

## Practical No: 1

### Design An Expert System using AIML

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

**Code:**

```
info = []
name = input("Enter Your name: ")
info.append(name)
age = int(input("Enter Your age: "))
info.append(age)
a = ["Fever", "Headache", "Tiredness", "Vomitting"]
b = ["Urinate A Lot", "Feels Thirsty", "Weight Loss", "Blurry Vision", "Feels Very Hungry",
"Feels Very Tired"]

print(a,b)

symp = input("Enter Symptoms As Above Separated By Comma ")
lst = symp.split(",")
print(info)

print("Symptoms:")
for i in lst:
    print(i)
    if i.strip() in a:
        print("You May Have Malaria")
        print("Visit A Doctor")
    elif i.strip() in b:
        print("You May Have Diabetes")
        print("Consume Less Sugar")
    else:
        print("Symptoms Does Not Match")
```

```

In [1]: runfile('D:/NK/NK_aai_1.py', wdir='D:/NK')
Enter Your Name: Ninad
Enter Your age: 30
['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 fever,headache
['Ninad', 30]
Symptoms:
fever
headache
You May Have Malaria
Visit A Doctor

```

b)

### Code:

```

name =input("Enter your name: ")
fever =input("DO YOU HAVE fever (Y/N)").lower()
cough =input("DO YOU HAVE cough (Y/N)").lower()
sob =input("DO YOU HAVE shortness of breath (Y/N)").lower()
st =input("DO YOU HAVE sore throat (Y/N)").lower()
mp =input("DO YOU HAVE mucle pain (Y/N)").lower()
hc =input("DO YOU HAVE headache(Y/N)").lower()
#CORONA
diarrhoea=input("DO YOU HAVE diarrhoea (Y/N)").lower()
conjunctivitis=input("DO YOU HAVE conjunctivitis (Y/N)").lower()
lot=input("DO YOU HAVE Loss OF taste (Y/N)").lower()
cp=input("DO YOU HAVE chest pain or pressure (Y/N)").lower()
lsp =input("DO YOU HAVE Loss Of Speech or movement (Y/N)").lower()
if fever=="y" and cough=="y" and sob=="y" and st=="y" and mp=="y" and hc=="y":
    print(name+" "+" YOU HAVE FLU")
    med=input("Sir/Ma'am would you like to took at some medicine for flu(Y/N)").lower()
    if med=="y":
        print("disclainer contact doctor for better guidance")
        print("There are four FDA-approved antiviral drugs recommended by CDC to treat flu
this season")
        print("1.Oseltasivir phosphate")
        print("2.zonasivir ")
        print("3.perasivir ")
        print("4.balaxavir morboxil ")
    elif diarrhoea=="y" and st=="y" and fever == "y" and cough=="y" and conjunctivitis=="y"
and lot=="y":
        print(name+" "+" YOU HAVE Corona")
        med=input("Sir/Ma'am would you like to take at some remedi for Corona(Y/N)").lower()
        if med=="y":
            print("TAKE VACCINE AND QUARANTINE")
    elif fever=="y" and cough=="y":

```

```
print(name+" "+" YOU HAVE Common Cold")
med= input("Sir/Ma'am would you like to take at some remedi for common
cold(Y/N)").lower()
if med=="y":
    print("-----")
    print("disclainer contact doctor for better guidance")
    print("-----")
    print("Treatment consists of anti-inflammatories and decongestants\n Most prople
recover on their own")
    print("1.Nonsteroidal anti-inflammatory drug, Analgesic, Antibistamine, Cough
medicine and Deconges")
else:
    print("Unable to identify")
```

```
In [2]: runfile('D:/NK/NK_aai_1-2.py', wdir='D:/NK')
Enter your name: Ninad
DO YOU HAVE fever (Y/N)y
DO YOU HAVE cough (Y/N)y
DO YOU HAVE shortness of breath (Y/N)y
DO YOU HAVE sore throat (Y/N)y
DO YOU HAVE mucle pain (Y/N)y
DO YOU HAVE headache(Y/N)y
DO YOU HAVE diarrhoea (Y/N)y
DO YOU HAVE conjunctivitis (Y/N)y
DO YOU HAVE Loss OF taste (Y/N)n
DO YOU HAVE chest pain or pressure (Y/N)y
DO YOU HAVE Loss Of Speech or movement (Y/N)y
Ninad YOU HAVE FLU
Sir/Ma'am would you like to took at some medicine for flu(Y/
N)y
disclainer contact doctor for better guidance
There are four FDA-approved antiviral drugs recommended by
CDC to treat flu this season
1.Oseltasivir phosphate
2.zonasivir
3.perasivir
4.balaxavir morboxil

In [3]:
```



## Practical No: 2

### Chatbot

#### AIM: Design a Chatbot using AIML

#### Code:

#### Open cmd and install pip –

```
pip install aiml  
pip install python-aiml
```

#### Prac2.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)
```

#### Basic\_chat.aiml

```
<aiml version="1.0.1" encoding="UTF-8">  
<!-- basic_chat.aiml -->  
    <category>  
        <pattern>HELLO *</pattern>  
        <template>  
            Well, Hello Ninad!  
        </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>
```

### Output:-

```
===== RESTART: D:\NK\pract2.py =====
Loading std-startup.xml...done (0.02 seconds)
Loading basic_chat.aiml...done (0.00 seconds)
>Human: hello bot
>Bot: Well, Hello Ninad!
>Human: what are you
>Bot: I'm a bot, and I'm silly!
>Human: who am i
>Bot: You are a Professional Footballer....
>Human: what do you do
>Bot: I'm here to motivate you!
>Human:
```

## Practical No: 3

### Implement Bayes Theorem using Python

**AIM:** 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):  
    p_not_h = 1 - p_h  
    p_e = (p_e_given_h * p_h) + (p_e_given_not_h * p_not_h)  
    p_h_given_e = (p_e_given_h * p_h) / p_e  
    return p_h_given_e  
p_h = float(input("Enter the probability of NK having a cold: "))  
p_e_given_h = float(input("Enter the probability of observing sneezing when NK has a cold:  
"))  
p_e_given_not_h = float(input("Enter the probability of observing sneezing when NK does  
not have a cold: "))  
result = bayes_theorem(p_h, p_e_given_h, p_e_given_not_h)  
print("NK's probability of having a cold given that he sneezes (P(H|E)) is:", round(result, 2))
```

#### OUTPUT

```
===== RESTART: D:/NK/AAI/aai_3_1.py =====  
Enter the probability of NK having a cold: 0.25  
Enter the probability of observing sneezing when NK has a cold: 0.9  
Enter the probability of observing sneezing when NK does not have a cold: 0.20  
NK's probability of having a cold given that he sneezes (P(H|E)) is: 0.6  
|
```

**b) AIM:** 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?

### Code:

```
def
drug_user(prob_th=0.5,sensitivity=0.97,specificity=0.95,prevelance=0.005,verbose=True):
#FORMULA
    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
    print("Probability of the NK being a drug user is", round(prob,3))
    if verbose:
        if prob > prob_th:
            print("The NK could be an user")
        else:
            print("The NK may not be an user")
    return prob
drug_user()
```

### OUTPUT:

```
===== RESTART: D:/NK/AAI/aai_3_2.py :
Probability of the NK being a drug user is 0.089
The NK may not be an user
```



## Practical No: 4

### Implement DFS and BFS algorithm

a) **AIM:** Write an application to implement DFS algorithm.

#### Code:

```
graph = {
    '5' : ['3','7'],
    '3' : ['2', '4'],
    '7' : ['8'],
    '2' : [],
    '4' : ['8'],
    '8' : []
}

visited = [] # List for visited nodes.
queue = []   #Initialize a queue

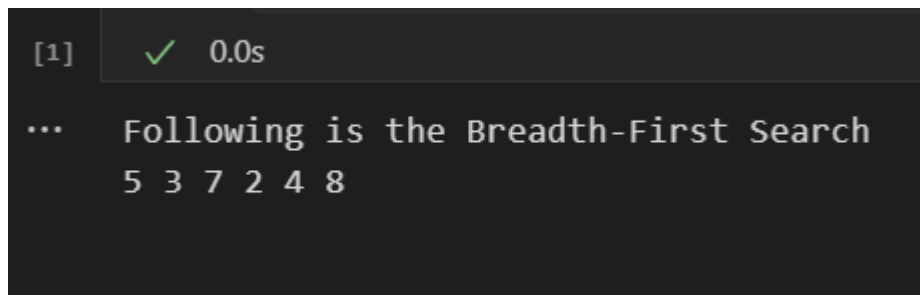
def bfs(visited, graph, node): #function for BFS
    visited.append(node)
    queue.append(node)

    while queue:          # Creating loop to visit each node
        m = queue.pop(0)
        print (m, end = " ")

        for neighbour in graph[m]:
            if neighbour not in visited:
                visited.append(neighbour)
                queue.append(neighbour)

# Driver Code
print("Following is the Breadth-First Search")
bfs(visited, graph, '5')# function calling
```

#### OUTPUT:



```
[1] ✓ 0.0s

... Following is the Breadth-First Search
5 3 7 2 4 8
```

**b) Aim: Write an application to implement BFS algorithm.****Code:**

```
# Using a Python dictionary to act as an adjacency list
```

```
graph = {  
    '5' : ['3','7'],  
    '3' : ['2', '4'],  
    '7' : ['8'],  
    '2' : [],  
    '4' : ['8'],  
    '8' : []  
}
```

```
visited = set() # Set to keep track of visited nodes of graph.
```

```
def dfs(visited, graph, node): #function for dfs
```

```
    if node not in visited:
```

```
        print (node)
```

```
        visited.add(node)
```

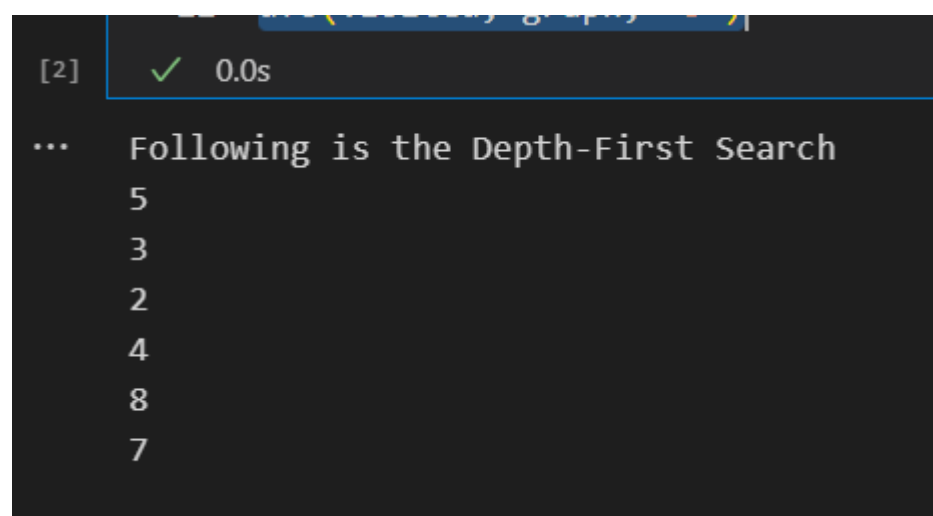
```
        for neighbour in graph[node]:
```

```
            dfs(visited, graph, neighbour)
```

```
# Driver Code
```

```
print("Following is the Depth-First Search")
```

```
dfs(visited, graph, '5')
```

**OUTPUT:**

```
[2] ✓ 0.0s  
... Following is the Depth-First Search  
    5  
    3  
    2  
    4  
    8  
    7
```

## Practical No: 5

### A program to implement Rule Based System.

**AIM:** Write a program which contains three predicates: male, female, parent. Make rules for following family relations: father, mother, grandfather, grandmother, brother, sister, uncle, aunt, nephew and niece, cousin.

#### Code:

```
male(vijay).
male(mahadev).
male(gaurihar).
male(omkar).
male(bajrang).
male(chaitanya).
```

```
female(vasanti).
female(indubai).
female(ashwini).
female(gayatri).
female(sangita).
```

```
parent(vijay,chaitanya).
parent(vasanti,chaitanya).
parent(vijay,gaurihar).
parent(vasanti,gaurihar).
parent(vijay,ashwini).
parent(vasanti,ashwini).
parent(mahadev,vijay).
parent(indubai,vijay).
```

```
mother(X,Y):-parent(X,Y),female(X).
father(X,Y):- parent(X,Y), male(X).
```

```
grandmother(GM,X):- mother(GM,Y) ,parent(Y,X).
grandfather(GF,X):- father(GF,Y) ,parent(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).
```

```
sibling(X,Y):-mother(M,X), mother(M,Y),X\=Y, father(F,X), father(F,Y).
brother(X,Y):-sibling(X,Y), male(X).
sister(X,Y):-sibling(X,Y), female(X).
uncle(U,X):- parent(Y,X), brother(U,Y).
aunt(A,X):- parent(Y,X), sister(A,Y).
nephew(N,X):- sibling(S,X),parent(S,N),male(N).
niece(N,X):-sibling(S,X), parent(S,N), female(N).
```

cousin(X,Y):-parent(P,Y),sibling(S,P),parent(S,X).

## OUTPUT:



 *father(X,Y).*

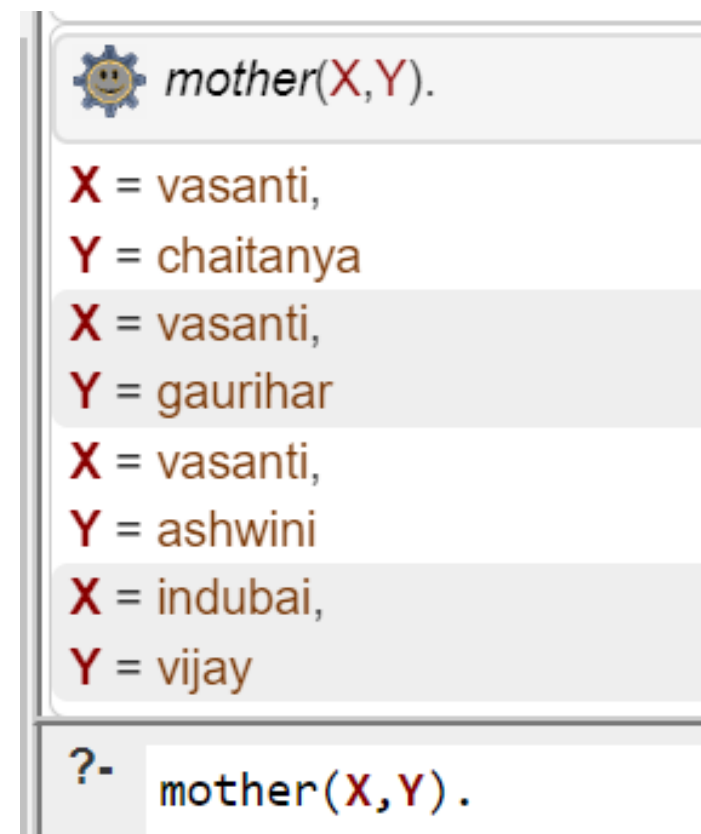
**X** = vijay,  
**Y** = chaitanya


**X** = vijay,  
**Y** = gaurihar

**X** = vijay,  
**Y** = ashwini

**X** = mahadev,  
**Y** = vijay

?- *father(X,Y).*



 *mother(X,Y).*

**X** = vasanti,  
**Y** = chaitanya

**X** = vasanti,  
**Y** = gaurihar

**X** = vasanti,  
**Y** = ashwini

**X** = indubai,  
**Y** = vijay

?- *mother(X,Y).*



 *sibling(X,Y).*

**X** = chaitanya,  
**Y** = gaurihar

**X** = chaitanya,  
**Y** = ashwini

**X** = gaurihar,  
**Y** = chaitanya

**X** = gaurihar,  
**Y** = ashwini

**X** = ashwini,  
**Y** = chaitanya

**X** = ashwini,  
**Y** = gaurihar

?- *sibling(X,Y).*

**b)**

**Code:**

```
/* Facts */
male(jack).
male(oliver).
male(ali).
male(james).
male(simon).
male(harry).
female(helen).
female(sophie).
female(jess).
female(lily).

parent_of(jack, jess).
parent_of(jack, lily).
parent_of(helen, jess).
parent_of(helen, lily).
parent_of(oliver, james).
parent_of(sophie, james).
parent_of(jess, simon).
parent_of(ali, simon).
parent_of(lily, harry).
parent_of(james, harry).

/* Rules */
father_of(X, Y):- male(X), parent_of(X, Y).
mother_of(X, Y):- female(X), parent_of(X, Y).
grandfather_of(X, Y):- male(X), parent_of(X, Z), parent_of(Z, Y).
grandmother_of(X, Y):- female(X), parent_of(X, Z), parent_of(Z, Y).
sister_of(X, Y):- female(X), father_of(F, Y), father_of(F, X), X \= Y.
sister_of(X, Y):- female(X), mother_of(M, Y), mother_of(M, X), X \= Y.
aunt_of(X, Y):- female(X), parent_of(Z, Y), sister_of(Z, X), !.
brother_of(X, Y):- male(X), father_of(F, Y), father_of(F, X), X \= Y.
brother_of(X, Y):- male(X), mother_of(M, Y), mother_of(M, X), X \= Y.
uncle_of(X, Y):- parent_of(Z, Y), brother_of(Z, X).
ancestor_of(X, Y):- parent_of(X, Y).
ancestor_of(X, Y):- parent_of(X, Z), ancestor_of(Z, Y).
```

**OUTPUT:**

 *parent\_of(X,Y).*

**X** = jack,  
**Y** = jess

**X** = jack,  
**Y** = lily

**X** = helen,  
**Y** = jess

**X** = helen,  
**Y** = lily

**X** = oliver,  
**Y** = james

**X** = sophie,  
**Y** = james


**X** = jess,  
**Y** = simon

**X** = ali,  
**Y** = simon

**X** = lily,  
**Y** = harry

**X** = james,  
**Y** = harry

?- *parent\_of(X,Y).*

 *mother\_of(X,Y).*

**X** = helen,  
**Y** = jess

**X** = helen,  
**Y** = lily

**X** = sophie,  
**Y** = james

**X** = jess,  
**Y** = simon

**X** = lily,  
**Y** = harry

 *father\_of(X,Y).*

**X** = jack,  
**Y** = jess

**X** = jack,  
**Y** = lily

**X** = oliver,  
**Y** = james

**X** = ali,  
**Y** = simon

**X** = james,  
**Y** = harry

## Practical No: 6

### Implement a Fuzzy based application.

**AIM: Design a Fuzzy based operations using Python / R.**

**Code:**

```
A = dict()
B = dict()
Y = dict()
# Initialize the dictionaries for fuzzy sets A, B, and the result
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}
result = {}
# Display the fuzzy sets A and B
print("The First Fuzzy Set is:", A)
print("The Second Fuzzy Set is:", B)
# Fuzzy Set Union
for i in A:
    if A[i] > B[i]:
        result[i] = A[i]
    else:
        result[i] = B[i]
print("Union of two sets is", result)
# Fuzzy Set Intersection
result = {}
for i in A:
    if A[i] < B[i]:
        result[i] = A[i]
    else:
        result[i] = B[i]
print("Intersection of two sets is", result)
# Fuzzy Set Complement
result = {}
for i in A:
    result[i] = round(1 - A[i], 2)
print("Complement of First set is", result)
# Fuzzy Set Difference
result = {}
for i in A:
    result[i] = round(min(A[i], 1 - B[i]), 2)
print("Difference of two sets is", result)
```

**Output:**

```
Ninad/MscIT/Semester 3/Applied Artificial Intelligence/Practical6/AAI_6
The First Fuzzy Set is: {'a': 0.2, 'b': 0.3, 'c': 0.6, 'd': 0.6}
The Second Fuzzy Set is: {'a': 0.9, 'b': 0.9, 'c': 0.4, 'd': 0.5}
Union of two sets is {'a': 0.9, 'b': 0.9, 'c': 0.6, 'd': 0.6}
Intersection of two sets is {'a': 0.2, 'b': 0.3, 'c': 0.4, 'd': 0.5}
Complement of First set is {'a': 0.8, 'b': 0.7, 'c': 0.4, 'd': 0.4}
Difference of two sets is {'a': 0.1, 'b': 0.1, 'c': 0.6, 'd': 0.5}
PS F:\GitHub\Practical BscIT MscIT Ninad>
```

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

```
# AAI 6B: AIM: Design a Fuzzy based application using Python / R.

# !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 WRatio: ", 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:**

```
FuzzyWuzzy Ratio: 84
FuzzyWuzzy PartialRatio: 85
FuzzyWuzzy TokenSortRatio: 84
FuzzyWuzzy TokenSetRatio: 86
FuzzyWuzzy WRatio: 84

List of ratios:
[('g. for geeks', 95), ('geek for geek', 93), ('geek geek', 86)]

Best among the above list: ('g. for geeks', 95)
PS F:\Github\Practical_BscIT_MscIT_Ninad>
```



## Practical No: 7

### Implement Conditional Probability And Joint Probability using Python.

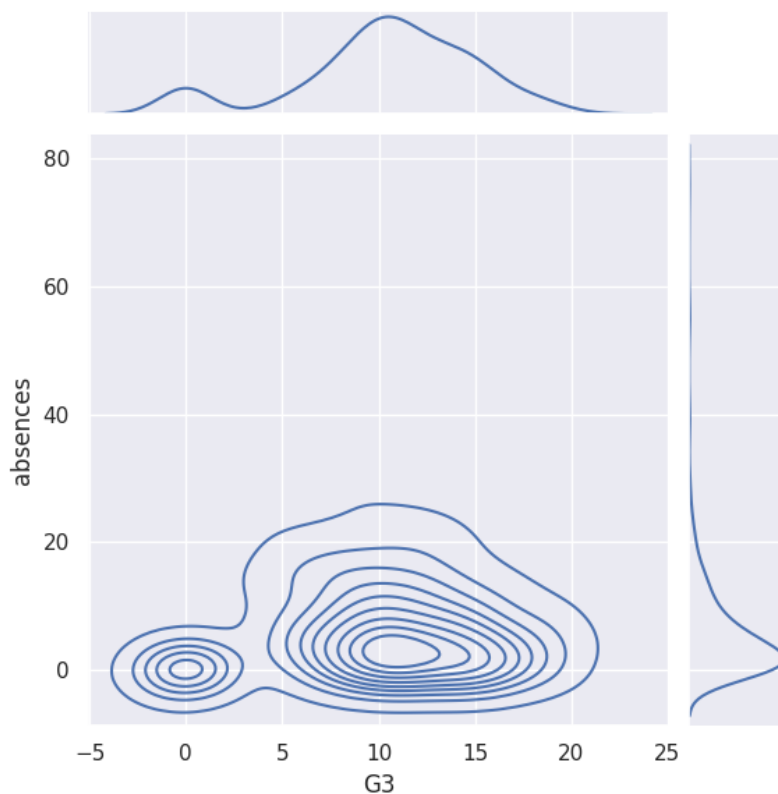
**AIM: Implement joint probability using Python.**

#### Code:

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
sns.set()
# Read the dataset
data = pd.read_csv('/content/student-mat.csv')
# Create a joint plot
sns.jointplot(data=data, x='G3', y='absences', kind='kde')

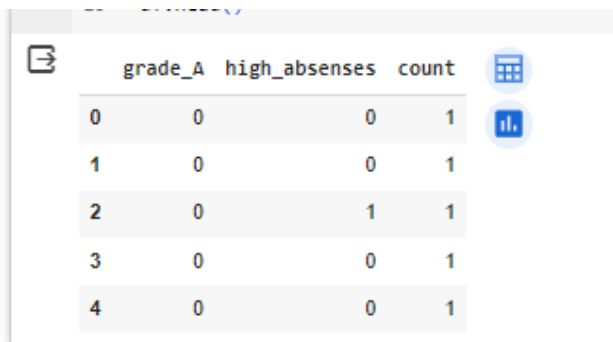
# Display the plot
plt.show()
```

#### Output:

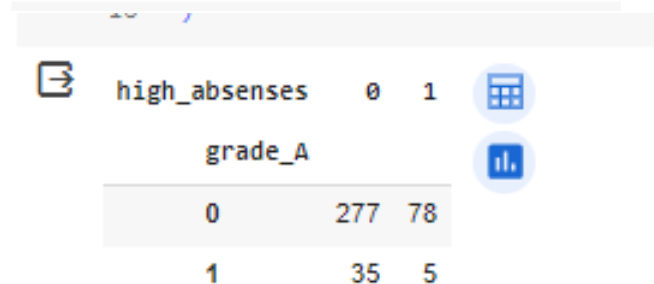


**b) AIM: Implement Conditional Probability using Python.****Code:**

```
import pandas as pd
df = pd.read_csv('/content/student-mat.csv')
df.head(3)
len(df)
import numpy as np
df['grade_A'] = np.where(df['G3']*5 >= 80, 1, 0)
df['high_absenses'] = np.where(df['absences'] >= 10, 1, 0)
df['count'] = 1
df = df[['grade_A', 'high_absenses', 'count']]
df.head()
pd.pivot_table(
    df,
    values='count',
    index=['grade_A'],
    columns=['high_absenses'],
    aggfunc=np.size,
    fill_value=0
)
```

**Output:**

	grade_A	high_absenses	count
0	0	0	1
1	0	0	1
2	0	1	1
3	0	0	1
4	0	0	1



	high_absenses	
	0	1
grade_A		
0	277	78
1	35	5

## Practical No: 8

### Clustering algorithm

**AIM: Write an application to implement clustering algorithm.**

#### **Code: Hierarchical clustering**

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import scipy.cluster.hierarchy as shc
from sklearn.cluster import AgglomerativeClustering

# Read the customer data from a CSV file
customer_data = pd.read_csv('/content/mall_customers.csv')

# Display the shape and the first few rows of the data
print(customer_data.shape)
customer_data.head()

# Extract the relevant columns from the data
data = customer_data.iloc[:, 3:5].values

# Create a dendrogram plot
plt.figure(figsize=(10, 7))
plt.title("Customer Dendrograms")
dend = shc.dendrogram(shc.linkage(data, method='ward'))

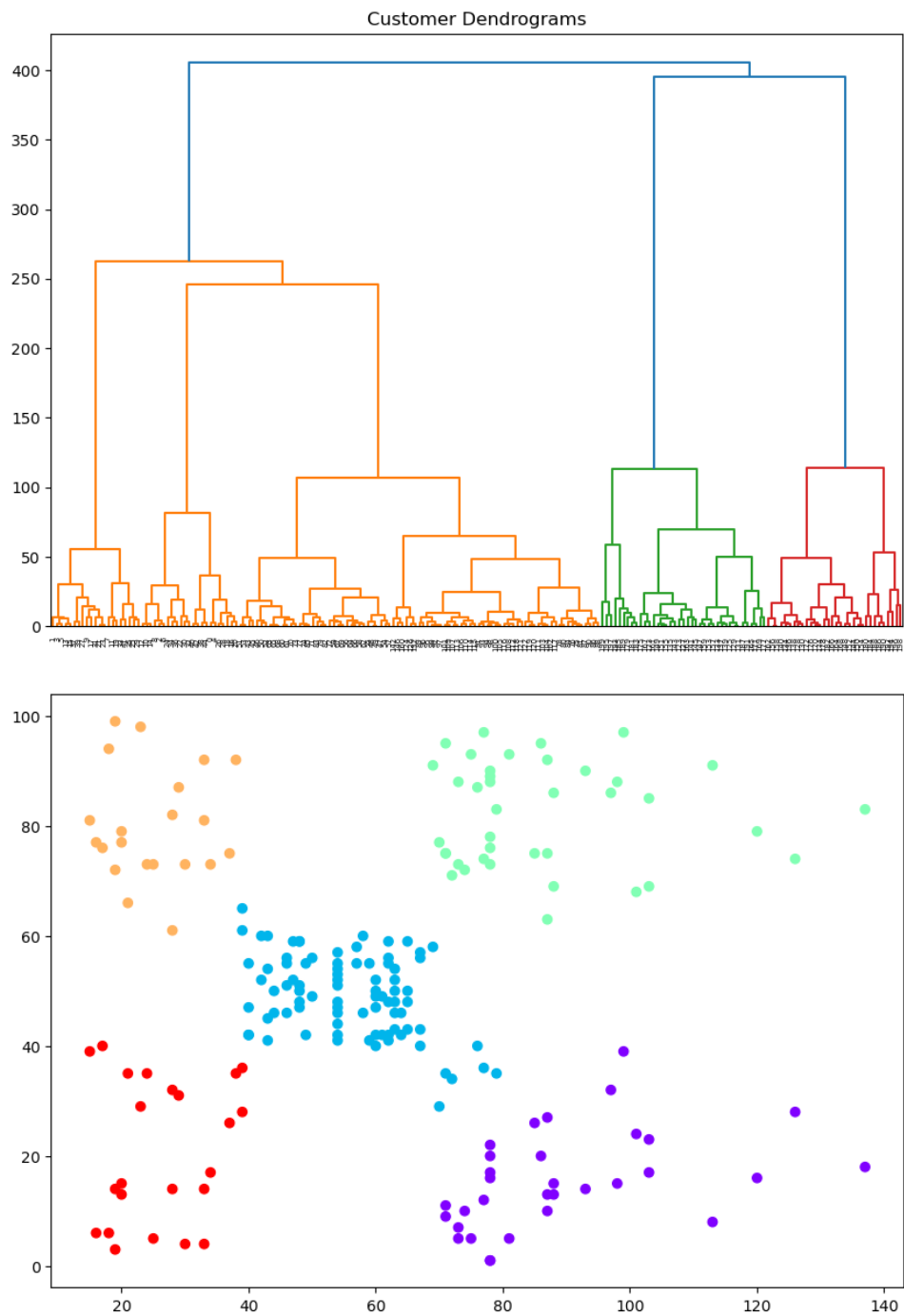
# Perform hierarchical clustering
cluster = AgglomerativeClustering(n_clusters=5, affinity='euclidean', linkage='ward')
cluster_labels = cluster.fit_predict(data)

# Create a scatter plot to visualize the clusters
plt.figure(figsize=(10, 7))
plt.scatter(data[:, 0], data[:, 1], c=cluster_labels, cmap='rainbow')
plt.show()
```

OUTPUT:

Out[5]:

	customer_id	gender	age	annual_income	spending_score
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40



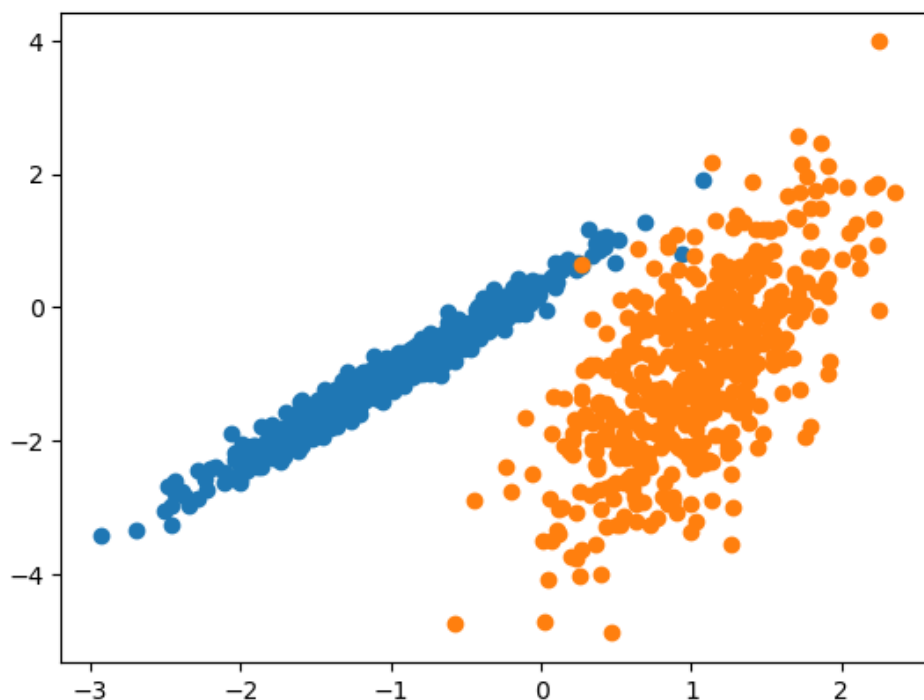
## b) Clustering

### 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_cluste
rs_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 No: 9

### SUPERVISED LEARNING METHODS USING PYTHON

**AIM:** 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:

**Step 1:** First we need to import pandas and numpy. Pandas are basically use for table manipulations. Using Pandas package, we are going to upload Titanic training dataset and then by using head () function we will look at first five rows.

```
import pandas as pd
import numpy as np
titanic= pd.read_csv("/content/sample_data/train.csv")
titanic.head()
```

#### Output:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

**Step 2:** Create Two Data Frames, one containing categories and one containing numbers

```
titanic_cat = titanic.select_dtypes(object)
titanic_num = titanic.select_dtypes(np.number)
```

**Step 3:** Now we need to drop two columns (name column and ticket column)

```
titanic_cat.head()
```

**Output:**

]:

	Name	Sex	Ticket	Cabin	Embarked
0	Braund, Mr. Owen Harris	male	A/5 21171	NaN	S
1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	PC 17599	C85	C
2	Heikkinen, Miss. Laina	female	STON/O2. 3101282	NaN	S
3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	113803	C123	S
4	Allen, Mr. William Henry	male	373450	NaN	S

titanic\_num.head()

**Output:**

Out[4]:

	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\_cat.drop(['Name','Ticket'], axis=1, inplace=True)

titanic\_cat.head()

**Step 4: Now to find the null values present in the above column**

titanic\_cat.isnull().sum()

**Output:**

```
Out[6]: Sex      0
Cabin    687
Embarked    2
dtype: int64
```

**Step 5: Replace all the null values present with the maximum count category**

titanic\_cat.Cabin.fillna(titanic\_cat.Cabin.value\_counts().idxmax(), inplace=True)

titanic\_cat.Embarked.fillna(titanic\_cat.Embarked.value\_counts().idxmax(), inplace=True)

**Step 6: After successfully removing all the null values our new data set is ready.**

titanic\_cat.head(20)

**Output:**



```
In [8]: titanic_cat.head(20)
```

```
Out[8]:
```

	Sex	Cabin	Embarked
0	male	B00 B00	S
1	female	C85	C
2	female	B00 B00	S
3	female	C123	S
4	male	B00 B00	S
5	male	B00 B00	Q
6	male	E46	S
7	male	B00 B00	S
8	female	B00 B00	S
9	female	B00 B00	C
10	female	G0	S
11	female	C103	S
12	male	B00 B00	S
13	male	B00 B00	S
14	female	B00 B00	S
15	female	B00 B00	S
16	male	B00 B00	Q
17	male	B00 B00	S
18	female	B00 B00	S
19	female	B00 B00	C

**Step 7:** The next step will be to replace all the categories with Numerical Labels. For that we will be using LabelEncoders Method.

```
from sklearn.preprocessing import LabelEncoder
```

```
le = LabelEncoder()
```

```
titanic_cat = titanic_cat.apply(le.fit_transform)
```

**Step 8:** Now we have only one column left which contain null value in it (Age). Let's replace it with mean

```
titanic_cat.head()
```

**Output:**

	Sex	Cabin	Embarked
0	1	47	2
1	0	81	0
2	0	47	2
3	0	55	2
4	1	47	2

```
titanic_num.isna().sum()
```

**Output:**

---

```

PassengerId      0
Survived          0
Pclass           0
Age             177
SibSp            0
Parch            0
Fare             0
dtype: int64

```

---

```

titanic_num.Age.fillna(titanic_num.Age.mean(), inplace=True)
titanic_num.isna().sum()

```

**Output:**

/usr/local/lib/python3.7/dist-packages/pandas/core/generic.py:6392: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-versus-a-copy](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)  
return self.\_update\_inplace(result)

```

PassengerId      0
Survived          0
Pclass           0
Age             0
SibSp            0
Parch            0
Fare             0
dtype: int64

```

**Step 9:** Now we need to remove the unnecessary columns, since the passengerid is an unnecessary column, we need to drop it

```

titanic_num.drop(['PassengerId'], axis=1, inplace=True)
titanic_num.head()

```

**Output:**

	Survived	Pclass	Age	SibSp	Parch	Fare
0	0	3	22.0	1	0	7.2500
1	1	1	38.0	1	0	71.2833
2	1	3	26.0	0	0	7.9250
3	1	1	35.0	1	0	53.1000
4	0	3	35.0	0	0	8.0500

**Step 10:** Now we will combine two data frames and make it as one

```

titanic_final = pd.concat([titanic_cat,titanic_num],axis=1)
titanic_final.head()

```

**Output:**

	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

**Step 11:** Now we will define dependent and independent variables

```
X=titanic_final.drop(['Survived'],axis=1)
```

```
Y= titanic_final['Survived']
```

**Step 12:** Now we will be taking 80% of the data as our training set, and remaining 20% as our test set.

```
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)
```

```
(712, 712, 179, 179)
```

**Step 13:** Now we will import all the algorithms

```
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
```

**Step 14:** Now we will initialize them in respective variables

```
LR = LogisticRegression()
```

```
KNN = KNeighborsClassifier()
```

```
NB = GaussianNB()
```

```
LSVM = LinearSVC()
```

```
NLSVM = SVC(kernel='rbf')
```

```
DT = DecisionTreeClassifier()
```

```
RF = RandomForestClassifier()
```

**Step 15:** Now we will train our model

```
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)
```

**Step 16:** Now we need to predict the test data set and compare the accuracy

score

```
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)
from sklearn.metrics import accuracy_score
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))
```

### Final Output:

```
Logistic Regression is 83.798883 percent accurate
KNN is 75.977654 percent accurate
Naive Bayes is 82.681564 percent accurate
Linear SVMs is 65.921788 percent accurate
Non Linear SVMs is 74.301676 percent accurate
Decision Trees is 81.005587 percent accurate
Random Forests is 85.474860 percent accurate
```

---

## **Practical No: 10**

### **Design an Artificial Intelligence application to implement 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):
        # Simulating weather conditions (you can modify this as needed)
        self.weather = input("Enter the weather (sunny, rainy, windy, snowy): ").lower()

    def suggest_clothes(self):
        if self.weather == "sunny":
            print(
                "It's sunny outside. You should wear light clothes, sunglasses, and sunscreen."
            )
        elif self.weather == "rainy":
            print(
                "It's rainy outside. Don't forget an umbrella, raincoat, and waterproof shoes."
            )
        elif self.weather == "windy":
            print("It's windy outside. Wear layers and a jacket to stay warm.")
        elif self.weather == "snowy":
            print(
                "It's snowy outside. Dress warmly with a heavy coat, gloves, and boots."
            )
        else:
            print(
                "Sorry, I don't understand the weather condition. Please enter sunny, rainy, windy, or snowy."
            )

def main():
    agent = ClothesAgent()
    agent.get_weather()
    agent.suggest_clothes()
```

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

### Final Output:

```
Enter the weather (sunny, rainy, windy, snowy): sunny  
It's sunny outside. You should wear light clothes, sunglasses, and sunscreen.
```

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

```
Enter the weather (sunny, rainy, windy, snowy): snowy  
It's snowy outside. Dress warmly with a heavy coat, gloves, and boots.
```

## Practical No: 11

### Design an application to simulate language parser.

**AIM: Design an application to simulate language parser.**

**Parsing is the process of analysing a sentence, breaking it down into smaller components, and identifying the grammatical structure of the sentence. It is a crucial component of NLP and helps machines understand human language.**

#### Code:

```
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)
```

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

```
[nltk data] Downloading package punkt to
```

```
import string
```

```
def remove_punctuation(input_string):  
    # Define a string of punctuation marks and symbols
```

```
punctuations = string.punctuation

# Remove the punctuation marks and symbols from the input string
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)
```

---

```
hello nlp world in this example we are going to do the basics of text processing which will be used later
```

---

```
#in this code, we are not using any libraries
#tokenize without using any function from string or any other function.
#only using loops and if/else
```

```
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)
```

```
['hello', 'nlp', 'world', 'in', 'this', 'example', 'we', 'are', 'going', 'to', 'do', 'the', 'basics', 'of', 'text', 'processi  
ng', 'which', 'will', 'be', 'used', 'later']
```

```
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)
```

```
# Tokenize the text
```

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

```
# Print the tokens
```

```
print(tokens)
```

```
['hello', 'nlp', 'world', 'in', 'this', 'example', 'we', 'are', 'going', 'to', 'do', 'the', 'basics', 'of', 'text', 'processi  
ng', 'which', 'will', 'be', 'used', 'later']
```

```
import nltk
```

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

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

```
print(tokens)
```

```
print(tokens)  
['We', "'re", 'going', 'to', 'John', "'s", 'house', 'today', '.']
```

