIML with suitable examples have been discussed in this tutorial.

**AIML Tags/Description**

⦁ <aiml> − defines the beginning and end of a AIML document.

⦁ <category> − defines the unit of knowledge in bot’s knowledge base.

⦁ <pattern> − defines the pattern to match what a user may input to an bot.

⦁ <template> − defines the response of a bot to user's input.

**Code:**

import aiml

def main():

  kernel = aiml.Kernel()

  kernel.learn("std-startup.xml")

  kernel.respond("load aiml b")

  print("Welcome to Bot World")

  while True:

    input\_text = input(">Human: ").strip().upper()

    if input\_text == 'EXIT':

      print("Goodbye!")

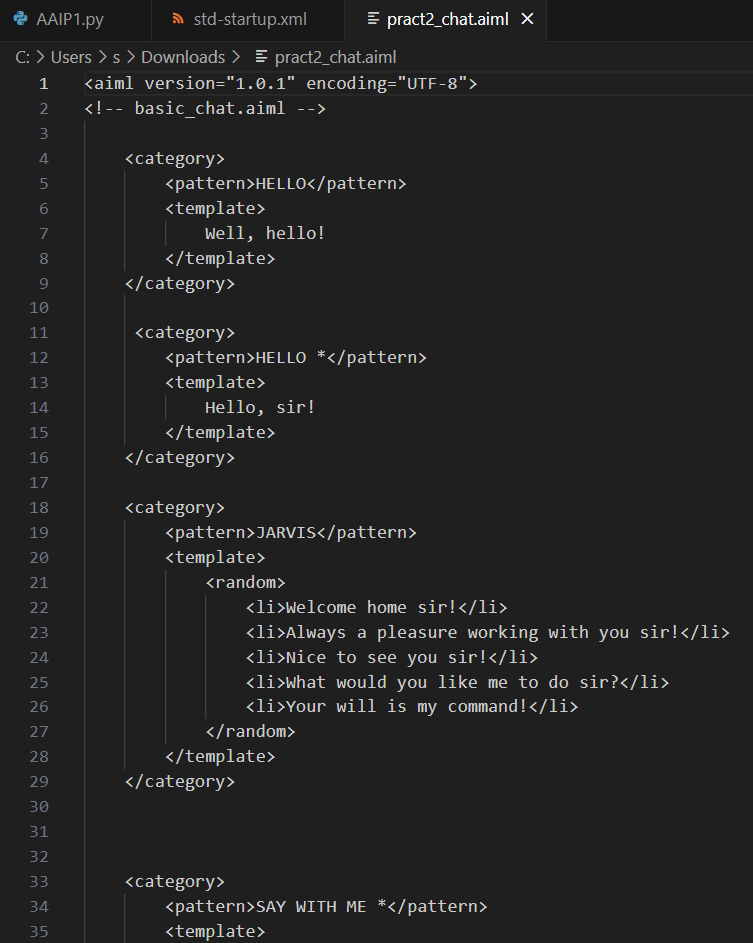
      break

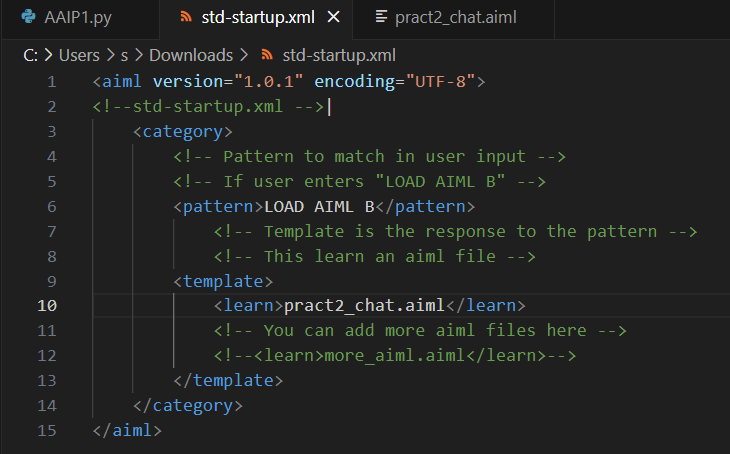
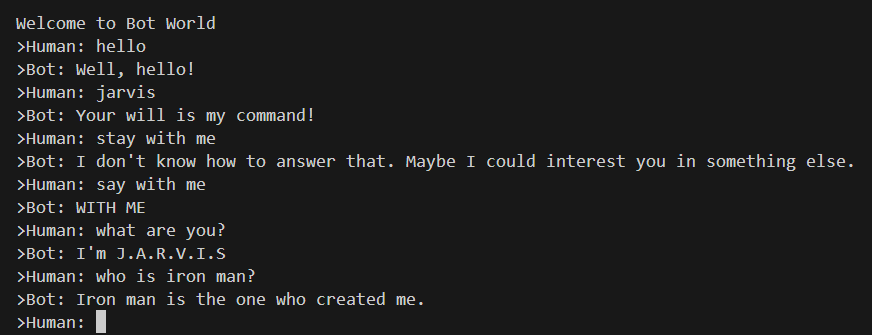
    response = kernel.respond(input\_text)

    print(">Bot: "+response)

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

  main()



**Output: **

# Practical: 3

**Aim: Implement Bayes Theorem using Python.**

**Description:**

Bayes’ Theorem provides a way that we can calculate the probability of a piece of data belonging to a given class, given our prior knowledge. Bayes’ Theorem is stated as:

**P(class|data) = (P(data|class) \* P(class)) / P(data)**

Where P(class|data) is the probability of class given the provided data.

Naive Bayes is a classification algorithm for binary (two-class) and multiclass classification problems. It is called Naive Bayes or idiot Bayes because the calculations of the probabilities for each class are simplified to make their calculations tractable.

**Code:**

def drug\_user(

    prob\_th=0.5,

    sensitivity=0.99,

    specificity=0.99,

    prevelance=0.01,

    verbose=True):

    p\_user = prevelance

    p\_non\_user = 1-prevelance

    p\_pos\_user = sensitivity

    p\_neg\_user = specificity

    p\_pos\_non\_user = 1-specificity

    num = p\_pos\_user\*p\_user

    den = p\_pos\_user\*p\_user+p\_pos\_non\_user\*p\_non\_user

    prob = num/den

    if verbose:

        if prob > prob\_th:

            print("The test-taker could be an user")

        else:

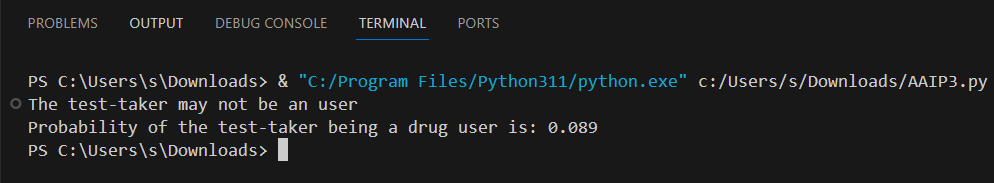
            print("The test-taker may not be an user")

    return prob

p=drug\_user(prob\_th=0.5, sensitivity=0.97, specificity=0.95, prevelance=0.005)

print("Probability of the test-taker being a drug user is:", round(p,3))

**Output:**



# Practical: 4

**Aim: Implement Conditional Probability and joint probability using Python.**

**Description:**

**What is Conditional Probability?**

The probability of one event given the occurrence of another event is called the conditional probability. The conditional probability of one to one or more random variables is referred to as the conditional probability distribution.

For example, the conditional probability of event A given event B is written formally as:

* **P(A given B)**

The “given” is denoted using the pipe “|” operator; for example:

* **P(A | B)**

The conditional probability for events A given event B is calculated as follows: • **P(A given B) = P(A and B) / P(B)**

**Code:**

def conditional():

    pass\_stats=float(input('Enter Number '))

    pass\_codingWStats=float(input('Enter Number '))

    pass\_codingStats=float(input('Enter Number '))

    prob\_both = pass\_stats\*pass\_codingStats

    print('The probability of passing is', round(prob\_both,3))

    prob\_coding=(prob\_both) + ((1-pass\_stats)\*pass\_codingStats)

    print('Probability that he/she passes only coding',round(prob\_coding,3))

    stats\_given\_coding=prob\_both/prob\_coding

    print('Conditional probabily is', round(stats\_given\_coding,3))

print('Hey Rameshwar')

conditional()

**Output:**



**Description:**

**What is Joint Probability?**

The probability of two (or more) events is called the joint probability. The joint probability of two or more random variables is referred to as the joint probability distribution. The joint probability for events A and B is calculated as the probability of event A given event B multiplied by the probability of event B.

This can be stated formally as follows:

**P(A and B) = P(A given B) \* P(B)**

The calculation of the joint probability is sometimes called the fundamental rule of probability or the “product rule” of probability or the “chain rule” of probability

**P(A and B) = P(A given B) \* P(B) = P(B given A) \* P(A)**

**Code:**

import numpy as np

import seaborn as sns

import pandas as pd

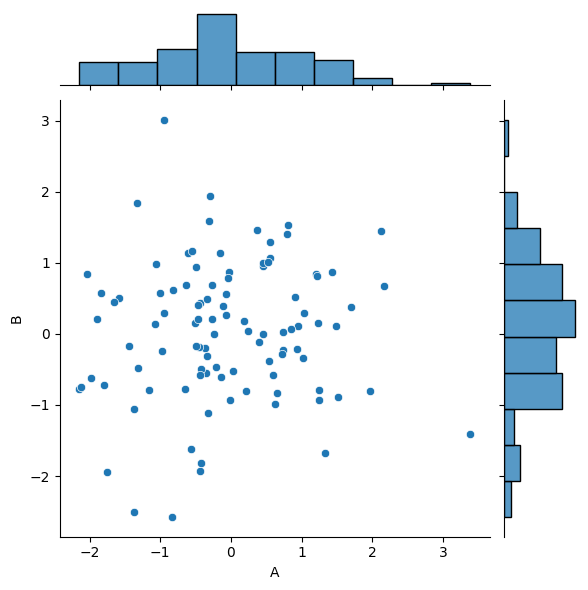
A = np.random.normal(size=100)

B = np.random.normal(size=100)

df = pd.DataFrame({'A' : A , 'B':B})

sns.jointplot(x='A', y='B' ,data=df )

**Output:**



# Practical: 5

**Aim: Design a Fuzzy based application using Python / R Description:**

**What is Fuzzy based application?**

Fuzzy sets were introduced by Lotfi Zadeh (1921–2017) in 1965.

Unlike crisp sets, a fuzzy set allows partial belonging to a set, that is defined by a degree of membership, denoted by µ, that can take any value from 0 (element does not belong at all in the set) to 1 (element belongs fully to the set).

It is evident that if we remove all the values of belonging except from 0 and 1, the fuzzy set will collapse to a crisp set that was described in the previous section.

**Code:**

import numpy as np

import skfuzzy as fuzz

from skfuzzy import control as ctrl

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')

quality.automf(3)

service.automf(3)

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])

quality['average'].view()

service.view()

tip.view()

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

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

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

rule1.view()

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

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

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

rule1.view()

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

tipping= ctrl.ControlSystemSimulation(tipping\_ctrl)

tipping.input['quality'] = 6.5

tipping.input['service'] = 9.8

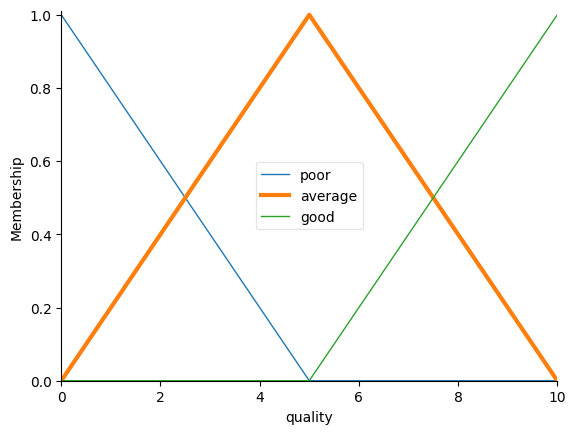
tipping.compute()

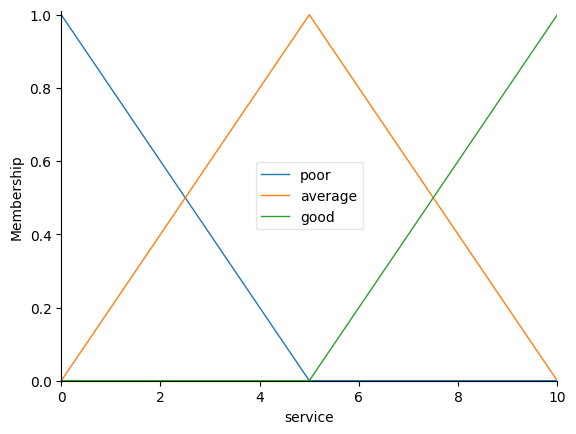
print(tipping.output['tip'])

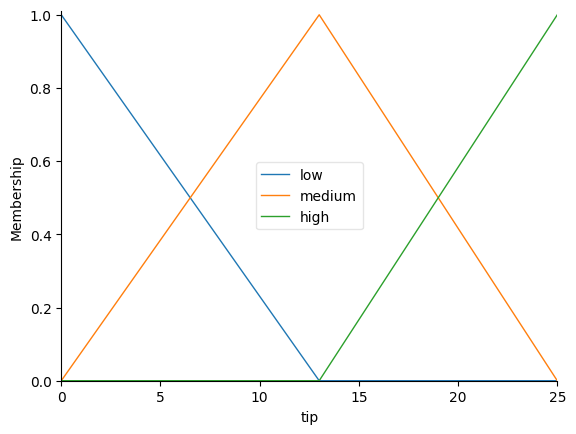
tip.view(sim=tipping)

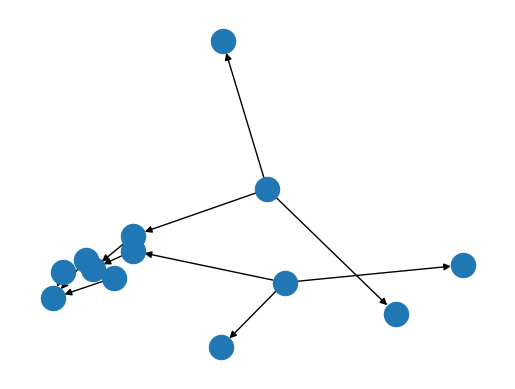
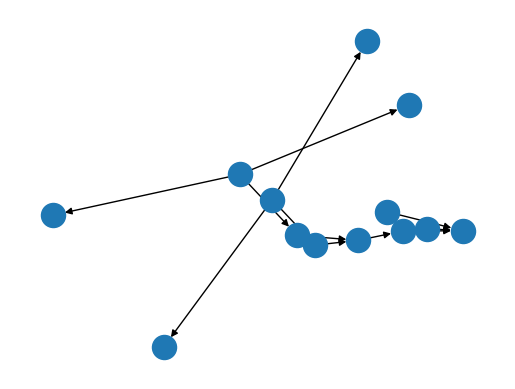
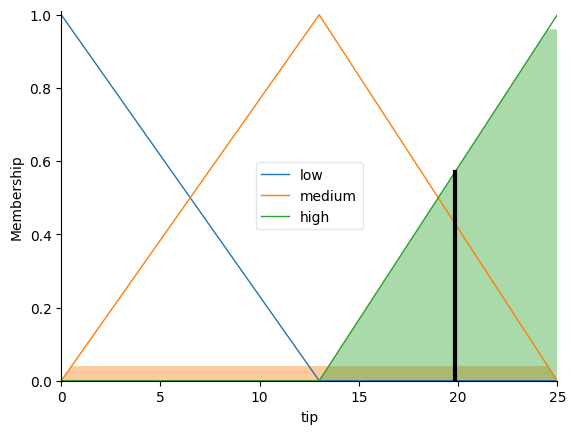
**Output:**

19.847607361963192







# Practical: 6

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

**Description:**

**What is supervised learning?**

Supervised learning as the name indicates the presence of a supervisor as a teacher. Basically, supervised learning is a learning in which we teach or train the machine using data which is well labelled that means some data is already tagged with the correct answer.

Supervised learning classified into two categories of algorithms:

⦁ Classification: A classification problem is when the Outputvariable is a category, such as “Red” or “blue” or “disease” and “no disease”.

⦁ Regression: A regression problem is when the Outputvariable is a real value, such as “dollars” or “weight”.

Supervised learning deals with or learns with “labelled” data. Which implies that some data is already tagged with the correct answer.

Types: -

⦁ **Regression**

⦁ **Logistic Regression**

⦁ **Classification**

⦁ **Naive Bayes Classifiers**

⦁ **K-NN (k nearest neighbours)**

⦁ **Decision Trees** ⦁ **Support Vector Machine**

**Code:**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from sklearn.linear\_model import LogisticRegression

from sklearn import datasets

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

dataset.describe()

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)

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

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

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

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

from sklearn.metrics import confusion\_matrix

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

print(cm)

import seaborn as sns

import pandas as pd

ax = plt.axes()

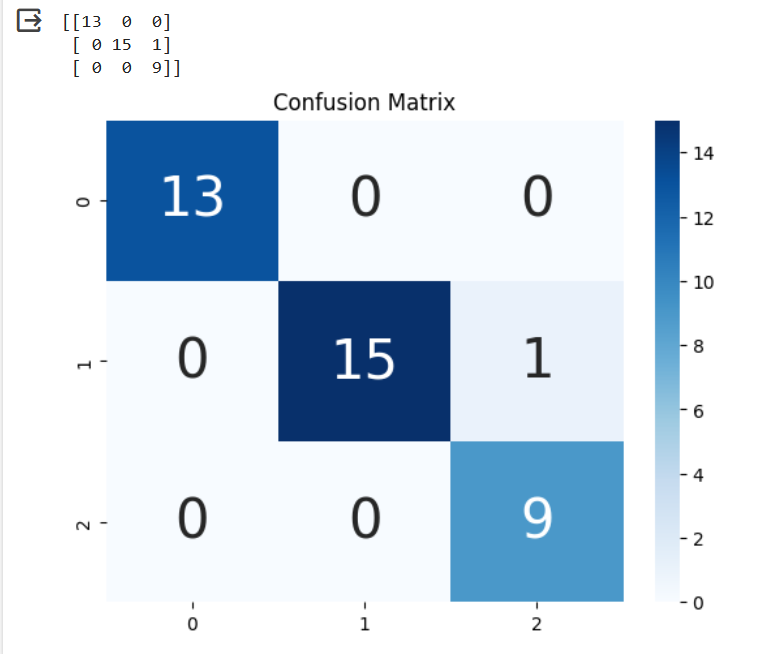
df\_cm = cm

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

ax.set\_title('Confusion Matrix')

plt.show()

**Output:**



**Description:**

**What is Unsupervised Learning?**

Unsupervised learning is the training of machine using information that is neither classified nor labelled and allowing the algorithm to act on that information without guidance. Here the task of machine is to group unsorted information according to similarities, patterns and differences without any prior training of data. Unsupervised learning classified into two categories of algorithms:

⦁ Clustering: A clustering problem is where you want to discover the inherent groupings in the data, such as grouping customers by purchasing behaviour.

⦁ Association: An association rule learning problem is where you want to discover rules that describe large portions of your data, such as people that buy X also tend to buy Y.

Types of Unsupervised Learning: -

**Clustering**

⦁ Exclusive (partitioning)

⦁ Agglomerative

⦁ Overlapping

⦁ Probabilistic

**Clustering Types: -**

⦁ Hierarchical clustering

⦁ K-means clustering

⦁ Principal Component Analysis

⦁ Singular Value Decomposition ⦁ Independent Component Analysis

**Code:**

from sklearn import datasets

import matplotlib.pyplot as plt

iris\_df = datasets.load\_iris()

print(dir(iris\_df))

print(iris\_df.feature\_names)

print(iris\_df.target)

print(iris\_df.target\_names)

label = {0: 'red', 1: 'blue', 2: 'green'}

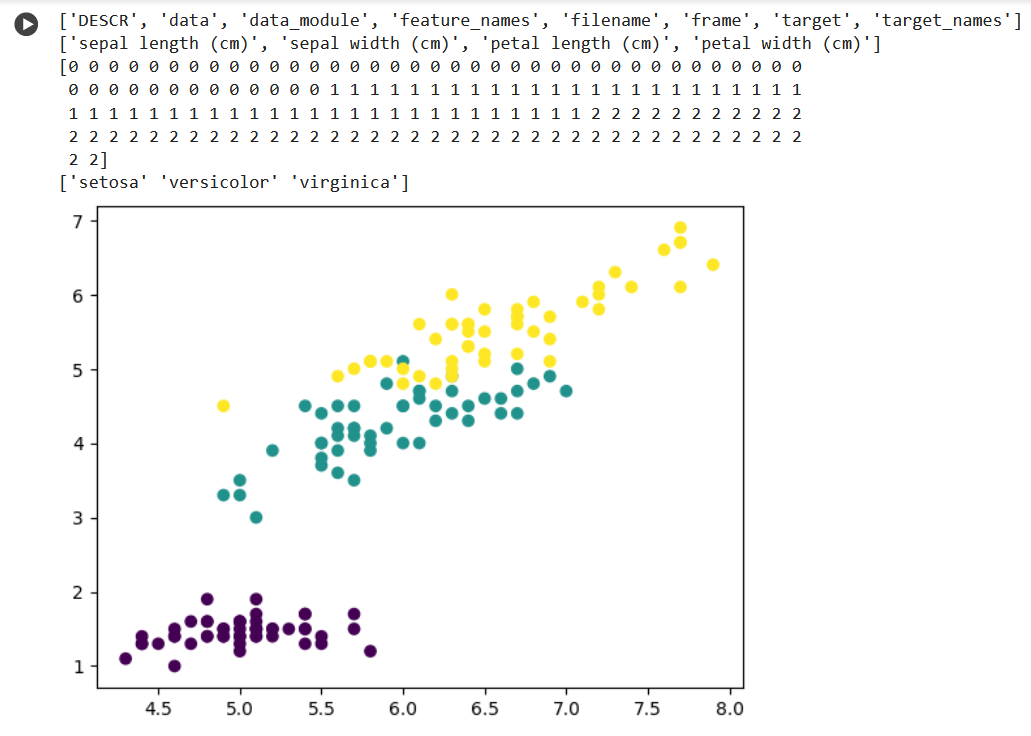
x\_axis = iris\_df.data[:, 0]

y\_axis = iris\_df.data[:, 2]

plt.scatter(x\_axis, y\_axis, c=iris\_df.target)

plt.show()

**Output:**



# Practical: 7

**Aim: Write an application to implement Clustering algorithm.**

**Description:**

**What is Clustering?**

Clustering is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group than those in other groups.

**Code:**

from numpy import where

from sklearn.datasets import make\_classification

from matplotlib import pyplot

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

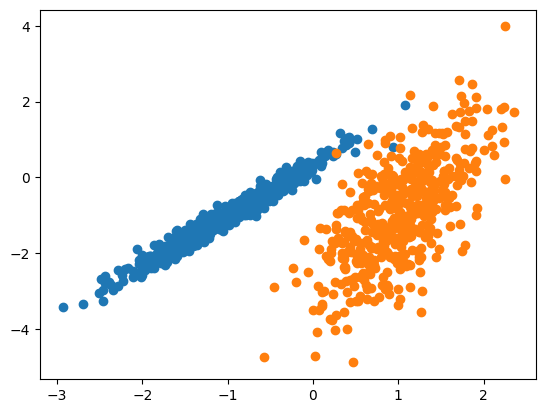
for class\_value in range(2):

  row\_ix = where(y == class\_value)

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

pyplot.show()

**Output:**



# Practical: 8

**Aim: Write an Program to implement BFS algorithm.**

**Description:**

**What is Breadth-First Search?**

Breadth-First Search (BFS) is an algorithm used for traversing graphs or trees. Traversing means visiting each node of the graph. Breadth-First Search is a recursive algorithm to search all the vertices of a graph or a tree. BFS in python can be implemented by using data structures like a dictionary and lists. As breadth-first search is the process of traversing each node of the graph, a standard BFS algorithm traverses each vertex of the graph into two parts:

1. Visited
2. Not Visited. So, the purpose of the algorithm is to visit all the vertex while avoiding cycles.

The steps of the algorithm work as follow:

* 1. Start by putting any one of the graph’s vertices at the back of the queue.
  2. Now take the front item of the queue and add it to the visited list.
  3. Create a list of that vertex's adjacent nodes. Add those which are not within the visited list to the rear of the queue.
  4. Keep continuing steps two and three till the queue is empty.

**Code:**

import collections

def bfs(graph,root):

    visited, queue = set(), collections.deque([root])

    visited.add(root)

    while queue:

        vertex = queue.popleft()

        print(str(vertex)+" ", end="")

        for neighbour in graph[vertex]:

            if neighbour not in visited:

                visited.add(neighbour)

                queue.append(neighbour)

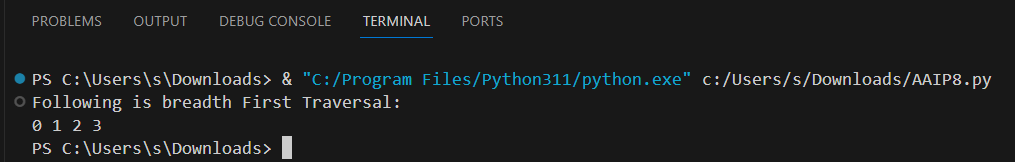
if \_\_name\_\_ == '\_\_main\_\_':

    graph = {0:[1,2],1:[2],2:[3],3:[1,2]}

    print("Following is breadth First Traversal:")

    bfs(graph,0)

**Output:**



# Practical: 9

**Aim: Write an Program to implement DFS algorithm.**

**Description:**

**What is Depth-First Search ?**

The Depth-First Search is a recursive algorithm that uses the concept of backtracking. It involves thorough searches of all the nodes by going ahead if potential, else by backtracking. Here, the word backtrack means once you are moving forward and there are not any more nodes along the present path, you progress backward on an equivalent path to seek out nodes to traverse.

**Algorithm:**

* Create a recursive function that takes the index of the node and a visited array.
* Mark the current node as visited and print the node.
* Traverse all the adjacent and unmarked nodes and call the recursive function with the index of the adjacent node.

**Code:**

#DFS algorithm

def dfs(graph,start,visited=None):

    if visited is None:

        visited = set()

    visited.add(start)

    print(start)

    for next in graph[start]-visited:

        dfs(graph, next, visited)

    return visited

graph= {'0':set(['1','2']),

        '1':set(['0','3','4']),

        '2':set(['0']),

        '3':set(['1']),

        '4':set(['2','3'])}

dfs(graph,'0')

**Output:**

