Practical No: 1
Design an Expert system.

Aim: An Expert system for responding the patient query for identifying the flu.

Program #1 Code:

```
info=[]
name=input("Enter your name: ") info.append(name)
age=int(input("Enter your age: ")) info.append(age)
print("----------------------------------------------") a=["Fever",
"Headache", "Tiredness", "Vomitting"]
b=["Urinate a lot", "Feels thirsty", "Weight loss", "Blurry vision", "Feels very hungry",
"Feels very tired"]
print("----------------------------------------------") print(a,
b)
symp=input("Enter symptoms as above separated by comm ")
lst=symp.split(",") print(info) print("Symptoms: ") for i in lst:
print(i) if i.strip() in a:     print("You May Have
Malaria\n...visit a Doctor") elif i.strip() in b:
   print("You May Have Diabetes\n...Consume less Sugar") else:
   print("Symptoms does not Match")
```

Output:

```
========== RESTART: C:/Users/admin/Downloads/DetectingFeverOrNot_Prac1-1.py ==========
Enter your name: Aditi
Enter your age: 22
----------------------------------------------
['Fever', 'Headache', 'Tiredness', 'Vomitting'] ['Urinate a lot', 'Feels thirsty', 'Weight loss', 'Blurry vision', 'Feels very hungry', 'Feels very tired']
Enter symptoms as above separated by comma: Headache, Weight loss, Feel very hungry, Vomitting
['Aditi', 22]
----------------------------------------------
Symptoms:
Headache
 Weight loss
 Feel very hungry
 Vomitting
You May Have Malaria
...visit a Doctor
>>>
```

Program #2 Code:

```
name=input("Enter your name: ")
fever=input("Do you have fever? (yes/no) ").lower() cough=input("Do you have
cough? (yes/no) ").lower() sob=input("Do you have shortness of breath? (yes/no)
").lower() st=input("Do you have sore throat? (yes/no) ").lower() mp=input("Do you
have muscle pain? (yes/no) ").lower() hc=input("Do you have headache? (yes/no)
```

```
").lower()  diarrhea=input("Do you have diarrhea? (yes/no) ").lower()
conjuctivitis=input("Do you have conjuctivitis? (yes/no) ").lower() lot=input("Do you
have Loss of Taste? (yes/no) ").lower() cp=input("Do you have Chest pain or
Pressure? (yes/no) ").lower() lsp=input("Do you have Loss of Speech or Movement?
(yes/no) ").lower()  if fever=="yes" and cough=="yes" and sob=="yes" and st=="yes"
and mp=="yes" and hc=="yes":
   print(name+" YOU HAVE FLU...")     med=input("Aditi!, would you like to look
at same medicine for the flu? (yes/no):
").lower()     if
med=="yes":
      print("Disclaimer: Contact a doctor for better guidance.")
      print("There are four FDA-approved antiviral drugs recommended by CDC to treat flu
this season: ")
      print("1. Oseltamivir phosphate")
print("2. Zanamivir")        print("3.
Peramivir")
      print("4. Baloxavir marboxil")
elif(diarrhea=="yes" and st=="yes" and fever=="yes" and cough=="yes" and
conjuctivitis=="yes" and lot=="yes"):     print(name+" YOU HAVE Corona")
med=input("Aditi!, would you like to look at some remedies for Corona? (yes/no):
").lower()     if
med=="yes":
      print("TAKE VACCINE AND QUARANTINE") elif
fever=="yes" and cough=="yes":
   print(name+" YOU HAVE COMMON CODE")     med=input("Aditi!, would you
like to look at some remedies for Corona? (yes/no):
").lower()     if
med=="yes":
     print("Disclaimer: Contact a doctor for better guidance")
     print("Treatment consists of abti-inflammatories and decongestants. Most people
d=recover on their own. ")
```

```
    print("1. Nonsteroidal abti-inflammatory drug")
print("2. Analgesic")      print("3. Antihistamine")
print("4. Cough medicine")      print("5.
Decongestant") else:
   print("Unable to identify")
```

Output:

```
== RESTART: C:/Users/admin/Downloads/DetectingFluUsingExpertSystem_Prac1-2.py ==
Enter your name: Aditi
Do you have fever? (yes/no) yes
Do you have cough? (yes/no) yes
Do you have shortness of breath? (yes/no) yes
Do you have sore throat? (yes/no) yes
Do you have muscle pain? (yes/no) yes
Do you have headache? (yes/no) yes
Do you have diarrhea? (yes/no) yes
Do you have conjuctivitis? (yes/no) yes
Do you have Loss of Taste? (yes/no) yes
Do you have Chest pain or Pressure? (yes/no) yes
Do you have Loss of Speech or Movement? (yes/no) yes
Aditi YOU HAVE FLU...
Aditi!, would you like to look at same medicine for the flu? (yes/no): yes
Disclaimer: Contact a doctor for better guidance.
There are four FDA-approved antiviral drugs recommended by CDC to treat flu this season:
1. Oseltamivir phosphate
2. Zanamivir
3. Peramivir
4. Baloxavir marboxil
>>>
```

Practical No: 2
AI bot.

Aim: Design a bot using AIML.

Code:

Open cmd and install pip –

pip install aiml pip

install python-aiml

basic_chat.aiml

```
<aiml version="1.0.1" encoding="UTF-8">
<!-- basic_chat.aiml -->

    <category>
        <pattern>HELLO *</pattern>
        <template>
            Well, Hello PCS!
        </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 motivate you!
    </template>
  </category>

  <category>
    <pattern>WHO AM I</pattern>
    <template>
      You are a Professional Footballer....
    </template>
  </category>

</aiml>
```

std-startup.xml

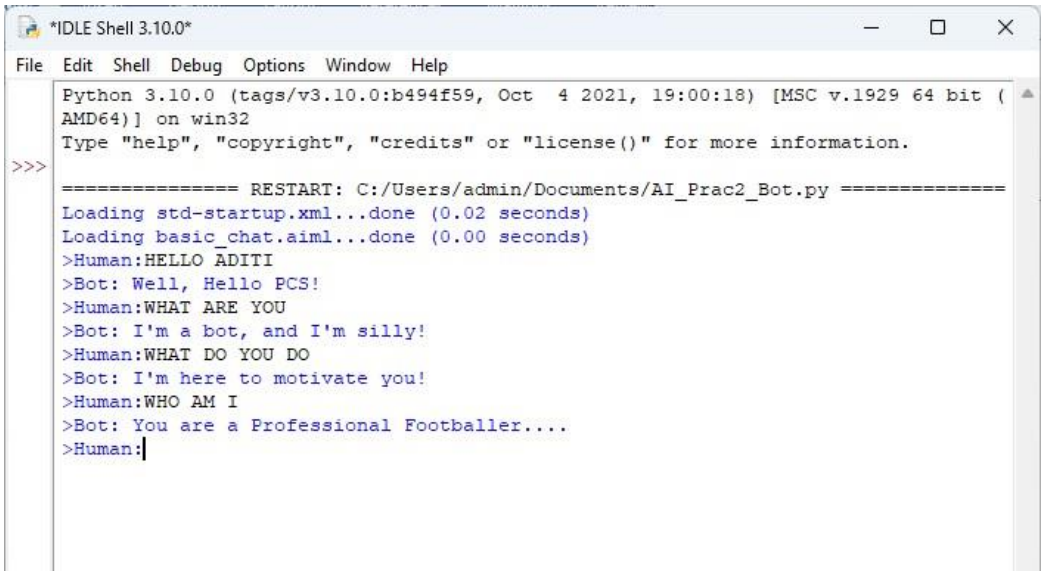
```
<aiml version="1.0.1" encoding="UTF-8">
<!-- std-startup.xml -->
<!-- Category is an atomic AIML unit -->
<category>
<!-- Pattern to match in user input -->
<!-- If user enters "LOAD AIML B" -->
<pattern>LOAD AIML B</pattern>
<!-- Template is the response to the pattern -->
<!-- This learn an aiml file -->
<template>
<learn>basic_chat.aiml</learn>
```

```
<!--  You can add more aiml files here  -->
<!-- <learn>more_aiml.aiml</learn> -->
</template>
</category>
</aiml>
```

AI_Prac2_Bot.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)
```

Output:-



Practical No: 3
Bayes Theorem.

Aim: A- Suppose we are given the probability of Mike has a cold as 0.25, the probability of Mike was observed sneezing when he had cold in the past was 0.9 and the probability of Mike was observed sneezing when he did not have cold as 0.20. Find the probability of Mike having a cold given that he sneezes.

Code:

```
def bayes_theorem(p_h, p_e_given_h, p_e_given_not_h):
not_h= 1 - p_h
    p_e= p_e_given_h * p_h + p_e_given_not_h * not_h
    p_h_given_e= (p_e_given_h * p_h)/p_e      return
p_h_given_e
p_h=float(input("Enter probability of hk having cold: "))
p_e_given_h=float(input("Enter probability of hk observed sneezing when he had cold: "))
p_e_given_not_h=float(input("Enter probability of hk observed sneezing when he did not have cold: "))
result=bayes_theorem(p_h, p_e_given_h, p_e_given_not_h)
print("Hk probability of having cold given that he sneezes is P(H|E)= ", round(result, 2))
```

Output:

```
>>>
=========== RESTART: C:/Users/admin/Downloads/BayesTheorem_Prac-3.py ===========
Enter probability of hk having cold: 0.2
Enter probability of hk observed sneezing when he had cold: 0.75
Enter probability of hk observed sneezing when he did not have cold: 0.1
Hk probability of having cold given that he sneezes is P(H|E)=  0.65
>>>
```

Aim: B-Suppose that a test for using a particular drug is 97% sensitive and 95% specific. That is, the test will produce 97% true positive results for drug users and 95% true negative results for non-drug users. These are the pieces of data that any screening test will have from their history of tests. Bayes' rule allows us to use this kind of data-driven knowledge to calculate the final probability. Suppose, we also know that 0.5% of the general population are users of the drug. What is the probability that a randomly selected individual with a positive test is a drug user?

Description:

$$P(\text{User}|+) = \frac{P(+|\text{User}).P(\text{User})}{P(+)} = \frac{P(+|\text{User}).P(\text{User})}{P(+|\text{User}).P(\text{User}) + P(+|\text{Non-user}).P(\text{Non-user})}$$

Here,

$P(\text{User})$ = Prevelance rate
$P(\text{Non-user})$ = 1−Prevelance rate
$P(+|\text{User})$ = Sensitivity
$P(-|\text{Non-user})$ = Specificity
$P(+|\text{Non-user})$ = 1−Specificity

Code:

```
def drug_user(prob_th=0.5, sensitivity=0.97, specificity=0.95, prevelance=0.005,
verbose=True):
    p_user=prevelance     p_non_user=1-
prevelance     p_pos_user=sensitivity
p_neg_user=1-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
    print("Probability of the test-taker being a drug user is ", round(prob, 1))
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 drug_user()
```

Output:

```
>>>
========== RESTART: C:/Users/admin/Downloads/BayesTheorem_DrugUser.py ==========
Probability of the test-taker being a drug user is  0.1
The test-taker may not be an user
```

Mohammed Zahid Wadiwale 22306A1053
Vidyalankar School of Information Technology

Practical No: 4
BFS & DFS.

Aim: Design a program to implement Breadth First Search (BFS).

BFS:

Code:

```
graph = {  '5' : ['3','7'],   '3' : ['2', '4'],   '7' : ['8'],   '2' : [],   '4' : ['8'],   '8' : [] }
visited = [] queue = []   def bfs(visited, graph, node):
  visited.append(node)   queue.append(node)
while queue:              m = queue.pop(0)
print (m, end = " ")      for neighbour in
graph[m]:       if neighbour not in visited:
visited.append(neighbour)
queue.append(neighbour) print("Following is
the Breadth-First Search") bfs(visited, graph,
'5')
```

Output:

```
>>>
========================================= RESTART: C:/Users/Aditi/O
Following is the Breadth-First Search
5 3 7 2 4 8
>>>
```

Aim: Design a program to implement Depth First Search(BFS).

DFS:

Code:

```
graph = {
  '5' : ['3','7'],   '3' : ['2', '4'],   '7' : ['8'],   '2' : [],   '4' : ['8'],   '8' : [] }
visited = set()  def dfs(visited, graph, node):       if node not in
visited:         print (node)         visited.add(node)         for
neighbour in graph[node]:             dfs(visited, graph, neighbour)
print("Following is the Depth-First Search") dfs(visited, graph,
'5')
```

Output:

```
>>>
========================================= RESTART: C:/Users/Aditi/OneI
Following is the Depth-First Search
5
3
2
4
8
7
>>>
```

Practical No: 5
Family Tree

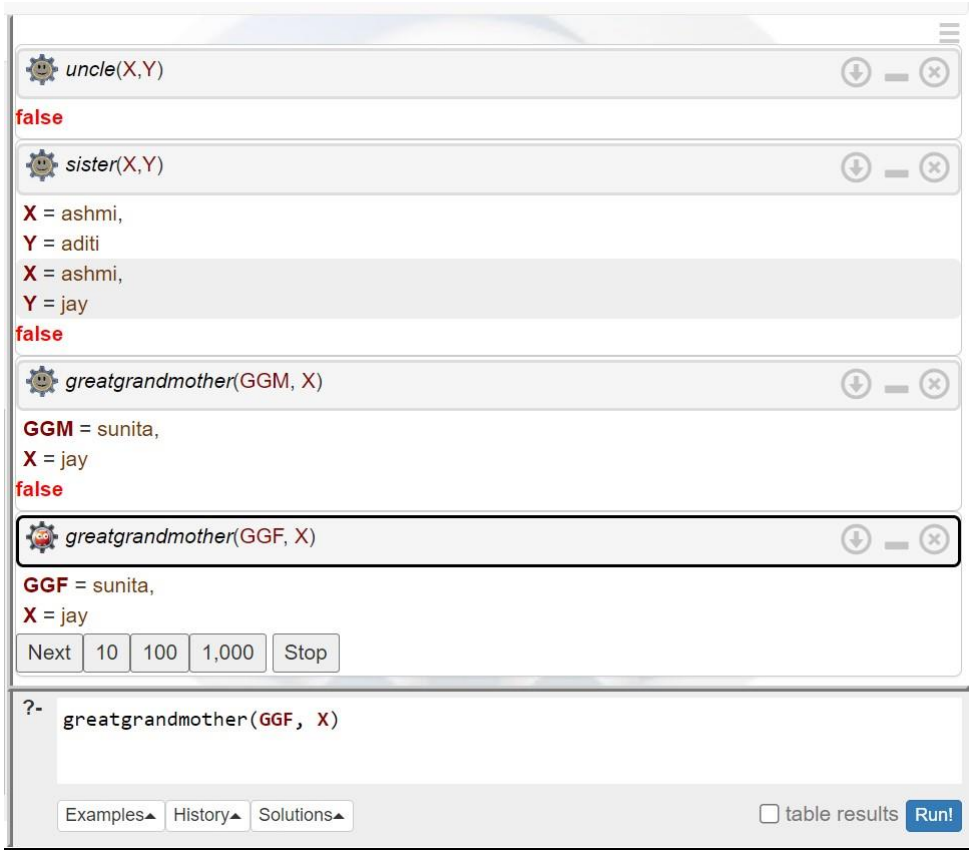
Aim: Write a program to implement Rule Based System.

Code:

```
male(kishor).    %father male(vasant).
%grandfather male(shankar).
%greatgrandfather
male(jay).       %brother male(ajit).
%uncle male(aditi).       %me
female(seema).       %mother
female(sunita).       %grandmother
female(parvati).    %greatgrandmother
female(ashmi).    %sister female(jui).
%cousin female(akshta).       %aunt
parent(kishor,aditi).
parent(seema,aditi).
parent(kishor,mudirka).
parent(seema,ashmi).
parent(kishor,jay).    parent(aditi,ajit).
parent(ajit,jay).    parent(seema,jay).
parent(vasant,kishor).
parent(sunita,kishor).
parent(shankar,vasant).
parent(parvati,prahulal).
mother(X,Y):-parent(X,Y),female(X).
father(X,Y):-parent(X,Y),    male(X).
sister(X,Y):-female(X),father(F,    Y),
father(F,X),X \= Y.  sister(X,Y):-
female(X),mother(M,          Y),
mother(M,X),X \= Y.  brother(X,Y):-
```

male(X),father(F, Y), father(F,X),X \= Y. brother(X,Y):-male(X), mother(M, Y), mother(M,X),X \= Y. grandfather(X,Y):- father(X,Z),father(Z,Y). grandmother(GM,X):-mother(GM,Y), father(Y,X). greatgrandmother(GGM,X):- mother(GGM,GM) ,parent(GM,F),parent(F,Y),parent(Y,X). greatgrandfather(GGF,X):- father(GGF,GF) ,parent(GF,F),parent(F,Y),parent(Y,X). uncle(X,Y):- father(X,Y), brother(X,Y). aunt(A,X):- parent(Y,X), sister(A,Y).

Output:


uncle(X,Y)
false
sister(X,Y)
X = ashmi,
Y = aditi
X = ashmi,
Y = jay
false
greatgrandmother(GGM, X)
GGM = sunita,
X = jay
false
greatgrandmother(GGF, X)
GGF = sunita,
X = jay
Next
10
100
1,000
Stop
?-
greatgrandmother(GGF, X)
Examples
History
Solutions
table results
Run!

Practical No: 6
Implement a Fuzzy based application

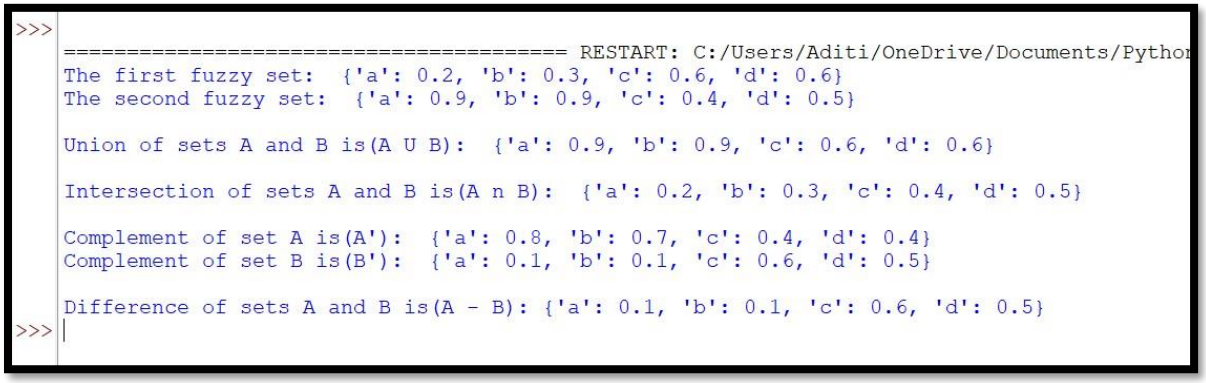
Aim: Design a fuzzy based operations using Python/ R.

Code:

```
A={"a":0.2, "b":0.3, "c":0.6, "d":0.6}

B={"a":0.9, "b":0.9, "c":0.4, "d":0.5}

print("The first fuzzy set: ", A) print("The

second fuzzy set: ", B)

#Union

result={}

for i in A:

if(A[i]>B[i]):

result[i]=A[i]     else:

        result[i]=B[i]

print("\nUnion of sets A and B is(A U B): ", result)

#Intersection result={}

for i in A:

    if(A[i]<B[i]):

result[i]=A[i]     else:

        result[i]=B[i]

print("\nIntersection of sets A and B is(A n B): ", result)

#Complement

result={}

for i in A:
    result[i]=round(1-A[i], 2) print("\nComplement

of set A is(A'): ", result)

for i in B:
```

```
    result[i]=round(1-B[i], 2) print("Complement
of set B is(B'): ", result)
#Difference result={}
for i in A:
    result[i]=round(min(A[i], 1-B[i]), 2) print("\nDifference
of sets A and B is(A - B):", result)
```

Output:

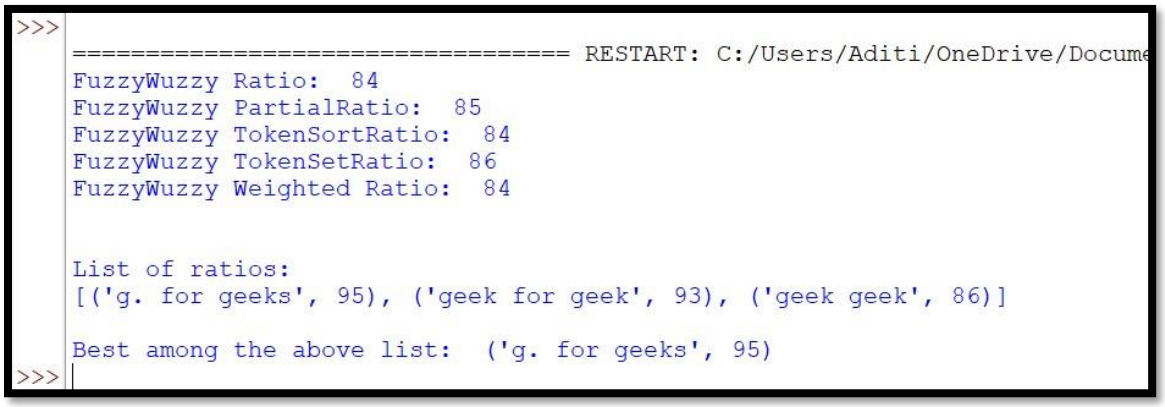


Aim: Design a Fuzzy based application using Python / R. **Code:**

```
#pip install fuzzywuzzy

from fuzzywuzzy import fuzz from
fuzzywuzzy import process

s1 = "I love GeeksforGeeks" s2 =
"I am loving GeeksforGeeks"
print("FuzzyWuzzy Ratio: ",
fuzz.ratio(s1, s2))
print("FuzzyWuzzy PartialRatio:
", fuzz.partial_ratio(s1, s2))
print("FuzzyWuzzy
TokenSortRatio: ",
fuzz.token_sort_ratio(s1, s2))
```

```
print("FuzzyWuzzy
TokenSetRatio: ",
fuzz.token_set_ratio(s1, s2))
print("FuzzyWuzzy Weighted
Ratio: ", fuzz.WRatio(s1,
s2),'\n\n')
# for process library, query
= 'geeks for geeks'
choices = ['geek for geek', 'geek geek', 'g. for geeks'] print("List
of ratios: ")
print(process.extract(query, choices), '\n')
print("Best among the above list: ",process.extractOne(query, choices))
```

Output:



Practical No: 7

Conditional Probability and Joint Probability Aim:

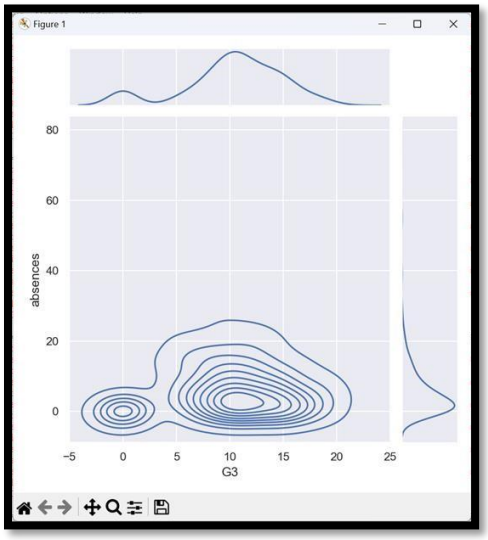
Implement joint probability using Python/ R.

Code:

```
import numpy as np import

matplotlib.pyplot as plt import

seaborn as sns import pandas

as pd sns.set()

data = pd.read_csv(r"C:\Users\Aditi\OneDrive\Documents\Python Scripts\student.csv",
usecols=['G3', 'absences']) print(data.head())

sns.jointplot(data=data, x='G3', y='absences', kind='kde') plt.show()
```

Output:

```
==============================
   absences  G3
0         6   6
1         4   6
2        10  10
3         2  15
4         4  10
```



B) Aim: Implement Conditional Probability using Python.

Code:

```
import pandas as pd import
matplotlib.pyplot as plt import
numpy as np
df = pd.read_csv(r"C:\Users\Aditi\OneDrive\Documents\Python Scripts\student.csv")
df['grade_A'] = np.where(df['G3'] * 5 >= 80, 1, 0) df['high_absences'] =
np.where(df['absences'] >= 10, 1, 0) df['count'] = 1
df = df[['grade_A', 'high_absences', 'count']] pivot_table =
pd.pivot_table(df, values='count', index=['grade_A'],
columns=['high_absences'], aggfunc=np.size, fill_value=0)
pivot_table.plot(kind='bar') plt.xlabel('Grade A')
plt.ylabel('Count')
plt.title('Grade A vs High Absences') plt.show()
```

Output:

```
>>>
= RESTART: C:/Users/Aditi/OneDrive/Documents/Python Scripts/AAI_Prac7_ConditionalProb.py
  school sex  age address famsize Pstatus  ...  Walc  health absences G1 G2  G3
0     GP   F   18       U     GT3       A  ...     1       3        6  5  6   6
1     GP   F   17       U     GT3       T  ...     1       3        4  5  5   6
2     GP   F   15       U     LE3       T  ...     3       3       10  7  8  10

[3 rows x 33 columns]
```

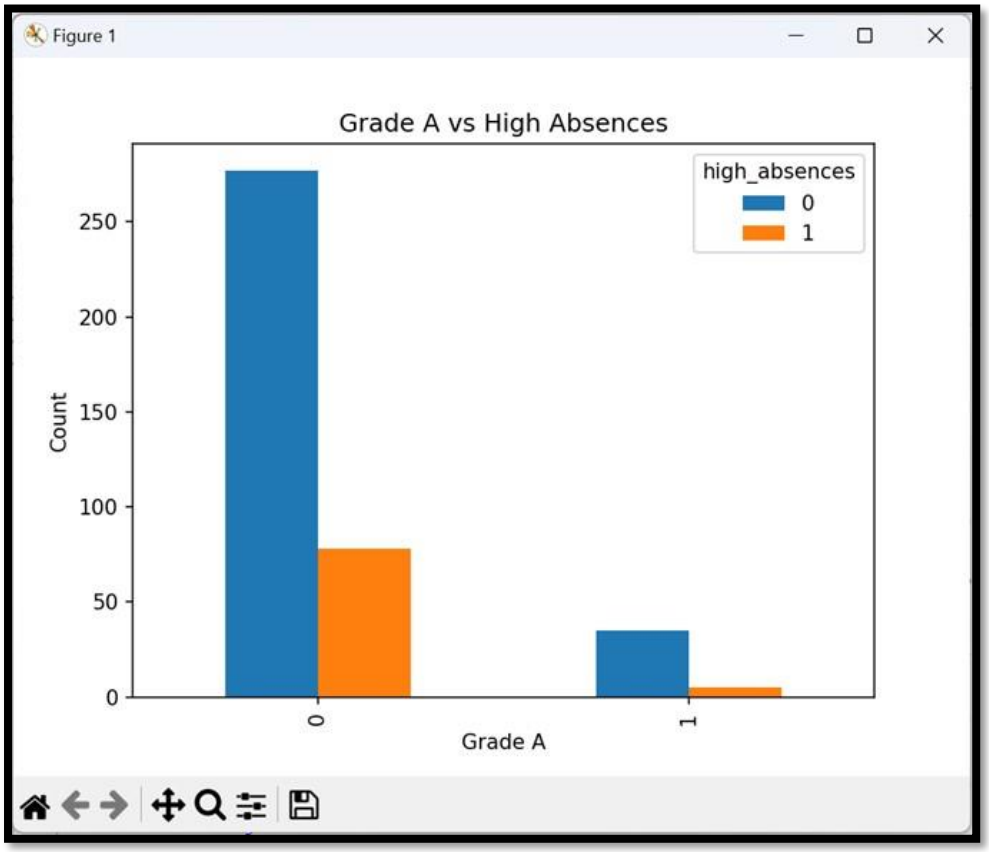


Figure 1
Grade A vs High Absences
high_absences
0
1
Count
250
200
150
100
50
0
0
1
Grade A

Mohammed Zahid Wadiwale 22306A1053
Vidyalankar School of Information Technology

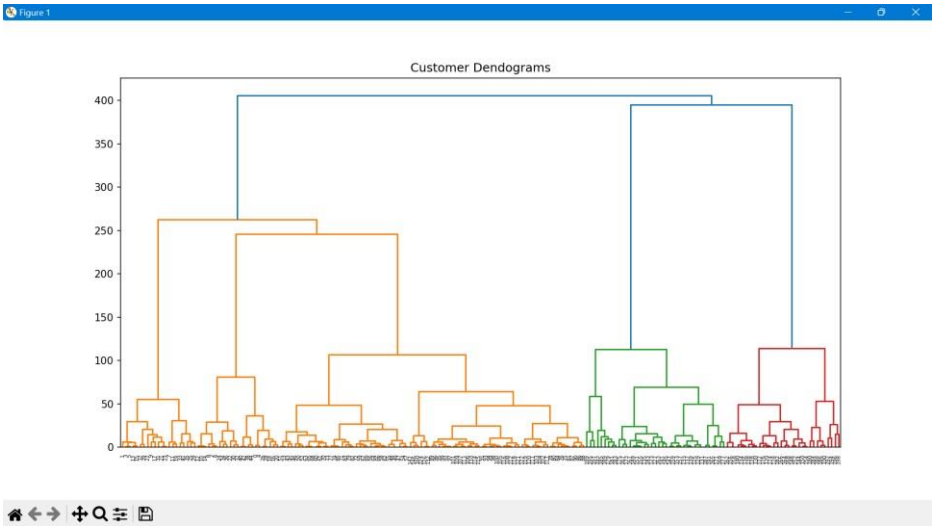
Practical No: 8
Clustering Algorithm

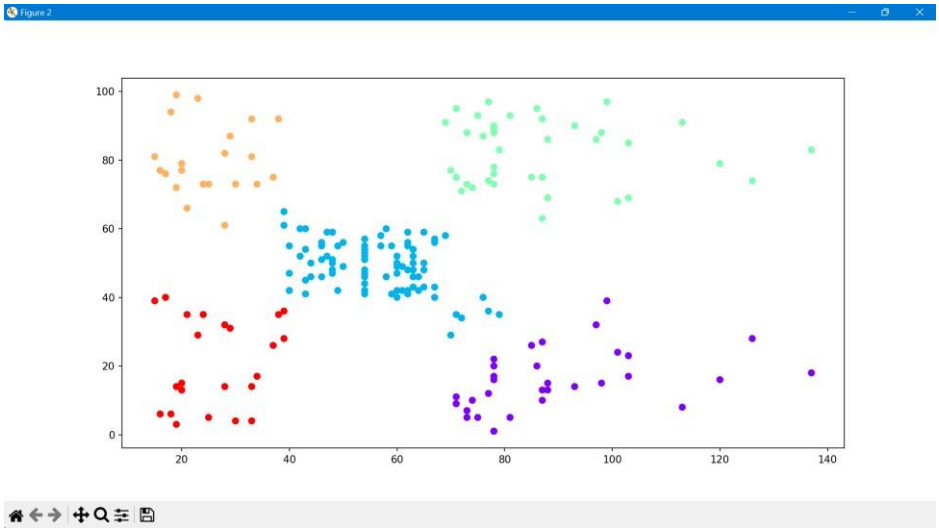
Aim: Write an application using clustering algorithm.

Code:

```
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
Dendograms")
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:





Code:

```
#Synthetic classification dataset from
numpy import where
from sklearn.datasets import make_classification from
matplotlib import pyplot
# Define datasets
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:

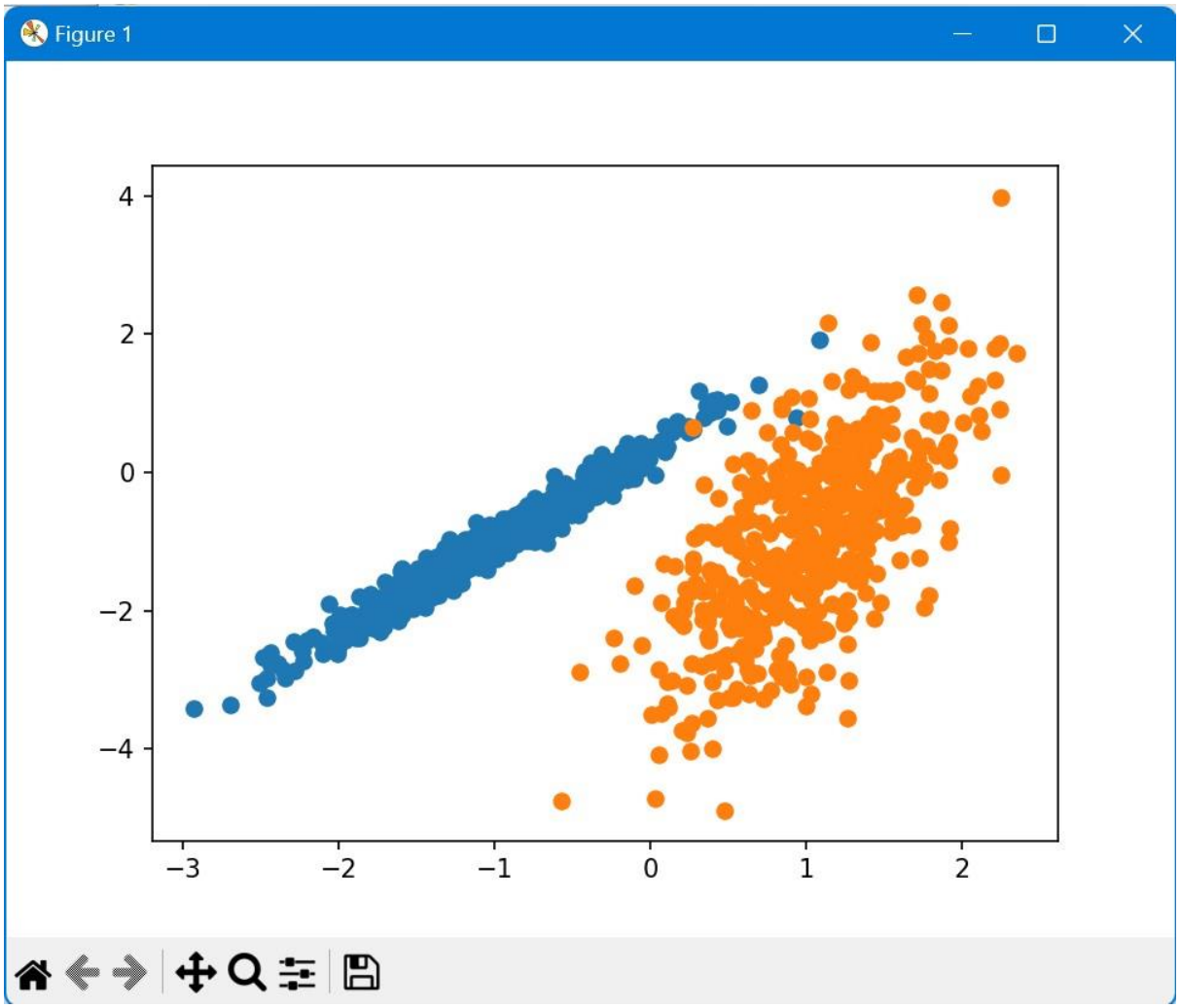


Figure 1
4
2
0
−2
−4
−3
−2
−1
0
1
2

Practical No: 9

Simulate Supervised & Unsupervised. Aim:

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

There are 11 variables using which we must predict whether a person will survive the accident or not. Use supervised learning methods of python.

Code:

```
import pandas as pd import
numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB from
sklearn.svm import LinearSVC from sklearn.svm
import SVC
from sklearn.tree import DecisionTreeClassifier from
sklearn.ensemble import RandomForestClassifier from
sklearn.metrics import accuracy_score
titanic= pd.read_csv('C:/Users/Aditi/OneDrive/Documents/Python Scripts/train.csv')
titanic.head()
titanic_cat = titanic.select_dtypes(object) titanic_num =
titanic.select_dtypes(np.number)
print("titanic_cat.head:\n",titanic_cat.head())
print("\ntitanic_num.head:\n",titanic_num.head())
titanic_cat.drop(['Name','Ticket'], axis=1, inplace=True)
titanic_cat.head() titanic_cat.isnull().sum()
titanic_cat.Cabin.fillna(titanic_cat.Cabin.value_counts().idxmax(), inplace=True)
titanic_cat.Embarked.fillna(titanic_cat.Embarked.value_counts().idxmax(), inplace=True)
titanic_cat.head(20) le = LabelEncoder()
titanic_cat = titanic_cat.apply(le.fit_transform)

titanic_cat.head() titanic_num.isna().sum()
```

```
titanic_num.Age.fillna(titanic_num.Age.mean(), inplace=True) titanic_num.isna().sum()
titanic_num.drop(['PassengerId'], axis=1, inplace=True) titanic_num.head()
titanic_final = pd.concat([titanic_cat,titanic_num],axis=1)
print("\ntitanic_final.head:\n",titanic_final.head())
X=titanic_final.drop(['Survived'],axis=1)
Y= titanic_final['Survived']
X_train = np.array(X[0:int(0.80*len(X))])
Y_train = np.array(Y[0:int(0.80*len(Y))])
X_test = np.array(X[int(0.80*len(X)):]) Y_test =
np.array(Y[int(0.80*len(Y)):]) len(X_train),
len(Y_train), len(X_test), len(Y_test)
LR = LogisticRegression()
KNN = KNeighborsClassifier()
NB = GaussianNB()
LSVM = LinearSVC()
NLSVM = SVC(kernel='rbf')
DT = DecisionTreeClassifier()
RF = RandomForestClassifier()
LR_fit = LR.fit(X_train, Y_train)
KNN_fit = KNN.fit(X_train, Y_train)
NB_fit = NB.fit(X_train, Y_train)
LSVM_fit = LSVM.fit(X_train, Y_train)
NLSVM_fit = NLSVM.fit(X_train, Y_train)
DT_fit = DT.fit(X_train, Y_train)
RF_fit = RF.fit(X_train, Y_train)
LR_pred = LR_fit.predict(X_test)
KNN_pred = KNN_fit.predict(X_test)
NB_pred = NB_fit.predict(X_test)
LSVM_pred = LSVM_fit.predict(X_test)
NLSVM_pred = NLSVM_fit.predict(X_test)
```

```
DT_pred = DT_fit.predict(X_test) RF_pred
= RF_fit.predict(X_test)
print("Logistic Regression is %f percent accurate" % (accuracy_score(LR_pred, Y_test)*100))
print("KNN is %f percent accurate" % (accuracy_score(KNN_pred, Y_test)*100))
print("Naive Bayes is %f percent accurate" % (accuracy_score(NB_pred, Y_test)*100))
print("Linear SVMs is %f percent accurate" % (accuracy_score(LSVM_pred, Y_test)*100))
print("Non Linear SVMs is %f percent accurate" % (accuracy_score(NLSVM_pred, Y_test)*100))
print("Decision Trees is %f percent accurate" % (accuracy_score(DT_pred, Y_test)*100))
print("Random Forests is %f percent accurate" % (accuracy_score(RF_pred, Y_test)*100))
```

Output:

```
>>>
================== RESTART: C:\Users\Aditi\Downloads\pr9.py ==================
titanic_cat.head:
                                                Name     Sex  ... Cabin Embarked
0                            Braund, Mr. Owen Harris    male  ...   NaN        S
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  ...   C85        C
2                             Heikkinen, Miss. Laina  female  ...   NaN        S
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  ...  C123        S
4                           Allen, Mr. William Henry    male  ...   NaN        S

[5 rows x 5 columns]

titanic_num.head:
   PassengerId  Survived  Pclass   Age  SibSp  Parch     Fare
0            1         0       3  22.0      1      0   7.2500
1            2         1       1  38.0      1      0  71.2833
2            3         1       3  26.0      0      0   7.9250
3            4         1       1  35.0      1      0  53.1000
4            5         0       3  35.0      0      0   8.0500

titanic_final.head:
   Sex  Cabin  Embarked  Survived  Pclass   Age  SibSp  Parch     Fare
0    1     47         2         0       3  22.0      1      0   7.2500
1    0     81         0         1       1  38.0      1      0  71.2833
2    0     47         2         1       3  26.0      0      0   7.9250
3    0     55         2         1       1  35.0      1      0  53.1000
4    1     47         2         0       3  35.0      0      0   8.0500
```

```
Logistic Regression is 83.798883 percent accurate
KNN is 75.977654 percent accurate
Naive Bayes is 82.681564 percent accurate
Linear SVMs is 37.988827 percent accurate
Non Linear SVMs is 74.301676 percent accurate
Decision Trees is 81.564246 percent accurate
Random Forests is 84.357542 percent accurate
>>>
```

Practical No: 10
Intelligent Agents.

Aim: Design an Artificial Intelligence application to implement intelligent agents. **Code:**

```
class ClothesAgent:
def __init__(self):
self.weather=None
def get_weather(self):
      self.weather=input("Enter the weather (Sunny, Rainy, Windy, Snowy) : ").lower()
def suggest_clothes(self):        if self.weather=="sunny":
          print("It is sunny outside. You should wear light clothes, sunglasses and sunscreen.")
elif self.weather=="rainy":
          print("It is rainy outside. Don't forget an umbrella, raincoat, and waterproof shoes.")
elif self.weather=="windy":
          print("It is windy outside. Wear layers and a jacket to stay warm.")
elif self.weather=="snowy":
          print("It is snowy outside. Dress warmly with a heavy coat, gloves, and boots.")
else:
          print("Sorry.I don't understand the weather conditions. Please enter sunny, rainy,
windy, or snowy.") def main():
   agent=ClothesAgent()
agent.get_weather()
agent.suggest_clothes()


if __name__=="__main__":
   main()
```

Output:

```
>>>
= RESTART: C:\Users\Aditi\OneDrive\Documents\Python Scripts\AAI_Prac-10-1_intelligentAgent.py
Enter the weather (Sunny, Rainy, Windy, Snowy) : Rainy
It is rainy outside. Don't forget an umbrella, raincoat, and waterproof shoes.
>>>
= RESTART: C:\Users\Aditi\OneDrive\Documents\Python Scripts\AAI_Prac-10-1_intelligentAgent.py
Enter the weather (Sunny, Rainy, Windy, Snowy) : Snowy
It is snowy outside. Dress warmly with a heavy coat, gloves, and boots.
>>>
```

Practical No: 11

Language Parser Aim:

Design an application to simulate language parser. Code & Output:

```
def sentenceSegment(text):
    sentences = []  start = 0  for i in
range(len(text)):  if text[i] == '.' or text[i] ==
'!' or text[i] == '?':
sentences.append(text[start:i+1].strip())
start = i + 1  return sentences
text = "Hello, NLP world!! In this example, we are going to do the basics of Text processing
which will be used later." print(sentenceSegment(text))
```

```
['Hello, NLP world!', '!', 'In this example, we are going to do the basics of Text processing which will be used later.']
```

```
import nltk nltk.download('punkt')
text = "Hello, NLP world!! In this example, we are going to do the basics of Text processing
which will be used later." sentences = nltk.sent_tokenize(text) print(sentences)
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
['Hello, NLP world!!', 'In this example, we are going to do the basics of Text processing which will be used later.']
```

```
import string def remove_punctuation(input_string):  punctuations = string.punctuation
    output_string = "".join(char for char in input_string if char not in punctuations)
return output_string
text = "Hello, NLP world!! In this example, we are going to do the basics of Text processing
which will be used later." sentences = sentenceSegment(text) puncRemovedText =
remove_punctuation(text) print(puncRemovedText)
```

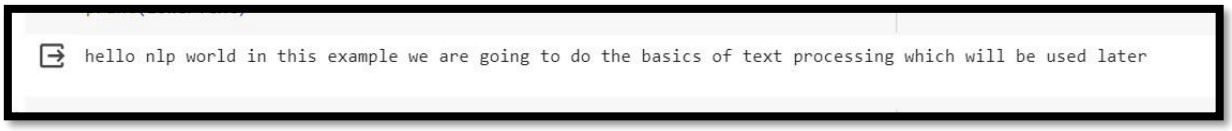
```
Hello NLP world In this example we are going to do the basics of Text processing which will be used later
```

```
def convertToLower(s):
  return s.lower()
```

text = "Hello, NLP world!! In this example, we are going to do the basics of Text processing which will be used later."

puncRemovedText = remove_punctuation(text) lowerText

= convertToLower(puncRemovedText) print(lowerText)



def tokenize(s):

words = [] #token words should be stored here

i = 0 word = "" while(i <len(s)): if (s[i] !=

" "):

word = word+s[i]

else:

words.append(word)

word = "" i = i + 1

words.append(word)

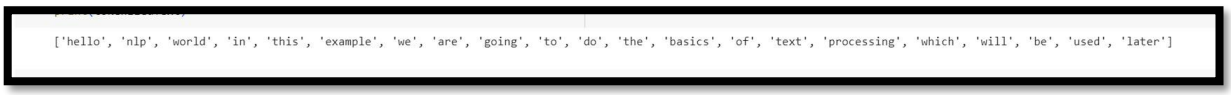
return words

text = "Hello, NLP world!! In this example, we are going to do the basics of Text processing which will be used later."

puncRemovedText = remove_punctuation(text) lowerText

= convertToLower(puncRemovedText) tokenizedText =

tokenize(lowerText) print(tokenizedText)



import nltk # Define input text

text = "Hello, NLP world!! In this example, we are going to do the basics of Text processing which will be used later."

#sentence segmentation - removal of punctuations and converting to lowercase

sentences = nltk.sent_tokenize(text) puncRemovedText =

remove_punctuation(text) lowerText = convertToLower(puncRemovedText)

tokens = nltk.word_tokenize(lowerText) print(tokens)

import nltk

sentence = "We're going to John's house today."

tokens = nltk.word_tokenize(sentence) print(tokens)