1. **Implement Conditional Probability and Joint Probability. Using Python.**

**CONDITIONAL**

def conditional():

    pass\_stats = 0.15

    pass\_codingWStats = 0.60

    pass\_codingWOStats = 0.40

    prob\_both = pass\_stats \*pass\_codingWStats

    print("The probability that applicant passes both is", round(prob\_both, 3))

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

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

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

    print("Conditional probabilty is", round(stats\_given\_coding, 3))

print("Hey Hulk")

conditional()

**JOINT**

import numpy as np

import pandas as pd

from scipy import stats

import matplotlib.pyplot as plt

import seaborn as sns

sns.set()

df\_obj1 = pd.DataFrame({"x": np.random.randn(500),

                   "y": np.random.randn(500)})

sns.jointplot(x="x", y="y", data=df\_obj1, kind="kde");

dataset = sns.load\_dataset("tips")

plt.show()

1. **Write And Application To Stimulate Supervised And Un-Supervised Learning Model.**

**SUPERVISED LEARNING**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from sklearn.linear\_model import LogisticRegression

from sklearn import datasets

#importing dataset

dataset = pd.read\_csv("/content/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 logistics regression to the training set

classifier = LogisticRegression()

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()

**UNSUPERVISED LEARNING**

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder

iris\_data = pd.read\_csv('/content/iris.csv')

iris\_data.shape

iris\_data.head()

df = pd.read\_csv('/content/iris.csv')

df['variety'].unique()

# Import label encoder

from sklearn import preprocessing

# label\_encoder object knows how to understand word labels.

label\_encoder = preprocessing.LabelEncoder()

# Encode labels in column 'species'.

df['variety']= label\_encoder.fit\_transform(df['variety'])

df['variety'].unique()

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

import scipy.cluster.hierarchy as shc

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

plt.title("Variety")

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

from sklearn.cluster import AgglomerativeClustering

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

cluster.fit\_predict(df)

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

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

plt.show()

1. **Write An Application to Implement Clustering Algorithm.**

# 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()

1. **Write a Program to Implement BFS Algorithm.**

import collections

# BFS algorithm

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)

1. **Write a Program to Implement 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, '1')

1. **Design A Fuzzy Based Application Using Python.**

pip install -U scikit-fuzzy

import numpy as np

import skfuzzy as fuzz

from skfuzzy import control as ctrl

# New Antecedent/Consequent objects hold universe variables and me

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)

1. **Implement Bayes Theorem Using Python.**

def drug\_user(

        prob\_th=0.8,

        sensitivity=0.79,

        specificity=0.79,

        prevelance=0.02,

        verbose=True):

#Computes the posterior using Bayes' rule

    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

print("Govind Saini")

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))

1. **Design A Bot Using AIML.**

**PRACT.PY**

import aiml

kernel = aiml.Kernel()

kernel.learn("std2-startup.xml")

kernel.respond("load prac 2")

while True:

    input\_text = input(" > Human: ")

    response = kernel.respond(input\_text)

    print(" > Bot: "+response)

**std2-startup.xml**

<aiml version="1.0.1" encoding="UTF-8">

    <category>

        <pattern>LOAD PRAC 2</pattern>

        <template>

            <learn>pract2\_chat.aiml</learn>

        </template>

    </category>

</aiml>

**pract2\_chat.aiml**

<aiml>

<category>

<pattern>MONDAY</pattern>

<template>the day of the week before Tuesday and following Sunday</template>

</category>

<category>

<pattern>TUESDAY</pattern>

<template>the day of the week before Wednesday and following Monday</template>

</category>

<category>

<pattern>WEDNESDAY</pattern>

<template>the day of the week before Thursday and following Tuesday</template>

</category>

</template>

</category>

<category>

<pattern>THURSDAY</pattern>

<template>the day of the week before Friday and following Wednesday</template>

</category>

<category>

<pattern>FRIDAY</pattern>

<template>the day of the week before Saturday and following Thursday</template

</category>

<category>

<pattern>SATURDAY</pattern>

<template>the day of the week before Sunday and following Friday. </template>

</category>

</aiml>

1. **Design An Expert System Using AIML.**

**pract.py**

import aiml

kernel = aiml.Kernel()

kernel.learn("std-startup.xml")

kernel.respond("load aiml b")

while True:

    input\_text = input("> Human: ")

    response = kernel.respond(input\_text)

    print("> Bot: "+response)

**std-startup.xml**

<aiml version="1.0.1" encoding="UTF-8">

    <category>

        <pattern>LOAD ATML B</pattern>

        <template>

            <learn>basic\_chat.aiml</learn>

        </template>

    </category>

</aiml>

**basic\_chat.aiml**

<aiml version="1.0.1" encoding="UTF-8">

    <!-- basic\_chat.aiml -->

    <category>

        <pattern>HELLO \*</pattern>

        <template>Well, hello govind!</template>

    </category>

    <category>

        <pattern>WHAT ARE YOU</pattern>

        <template>I'm a bot, and I'm silly! </template>

    </category>

    <category>

        <pattern>WHAT DO YOU DO</pattern>

        <template>I'm here to annoy you!</template>

    </category>

    <category>

        <pattern>WHO I AM</pattern>

        <template>You are Govind Saini, and you working on Web Developer...</template>

    </category>

</aiml>

1. **A Program to Implement Rule Based System.**

man (lou).

man (pete).

man (ian).

man (peter).

woman (pauline).

woman (cathy).

woman (lucy).

parent (ian, lucy).

parent (ian, peter).

parent (cathy, ian).

parent (pete, ian).

parent (lou, pete).

parent (lou, pauline).

mother (X, Y) :- woman (x), parent (X, Y), (X\=Y).

father (X, Y) :- man (X), parent (X, Y), (X\=Y).

sibling (X, Y) - parent (Z, X), parent (Z, Y), (X\=Y).

brother (X, Y):- man (X), sibling (X, Y), (X\=Y).

sister (X, Y) :- woman (X), sibling (X, Y), (X\=Y).

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

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

ancestor (X, Y) :- parent (X, Y), (X\=Y).

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