Practical – 1

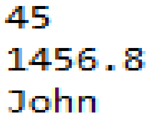
Aim: Perform practical using Python.

1.1 Write a program to demonstrate variable creation in python.

Program -

```
# An integer assignment
age = 45


# A floating point
salary = 1456.8


# A string
name = "John"


print(age)
print(salary)
print(name)
```

Output -



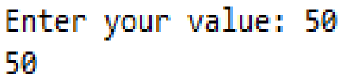
```
45
1456.8
John
```

1.2 Write a program to demonstrate command input in python.

Program -

```
val = input("Enter your value: ")
print(val)
```

Output –



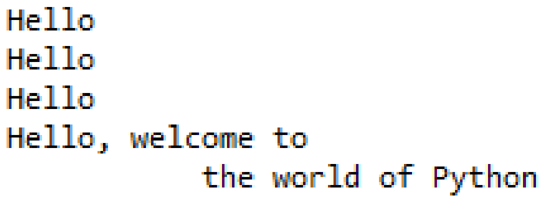
```
Enter your value: 50
50
```

1.3 Write a program to demonstrate numbers and stings in python.

Program –

```
my_string = 'Hello'

print(my_string)


my_string = "Hello"

print(my_string)


my_string = '''Hello'''

print(my_string)


# triple quotes string can extend multiple lines

my_string = """Hello, welcome to

           the world of Python"""

print(my_string)
```

Output –



```
Hello
Hello
Hello
Hello, welcome to
           the world of Python
```

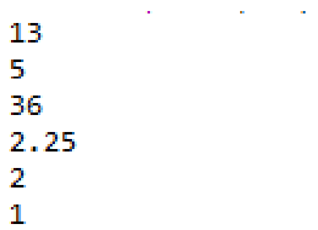
1.4 Write a program to demonstrate operators in python.

Program –

```
# Examples of Arithmetic Operator
```

```
a = 9

b = 4


# Addition of numbers

add = a + b

# Subtraction of numbers

sub = a - b

# Multiplication of number

mul = a * b

# Division(float) of number

div1 = a / b

# Division(floor) of number

div2 = a // b

# Modulo of both number

mod = a % b


# print results

print(add)

print(sub)

print(mul)

print(div1)

print(div2)

print(mod)
```

Output –



```
13
5
36
2.25
2
1
```

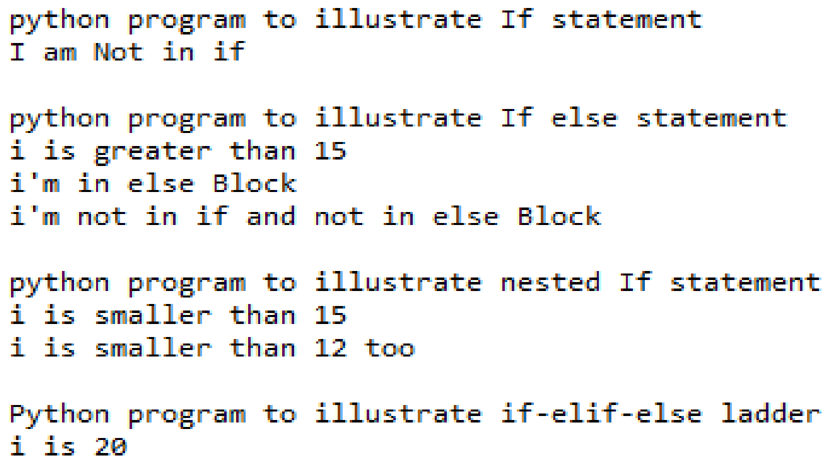
1.5 Write a program to demonstrate decision making in python.

Program –

```
print("python program to illustrate If statement ")
i = 10
if (i > 15):
   print ("10 is less than 15")
print ("I am Not in if")
print()
print("python program to illustrate If else statement")
i = 20;
if (i < 15):
    print ("i is smaller than 15")
    print ("i'm in if Block")
else:
    print ("i is greater than 15")
    print ("i'm in else Block")
print ("i'm not in if and not in else Block")
print()
print("python program to illustrate nested If statement")
i = 10
```

```
if (i == 10):
    #  First if statement
    if (i < 15):
        print ("i is smaller than 15")
    # Nested - if statement
    # Will only be executed if statement above
    # it is true
    if (i < 12):
        print ("i is smaller than 12 too")
    else:
        print ("i is greater than 15")
print()
print("Python program to illustrate if-elif-else ladder")
i = 20
if (i == 10):
          print ("i is 10")
elif (i == 15):
          print ("i is 15")
elif (i == 20):
          print ("i is 20")
else:
          print ("i is not present")
print()
```

Output –



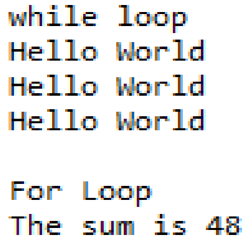
```
python program to illustrate If statement
I am Not in if

python program to illustrate If else statement
i is greater than 15
i'm in else Block
i'm not in if and not in else Block

python program to illustrate nested If statement
i is smaller than 15
i is smaller than 12 too

Python program to illustrate if-elif-else ladder
i is 20
```

1.6 Write a program to demonstrate loops and function in python.

Program –

```
print("while loop")
count = 0
while (count < 3):
   count = count + 1
   print("Hello World")
print()
print("For Loop")
numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11]

sum = 0

for val in numbers:
        sum = sum+val

print("The sum is", sum)
```

Output –



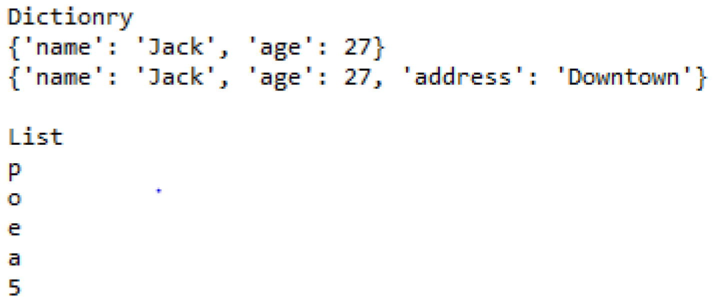
```
while loop
Hello World
Hello World
Hello World

For Loop
The sum is 48
```

1.7 Write a program to demonstrate lists and dictionary in python.

Program –

```
print()

print("Dictionry")

my_dict = {'name':'Jack', 'age': 26}

my_dict['age'] = 27

print(my_dict)

my_dict['address'] = 'Downtown'

print(my_dict)


print()

print("List")

my_list = ['p','r','o','b','e']

print(my_list[0])

print(my_list[2])

print(my_list[4])

n_list = ["Happy", [2,0,1,5]]

print(n_list[0][1])

print(n_list[1][3])
```

Output –



```
Dictionry
{'name': 'Jack', 'age': 27}
{'name': 'Jack', 'age': 27, 'address': 'Downtown'}

List
p
o
e
a
5
```

Conclusion –

Thus, from this practical we conclude that python has a standard library for development and a few for AI and it has an intuitive syntax, basic control flow, and data structures.

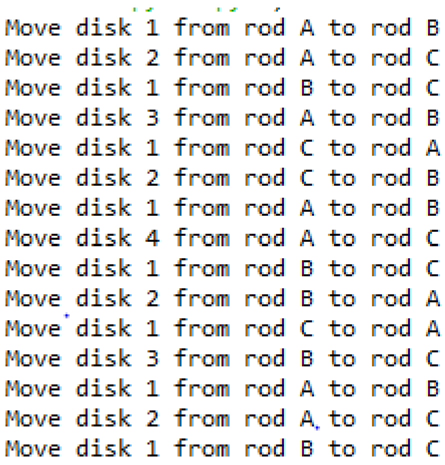
Practical - 2

Aim: Write a program to solve Tower of Hanoi problem in python using recursion.

Program –

```
def TowerOfHanoi(n , from_rod, to_rod, aux_rod):
    if n == 1:
        print("Move disk 1 from rod",from_rod,"to rod",to_rod)
        return
    TowerOfHanoi(n-1, from_rod, aux_rod, to_rod)
    print("Move disk",n,"from rod",from_rod,"to rod",to_rod)
    TowerOfHanoi(n-1, aux_rod, to_rod, from_rod)


n = 4
TowerOfHanoi(n, 'A', 'C', 'B')
```

Output –



```
Move disk 1 from rod A to rod B
Move disk 2 from rod A to rod C
Move disk 1 from rod B to rod C
Move disk 3 from rod A to rod B
Move disk 1 from rod C to rod A
Move disk 2 from rod C to rod B
Move disk 1 from rod A to rod B
Move disk 4 from rod A to rod C
Move disk 1 from rod B to rod C
Move disk 2 from rod B to rod A
Move disk 1 from rod C to rod A
Move disk 3 from rod B to rod C
Move disk 1 from rod A to rod B
Move disk 2 from rod A to rod C
Move disk 1 from rod B to rod C
```

Conclusion –

Thus, from this practical we conclude that tower of Hanoi is a mathematical puzzle where we have 3 rods and n disks. The objective of the puzzle is to move the entire stack to another rod.

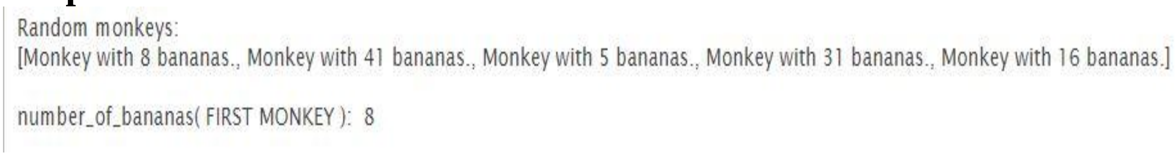
Practical – 3

Aim: Write a program to solve Monkey Banana problem in python.

Program –

```
import random

class Monkey:
        def __init__(self, bananas):
                self.bananas = bananas

        def __repr__(self):
                return "Monkey with %d bananas." % self.bananas

monkeys = [Monkey(random.randint(0, 50)) for i in range(5)]


print "Random monkeys:"
print monkeys

print

def number_of_bananas(monkey):
    """Returns number of bananas that monkey has."""
    return monkey.bananas

print "number_of_bananas( FIRST MONKEY ): ", number_of_bananas(monkeys[0])

print

max_monkey = max(monkeys, key=number_of_bananas)
print "Max monkey: ", max_monkey
```

Output –



```
Random monkeys:
[Monkey with 8 bananas., Monkey with 41 bananas., Monkey with 5 bananas., Monkey with 31 bananas., Monkey with 16 bananas.]

number_of_bananas( FIRST MONKEY ):  8
```

Conclusion:

Thus, from this practical we conclude that monkey and banana problem is often used as a simple example of problem solving.

Practical – 4

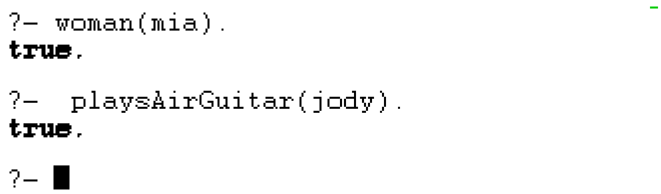
Aim: Preform practical using Prolog.

4.1 Write a program in prolog to implement simple facts and Queries.

Program –

```
woman(mia).
woman(jody).
woman(yolanda).
playsAirGuitar(jody).
party.
```

Output –



```
?- woman(mia).
true.

?-  playsAirGuitar(jody).
true.

?- █
```

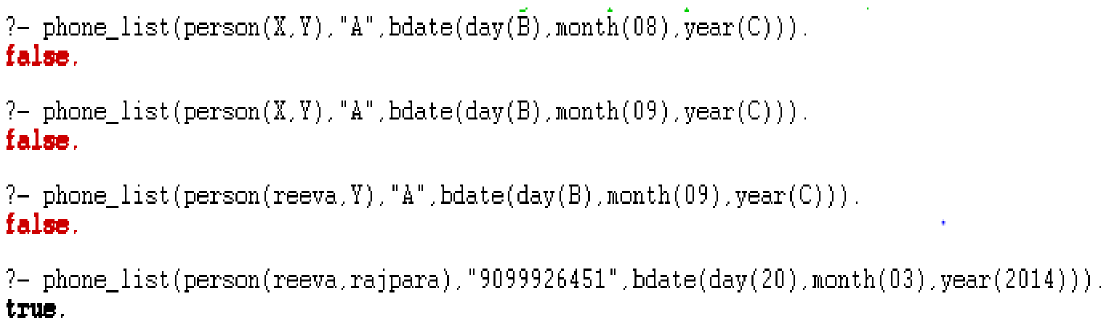
4.2 Write a program in prolog to implement phone list which store name, phone number and birthdays of friends and family members. Write a query to get birthday a list of people whose birthdays are in the current month.

Program –

```
phone_list(person(vruti,antala),"9099901158",bdate(day(07),month(08),year(1999))).
phone_list(person(alaknand,faldu),"7984826480",bdate(day(15),month(05),year(1999))).
phone_list(person(vrusabh,shah),"894100568",bdate(day(10),month(07),year(1999))).
phone_list(person(foram,thakkar),"9409474751",bdate(day(15),month(08),year(1999))).
phone_list(person(vatsal,halpara),"8155979674",bdate(day(19),month(04),year(1999))).
phone_list(person(krish,patel),"7841025569",bdate(day(11),month(07),year(1999))).
phone_list(person(esha,bhagat),"9099989456",bdate(day(28),month(09),(1999))).
phone_list(person(kavya,mistry),"9099986486",bdate(day(25),month(11),year(1999))).
phone_list(person(nitsha,gupta),"9845562312",bdate(day(18),month(02),year(1
```

999))).
phone_list(person(aditya,dobariya),"9726826768",bdate(day(18),month(10),year(1997))).
phone_list(person(reeva,rajpara),"9099926451",bdate(day(20),month(03),year(2014))).

Output –

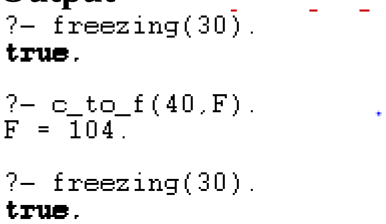


```
?- phone_list(person(X,Y),"A",bdate(day(B),month(08),year(C))).
false.

?- phone_list(person(X,Y),"A",bdate(day(B),month(09),year(C))).
false.

?- phone_list(person(reeva,Y),"A",bdate(day(B),month(09),year(C))).
false.

?- phone_list(person(reeva,rajpara),"9099926451",bdate(day(20),month(03),year(2014))).
true.
```

4.3 Write predicates one converts centigrade temperatures to Fahrenheit, the other checks if a temperature is below freezing.

Program –
convert_clpr(Celsius, Fahrenheit) :-
Celsius is (Fahrenheit - 32) * 5 / 9 .
c_to_f(C,F) :-
F is C * 9 / 5 + 32.
freezing(F) :-
F =< 32

Output –



```
?- freezing(30).
true.

?- c_to_f(40,F).
F = 104.

?- freezing(30).
true.
```

Conclusion –

Thus, from this practical we conclude that prolog is a logic programming language associated with artificial intelligence and computational linguistics.

Practical – 5

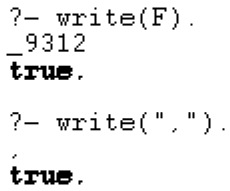
Aim: Perform practical using arithmetic operators in prolog

5.1 Write a program to display Fibonacci series in prolog.

Program –

```
fib(0,0).
fib(1,1).
fib(F,N) :-
   N>1,
   N1 is N-1,
   N2 is N-2,
   fib(F1,N1),
   fib(F2,N2),
   F is F1+F2,
   write(F," ,").
```

Output –



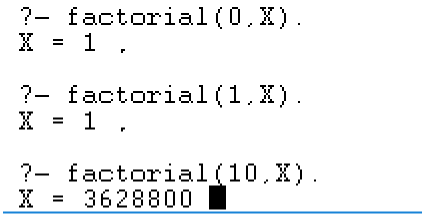
```
?- write(F).
_9312
true.

?- write(",").
,
true.
```

5.2 Write a program to find factorial of a number in prolog using recursion.

Program -

```
factorial(0,1).

factorial(A,B) :-
          A > 0,
          C is A-1,
          factorial(C,D),
          B is A*D.
```

Output –



```
?- factorial(0,X).
X = 1 .

?- factorial(1,X).
X = 1 .

?- factorial(10,X).
X = 3628800 
```

Conclusion:

Thus, from this practical we conclude that Prolog is not the programming language of choice for carrying out heavy-duty mathematics. It does, however, provide arithmetical capabilities. The pattern for evaluating arithmetic expressions is (where `Expression` is some arithmetical expression

Practical – 6

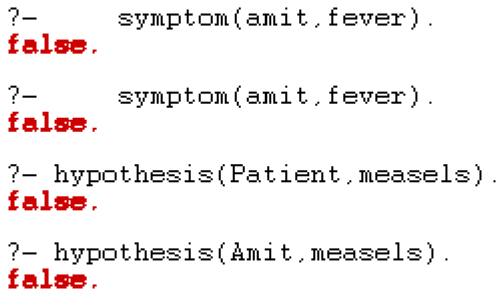
Aim: Write a prolog program for medical diagnosis system of childhood diseases.

Program:

```
domains
   disease,indication,name = symbol

predicates
   hypothesis(name,disease)
   symptom(name,indication)

clauses
   symptom(amit,fever).
   symptom(amit,rash).
   symptom(amit,headache).
   symptom(amit,runn_nose).

   symptom(kaushal,chills).
   symptom(kaushal,fever).
   symptom(kaushal,hedache).

   symptom(dipen,runny_nose).
   symptom(dipen,rash).
   symptom(dipen,flu).


   hypothesis(Patient,measels):-
      symptom(Patient,fever),
      symptom(Patient,cough),
      symptom(Patient,conjunctivitis),
      symptom(Patient,rash).

   hypothesis(Patient,german_measles) :-
      symptom(Patient,fever),
      symptom(Patient,headache),
      symptom(Patient,runny_nose),
      symptom(Patient,rash).
```

```
hypothesis(Patient,flu) :-
   symptom(Patient,fever),
   symptom(Patient,headache),
   symptom(Patient,body_ache),
   symptom(Patient,chills).

hypothesis(Patient,common_cold) :-
   symptom(Patient,headache),
   symptom(Patient,sneezing),
   symptom(Patient,sore_throat),
   symptom(Patient,chills),
   symptom(Patient,runny_nose).

hypothesis(Patient,mumps) :-
   symptom(Patient,fever),
   symptom(Patient,swollen_glands).

hypothesis(Patient,chicken_pox) :-
   symptom(Patient,fever),
   symptom(Patient,rash),
   symptom(Patient,body_ache),
   symptom(Patient,chills).
```

Output –



```
?-      symptom(amit,fever).
false.

?-      symptom(amit,fever).
false.

?- hypothesis(Patient,measels).
false.

?- hypothesis(Amit,measels).
false.
```

Conclusion –

Thus, from this practical we conclude that prolog is most useful in the areas related to AI research, such as problem solving, planning or naural language interpretation.

Practical – 7

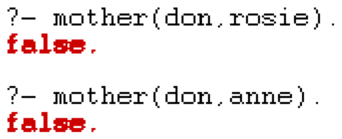
Aim: Write a program which contains three predicates: male, female, parent. Make rules for following family relations: father, mother, grandfather, grandmother, brother, sister, uncle, aunt, nephew and niece, cousin.

Program –

```
:- discontiguous male/1, female/1, parent/2.
male(dicky).
male(randy).
male(mike).
male(don).
male(elmer).
female(anne).
female(rosie).
female(esther).
female(mildred).
female(greatgramma).
male(blair).
male(god).
```

```
female(god).

parent(don,randy).
parent(don,mike).
parent(don,anne).
parent(rosie,randy).
parent(rosie,mike).
parent(rosie,anne).
parent(elmer,don).
parent(mildred,don).
parent(esther,rosie).
parent(esther,dicky).
parent(greatgramma,esther).
parent(randy,blair).

male(mel).
male(teo).
parent(melsr,mel).
parent(melsr,teo).

indian(anne).
indian(X) :-  ancestor(X,anne).
indian(X) :- ancestor(anne,X).
relation(X,Y) :- ancestor(A,X), ancestor(A,Y).

father(X,Y) :- male(X),parent(X,Y).
father(god, _) :- male(god).
mother(X,Y) :- female(X),parent(X,Y).
son(X,Y) :- male(X),parent(Y,X).
daughter(X,Y) :- female(X),parent(Y,X).
grandfather(X,Y) :- male(X),parent(X,Somebody),parent(Somebody,Y).
aunt(X,Y) :- female(X),sister(X,Mom),mother(Mom,Y).
aunt(X,Y) :- female(X),sister(X,Dad),father(Dad,Y).
sister(X,Y) :- female(X),parent(Par,X),parent(Par,Y), X \= Y.
uncle(X,Y) :- brother(X,Par),parent(Par,Y).
cousin(X,Y) :- uncle(Unc , X),father(Unc,Y).
ancestor(X,Y) :- parent(X,Y).
ancestor(X,Y) :- parent(X,Somebody),ancestor(Somebody,Y).
brother(X,Y) :-  male(X),parent(Somebody,X),parent(Somebody,Y), X \= Y.
```

Output –



```
?- mother(don,rosie).
false.

?- mother(don,anne).
false.
```

Conclusion –

Thus, from this practical we conclude that **Prolog** is intended primarily as a declarative programming language. In **prolog**, logic is expressed as relations (called as Facts and Rules). Core heart of **prolog** lies at the logic being applied.

Practical – 8

Aim: Write a program to perform following operations on lists in prolog.

8.1 Create a List

?- X=alpha,Y=27,Z=[alpha,beta],write('List is:'),write([X,Y,Z]),nl.

List is:[alpha,27,[alpha,beta]]

X = alpha,

Y = 27,

Z = [alpha, beta].

8.2 Write in List

?- write([alpha|[beta,gamma,delta]]),nl.

```
[alpha,beta,gamma,delta]
true.

?- write([alpha,beta,gamma|[delta]]),nl.
[alpha,beta,gamma,delta]
true.

?- write([alpha,beta,gamma,delta|[]]),nl.
[alpha,beta,gamma,delta]
true.

?- write([alpha,beta|[gamma,delta]]),nl.
[alpha,beta,gamma,delta]
true.

?- write([alpha,beta|[gamma|[delta|[]]]]),nl.
[alpha,beta,gamma,delta]
true.

?- L=[red,blue,green,yellow],write([brown|L]),nl.
[brown,red,blue,green,yellow]
L = [red, blue, green, yellow].
```

8.3 Member

```
?- member(a,[a,b,c]).
```

```
true .


?- member(mypred(a,b,c),[q,r,s,mypred(a,b,c),w]).

true .


?- member([1,2,3],[a,b,[1,2,3],c]).

true .


?- member(X,[a,b,c]).

X = a .
```

8.4 Length

```
?- member(a,[a,b,c]).

true .


?- member(mypred(a,b,c),[q,r,s,mypred(a,b,c),w]).

true .


?- member([1,2,3],[a,b,[1,2,3],c]).

true .


?- member(X,[a,b,c]).

X = a .


?- length([a,b,c,d],X).
```

```
X = 4.


?- length([[a,b,c],[d,e,f],[g,h,i]],L).
L = 3.


?- length([],L).
L = 0.


?- length([a,b,c],3).
true.


?- length([a,b,c],4).
false.


?- N is 3,length([a,b,c],N).
N = 3.
```

8.5 Reverse

```
?- reverse([1,2,3,4],L).
L = [4, 3, 2, 1].


?- reverse(L,[1,2,3,4]).
L = [4, 3, 2, 1] .


?- reverse([[dog,cat],[1,2],[bird,mouse],[3,4,5,6]],L).
```

```
L = [[3, 4, 5, 6], [bird, mouse], [1, 2], [dog, cat]].


?- reverse([1,2,3,4],[4,3,2,1]).

true.
```

8.6 Delete

```
del(X,[X|T],T).


del(X,[H|T],[H|T1]):-


del(X,T,T1).
```

O/p:

```
?- del(1,[1,2,3,4,5],X).

X = [2, 3, 4, 5]
```

8.7 Append

```
?- append(L1,L2,[1,2,3,4,5]).

L1 = [],

L2 = [1, 2, 3, 4, 5] .


?- append([[a,b,c],d,e,f],[g,h,[i,j,k]],L).

L = [[a, b, c], d, e, f, g, h, [i, j|...]].


?- append([],[1,2,3],L).

L = [1, 2, 3].
```

```
?- append(X,[Y|Z],[1,2,3,4,5,6]).

X = [],

Y = 1,

Z = [2, 3, 4, 5, 6] .
```

8.8 Permutation

```
is_permutation(Xs,Ys):-
    msort(Xs,Sorted),
    msort(Ys,Sorted).
```

O/p

```
?- permutation([1,2], [X,Y]).

X = 1,

Y = 2 .


?- permutation([1,2,3], [X,Y,Z]).

X = 1,

Y = 2,

Z = 3 .
```

8.9 Add New Element

```
add(X, L, [X|L]).


add_list([], L, L).
add_list([H|T], L, L1) :- add(H, L2, L1), add_list(T, L, L2).
```

O/p

```
?- add_list([1,2,3],X,A).

A = [1, 2, 3|X].


?- add_list([1,2,3],X,A), writeln(A-X), add_list([4,5],Y,X).

[1,2,3|_3096]-_3096

X = [4, 5|Y],

A = [1, 2, 3, 4, 5|Y].


?- add_list([1,2,3],X,A),writeln(A-X),add_list([4,5],Y,X),Y=[].

[1,2,3|_3096]-_3096

X = [4, 5],

A = [1, 2, 3, 4, 5],

Y = [].
```

Conclusion –

Thus, from this practical we conclude that Prolog also has a special facility to split the first part of the list (called the head) away from the rest of the list (known as the tail). We can place a special symbol | (pronounced 'bar') in the list to distinguish between the first item in the list and the remaining list.

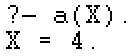
Practical – 9

Aim: Write a program to demonstrate cut and fail in prolog.

Program –

```
a(X) :- b(X),c(X),fail.
a(X) :- d(X).

b(1).
b(4).
c(1).
c(3).

d(4).
```

Output –



```
?- a(X).
X = 4.
```

Conclusion –

Thus, from this practical we conclude that The **cut**, in **Prolog**, is a goal, written as ! , which always succeeds, but cannot be backtracked past. It is used to prevent unwanted backtracking, for example, to prevent extra solutions being found by **Prolog**. **fail**/0 is a special symbol that will immediately **fail** when **Prolog** encounters it as a goal. That may not sound too useful, but remember: when **Prolog fails**, it tries to backtrack.

Practical – 10

Aim: Write a program to demonstrate Depth First Search Tree and Breadth First Search Tree for Water-Jug Problem in python.

Program –

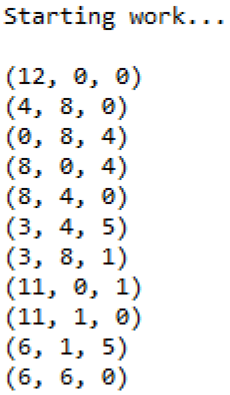
```
# 3 water jugs capacity -> (x,y,z) where x>y>z
# initial state (12,0,0)
# final state (6,6,0)


capacity = (12,8,5)
# Maximum capacities of 3 jugs -> x,y,z
x = capacity[0]
y = capacity[1]
```

```
z = capacity[2]

# to mark visited states
memory = {}

# store solution path
ans = []

def get_all_states(state):
  # Let the 3 jugs be called a,b,c
  a = state[0]
  b = state[1]
  c = state[2]

  if(a==6 and b==6):
      ans.append(state)
      return True

  # if current state is already visited earlier
  if((a,b,c) in memory):
      return False

  memory[(a,b,c)] = 1

  #empty jug a
  if(a>0):
      #empty a into b
      if(a+b<=y):
          if( get_all_states((0,a+b,c)) ):
              ans.append(state)
              return True
      else:
          if( get_all_states((a-(y-b), y, c)) ):
              ans.append(state)
              return True
      #empty a into c
      if(a+c<=z):
          if( get_all_states((0,b,a+c)) ):
              ans.append(state)
              return True
      else:
          if( get_all_states((a-(z-c), b, z)) ):
              ans.append(state)
```

```
            return True

#empty jug b
if(b>0):
    #empty b into a
    if(a+b<=x):
        if( get_all_states((a+b, 0, c)) ):
            ans.append(state)
            return True
    else:
        if( get_all_states((x, b-(x-a), c)) ):
            ans.append(state)
            return True
    #empty b into c
    if(b+c<=z):
        if( get_all_states((a, 0, b+c)) ):
            ans.append(state)
            return True
    else:
        if( get_all_states((a, b-(z-c), z)) ):
            ans.append(state)
            return True

#empty jug c
if(c>0):
    #empty c into a
    if(a+c<=x):
        if( get_all_states((a+c, b, 0)) ):
            ans.append(state)
            return True
    else:
        if( get_all_states((x, b, c-(x-a))) ):
            ans.append(state)
            return True
    #empty c into b
    if(b+c<=y):
        if( get_all_states((a, b+c, 0)) ):
            ans.append(state)
            return True
    else:
        if( get_all_states((a, y, c-(y-b))) ):
            ans.append(state)
            return True
```

```
  return False

initial_state = (12,0,0)
print("Starting work...\n")
get_all_states(initial_state)
ans.reverse()
for i in ans:
  print(i)
```

Output –



```
Starting work...

(12, 0, 0)
(4, 8, 0)
(0, 8, 4)
(8, 0, 4)
(8, 4, 0)
(3, 4, 5)
(3, 8, 1)
(11, 0, 1)
(11, 1, 0)
(6, 1, 5)
(6, 6, 0)
```

Conclusion –

Thus, from this practical we conclude that in a **Water Jug Problem**: You are given two jugs, a 4-gallon one and a 3-gallon one, a pump which has unlimited **water** which you can use to fill the **jug**, and the ground on which **water** may be poured.

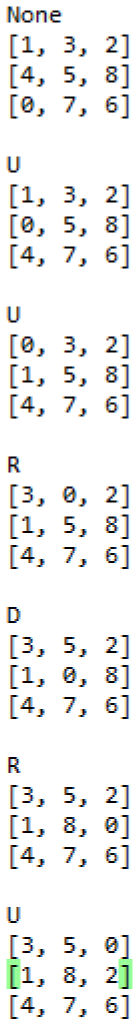
Practical -11

Aim: Write a program to solve 8 puzzle problem using A*Algorithm in python

Program –

```
import random
import itertools
import collections

class Node:
    """
    A class representing an Solver node
    - 'puzzle' is a Puzzle instance
    - 'parent' is the preceding node generated by the solver, if any
    - 'action' is the action taken to produce puzzle, if any
    """
    def __init__(self, puzzle, parent=None, action=None):
        self.puzzle = puzzle
        self.parent = parent
        self.action = action

    @property
    def state(self):
        """
        Return a hashable representation of self
        """
        return str(self)

    @property
    def path(self):
        """
        Reconstruct a path from to the root 'parent'
        """
        node, p = self, []
        while node:
            p.append(node)
            node = node.parent
        yield from reversed(p)

    @property
    def solved(self):
        """ Wrapper to check if 'puzzle' is solved """
        return self.puzzle.solved

    @property
    def actions(self):
        """ Wrapper for 'actions' accessible at current state """
        return self.puzzle.actions

    def __str__(self):
        return str(self.puzzle)

class Solver:
    """
    An '8-puzzle' solver
    - 'start' is a Puzzle instance
    """
    def __init__(self, start):
        self.start = start

    def solve(self):
        """
        Perform breadth first search and return a path
        to the solution, if it exists
        """
        queue = collections.deque([Node(self.start)])
        seen  = set()
        seen.add(queue[0].state)
        while queue:
            node = queue.pop()
            if node.solved:
                return node.path

            for move, action in node.actions:
                child = Node(move(), node, action)

                if child.state not in seen:
                    queue.appendleft(child)
                    seen.add(child.state)

class Puzzle:
    """
    A class representing an '8-puzzle'.
    - 'board' should be a square list of lists with integer entries 0...width^2 - 1
       e.g. [[1,2,3],[4,0,6],[7,5,8]]
    """
    def __init__(self, board):
        self.width = len(board[0])
        self.board = board

    @property
    def solved(self):
        """
        The puzzle is solved if the flattened board's numbers are in
        increasing order from left to right and the '0' tile is in the
        last position on the board
        """
        N = self.width * self.width
        return str(self) == ''.join(map(str, range(1,N))) + '0'

    @property
    def actions(self):
        """
        Return a list of 'move', 'action' pairs. 'move' can be called
        to return a new puzzle that results in sliding the '0' tile in
        the direction of 'action'.
        """
        def create_move(at, to):
            return lambda: self._move(at, to)

        moves = []
        for i, j in itertools.product(range(self.width),
                                      range(self.width)):
            direcs = {'R':(i, j-1),
                      'L':(i, j+1),
                      'D':(i-1, j),
                      'U':(i+1, j)}

            for action, (r, c) in direcs.items():
                if r >= 0 and c >= 0 and r < self.width and c < self.width and \
                   self.board[r][c] == 0:
                    move = create_move((i,j), (r,c)), action
                    moves.append(move)
        return moves

    def shuffle(self):
        """
        Return a new puzzle that has been shuffled with 1000 random moves
        """
        puzzle = self
        for _ in range(1000):
            puzzle = random.choice(puzzle.actions)[0]()
        return puzzle

    def copy(self):
        """
        Return a new puzzle with the same board as 'self'
        """
        board = []
        for row in self.board:
            board.append([x for x in row])
        return Puzzle(board)

    def _move(self, at, to):
        """
        Return a new puzzle where 'at' and 'to' tiles have been swapped.
        NOTE: all moves should be 'actions' that have been executed
        """
        copy = self.copy()
        i, j = at
        r, c = to
        copy.board[i][j], copy.board[r][c] = copy.board[r][c], copy.board[i][j]
        return copy

    def pprint(self):
        for row in self.board:
            print(row)
        print()

    def __str__(self):
        return ''.join(map(str, self))

    def __iter__(self):
        for row in self.board:
            yield from row


# example of use
board = [[1,2,3],[4,0,6],[7,5,8]]

puzzle = Puzzle(board)
puzzle = puzzle.shuffle()
s = Solver(puzzle)
p = s.solve()

for node in p:
    print(node.action)
    node.puzzle.pprint()
```

Output –



```
None
[1, 3, 2]
[4, 5, 8]
[0, 7, 6]

U
[1, 3, 2]
[0, 5, 8]
[4, 7, 6]

U
[0, 3, 2]
[1, 5, 8]
[4, 7, 6]

R
[3, 0, 2]
[1, 5, 8]
[4, 7, 6]

D
[3, 5, 2]
[1, 0, 8]
[4, 7, 6]

R
[3, 5, 2]
[1, 8, 0]
[4, 7, 6]

U
[3, 5, 0]
[1, 8, 2]
[4, 7, 6]
```

Conclusion –

Thus, from this practical we conclude that A* is computer **algorithm** that is widely used in pathfinding and graph traversal, which is the process of finding a path between multiple points, called "nodes". It enjoys widespread use due to its performance and accuracy.

Practical – 12

Aim: Write a program for game Tic-Tac-Toe using MINIMAX Algorithm in python.

Program –

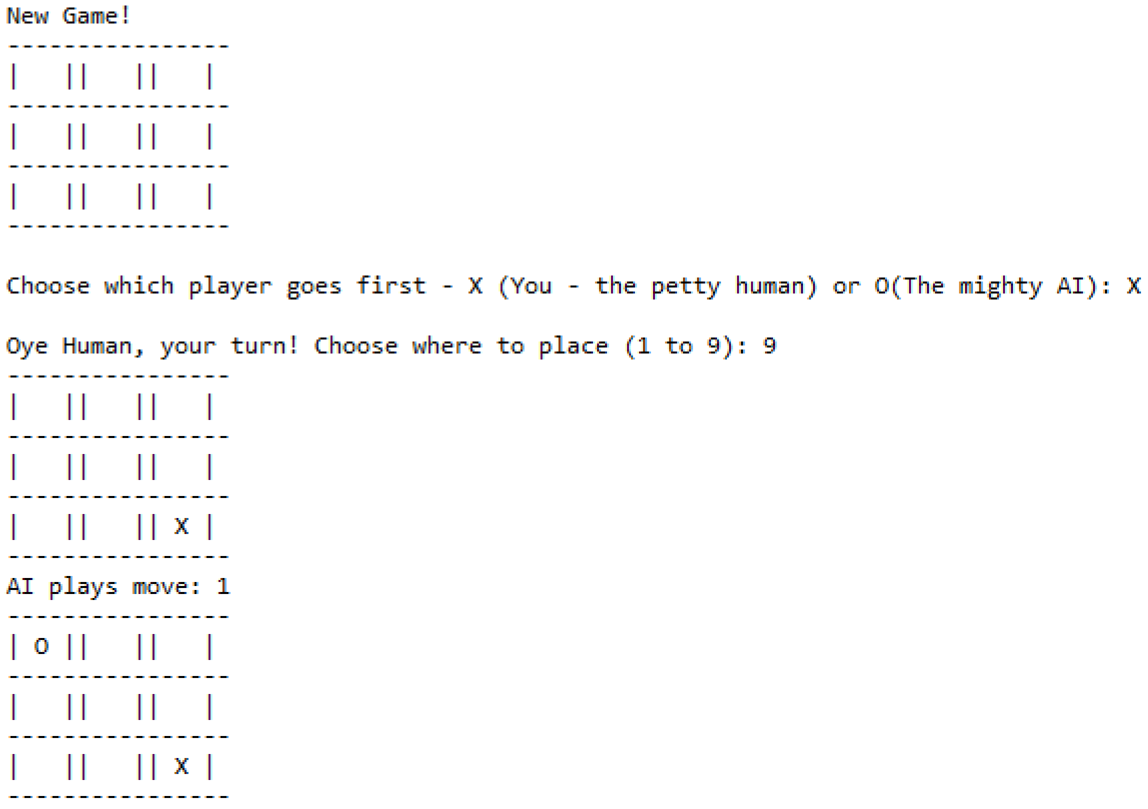
```
import numpy as np
from math import inf as infinity
```

```
game_state = [[' ',' ',' '],
              [' ',' ',' '],
              [' ',' ',' ']]
players = ['X','O']

def play_move(state, player, block_num):
    if state[int((block_num-1)/3)][(block_num-1)%3] is ' ':
        state[int((block_num-1)/3)][(block_num-1)%3] = player
    else:
        block_num = int(input("Block is not empty, ya blockhead! Choose again:
"))
        play_move(state, player, block_num)

def copy_game_state(state):
    new_state = [[' ',' ',' '],[' ',' ',' '],[' ',' ',' ']]
    for i in range(3):
        for j in range(3):
            new_state[i][j] = state[i][j]
    return new_state

def check_current_state(game_state):
    # Check if draw
    draw_flag = 0
    for i in range(3):
        for j in range(3):
            if game_state[i][j] is ' ':
                draw_flag = 1
    if draw_flag is 0:
        return None, "Draw"

    # Check horizontals
    if (game_state[0][0] == game_state[0][1] and game_state[0][1] ==
game_state[0][2] and game_state[0][0] is not ' '):
        return game_state[0][0], "Done"
    if (game_state[1][0] == game_state[1][1] and game_state[1][1] ==
game_state[1][2] and game_state[1][0] is not ' '):
        return game_state[1][0], "Done"
    if (game_state[2][0] == game_state[2][1] and game_state[2][1] ==
game_state[2][2] and game_state[2][0] is not ' '):
        return game_state[2][0], "Done"

    # Check verticals
    if (game_state[0][0] == game_state[1][0] and game_state[1][0] ==
game_state[2][0] and game_state[0][0] is not ' '):
        return game_state[0][0], "Done"
    if (game_state[0][1] == game_state[1][1] and game_state[1][1] ==
game_state[2][1] and game_state[0][1] is not ' '):
        return game_state[0][1], "Done"
    if (game_state[0][2] == game_state[1][2] and game_state[1][2] ==
game_state[2][2] and game_state[0][2] is not ' '):
        return game_state[0][2], "Done"

    # Check diagonals
    if (game_state[0][0] == game_state[1][1] and game_state[1][1] ==
game_state[2][2] and game_state[0][0] is not ' '):
        return game_state[1][1], "Done"
    if (game_state[2][0] == game_state[1][1] and game_state[1][1] ==
game_state[0][2] and game_state[2][0] is not ' '):
        return game_state[1][1], "Done"

    return None, "Not Done"

def print_board(game_state):
    print('----------------')
    print('| ' + str(game_state[0][0]) + ' || ' + str(game_state[0][1]) + ' || ' +
str(game_state[0][2]) + ' |')
    print('----------------')
    print('| ' + str(game_state[1][0]) + ' || ' + str(game_state[1][1]) + ' || ' +
str(game_state[1][2]) + ' |')
    print('----------------')
    print('| ' + str(game_state[2][0]) + ' || ' + str(game_state[2][1]) + ' || ' +
str(game_state[2][2]) + ' |')
    print('----------------')


def getBestMove(state, player):
    '''
    Minimax Algorithm
    '''
    winner_loser , done = check_current_state(state)
    if done == "Done" and winner_loser == 'O': # If AI won
        return 1
    elif done == "Done" and winner_loser == 'X': # If Human won
        return -1
    elif done == "Draw":    # Draw condition
```

```
        return 0

    moves = []
    empty_cells = []
    for i in range(3):
        for j in range(3):
            if state[i][j] is ' ':
                empty_cells.append(i*3 + (j+1))

    for empty_cell in empty_cells:
        move = {}
        move['index'] = empty_cell
        new_state = copy_game_state(state)
        play_move(new_state, player, empty_cell)

        if player == 'O':    # If AI
            result = getBestMove(new_state, 'X')    # make more depth tree for human
            move['score'] = result
        else:
            result = getBestMove(new_state, 'O')    # make more depth tree for AI
            move['score'] = result

        moves.append(move)

    # Find best move
    best_move = None
    if player == 'O':   # If AI player
        best = -infinity
        for move in moves:
            if move['score'] > best:
                best = move['score']
                best_move = move['index']
    else:
        best = infinity
        for move in moves:
            if move['score'] < best:
                best = move['score']
                best_move = move['index']

    return best_move

# PLaying
```

```
play_again = 'Y'
while play_again == 'Y' or play_again == 'y':
    game_state = [[' ',' ',' '],
              [' ',' ',' '],
              [' ',' ',' ']]
    current_state = "Not Done"
    print("\nNew Game!")
    print_board(game_state)
    player_choice = input("Choose which player goes first - X (You - the petty 
human) or O(The mighty AI): ")
    winner = None

    if player_choice == 'X' or player_choice == 'x':
        current_player_idx = 0
    else:
        current_player_idx = 1

    while current_state == "Not Done":
        if current_player_idx == 0: # Human's turn
            block_choice = int(input("Oye Human, your turn! Choose where to 
place (1 to 9): "))
            play_move(game_state ,players[current_player_idx], block_choice)
        else:   # AI's turn
            block_choice = getBestMove(game_state, players[current_player_idx])
            play_move(game_state ,players[current_player_idx], block_choice)
            print("AI plays move: " + str(block_choice))
        print_board(game_state)
        winner, current_state = check_current_state(game_state)
        if winner is not None:
            print(str(winner) + " won!")
        else:
            current_player_idx = (current_player_idx + 1)%2

        if current_state is "Draw":
            print("Draw!")

    play_again = input('Wanna try again?(Y/N) : ')
    if play_again == 'N':
        print('Suit yourself!')
```

Output –



```
New Game!
----------------
|   ||   ||   |
----------------
|   ||   ||   |
----------------
|   ||   ||   |
----------------

Choose which player goes first - X (You - the petty human) or O(The mighty AI): X

Oye Human, your turn! Choose where to place (1 to 9): 9
----------------
|   ||   ||   |
----------------
|   ||   ||   |
----------------
|   ||   || X |
----------------
AI plays move: 1
----------------
| O ||   ||   |
----------------
|   ||   ||   |
----------------
|   ||   || X |
----------------
```

Conclusion –

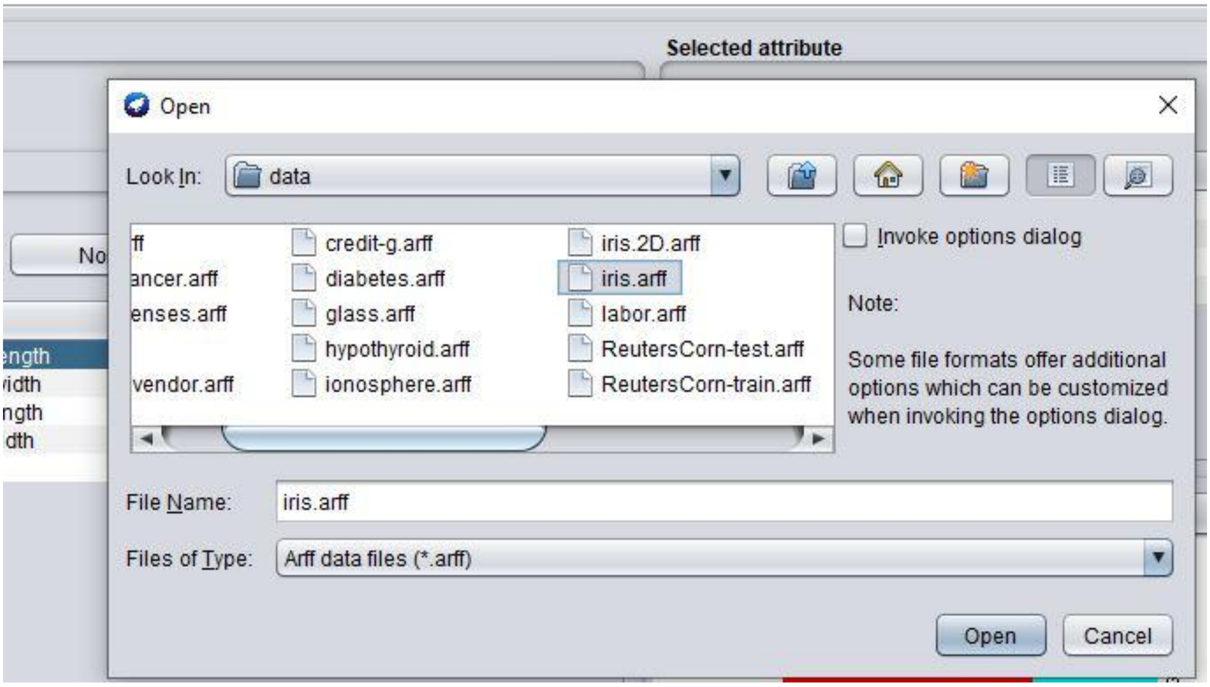
Thus, from this practical we conclude that to apply **minimax algorithm** in two-player games, we are going to assume that X is a maximizing player and O is a minimizing player. The maximizing player will try to maximize its score or in other words choose the move **with** the highest value

PRACTICAL 13

Aim: Perform classification on Iris dataset using neural network tools such as WEKA, ORANGE, NEUROINTELLIGENCE, and EasyNN.

Classification on Iris Dataset:

1) Loading the dataset



Selected attribute

Open

Look In: data

ff | credit-g.arff | iris.2D.arff
ancer.arff | diabetes.arff | iris.arff
enses.arff | glass.arff | labor.arff
| hypothyroid.arff | ReutersCorn-test.arff
vendor.arff | ionosphere.arff | ReutersCorn-train.arff

□ Invoke options dialog

Note:

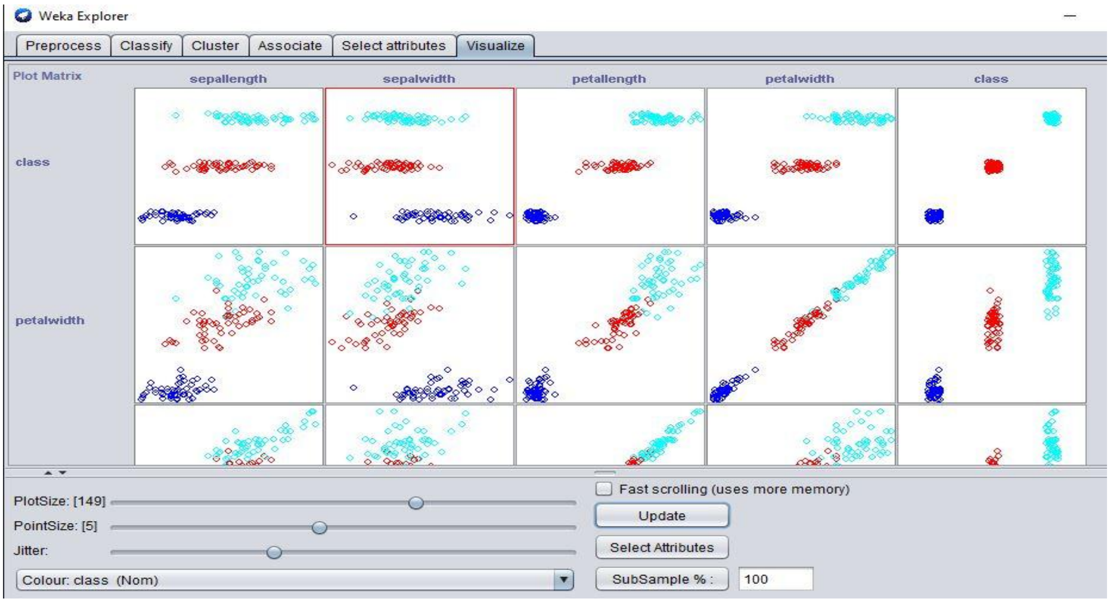
Some file formats offer additional options which can be customized when invoking the options dialog.

File Name: iris.arff

Files of Type: Arff data files (*.arff)

Open | Cancel

2) Editing the dataset



PlotSize: [149]

PointSize: [5]

Jitter:

Colour: class (Nom)

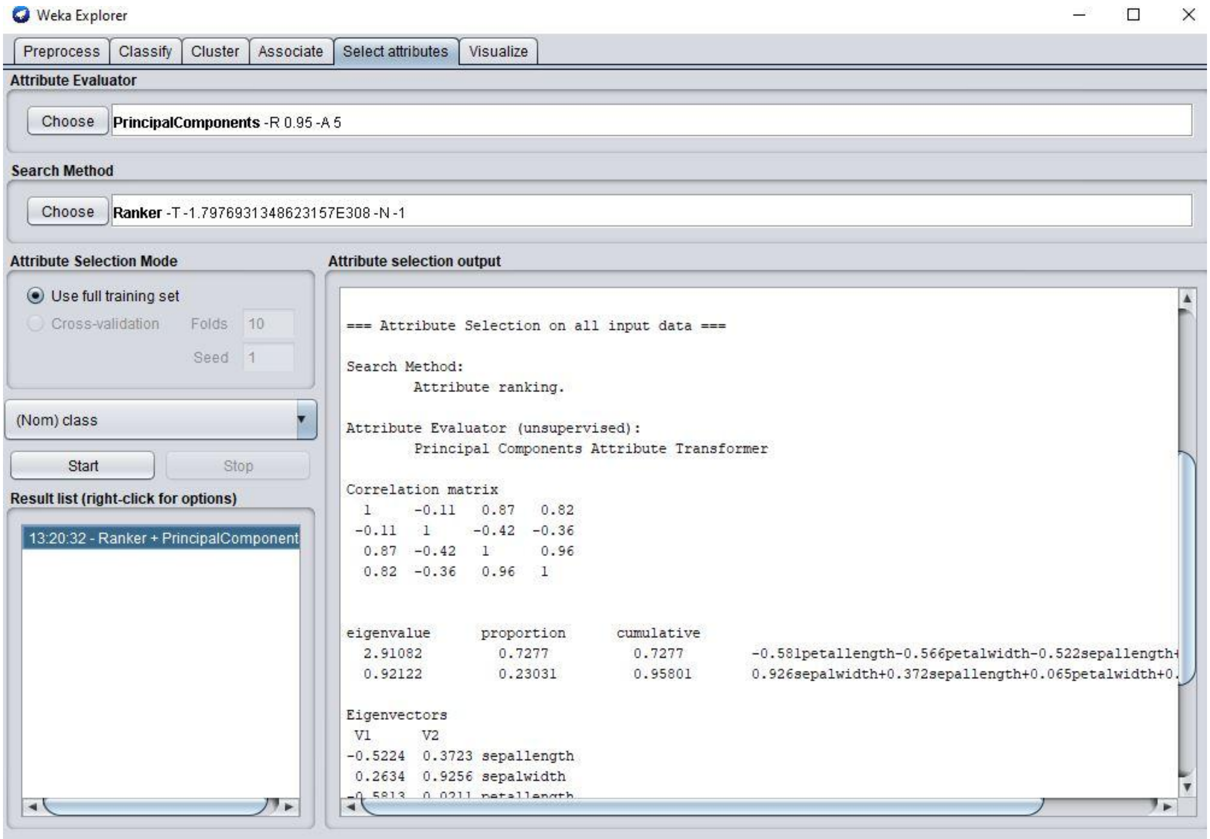
□ Fast scrolling (uses more memory)

Update

Select Attributes

SubSample % : 100

3) Selecting the Attributes



Attribute Evaluator: **PrincipalComponents** -R 0.95 -A 5

Search Method: **Ranker** -T -1.7976931348623157E308 -N -1

Attribute Selection Mode: ● Use full training set ○ Cross-validation Folds 10 Seed 1

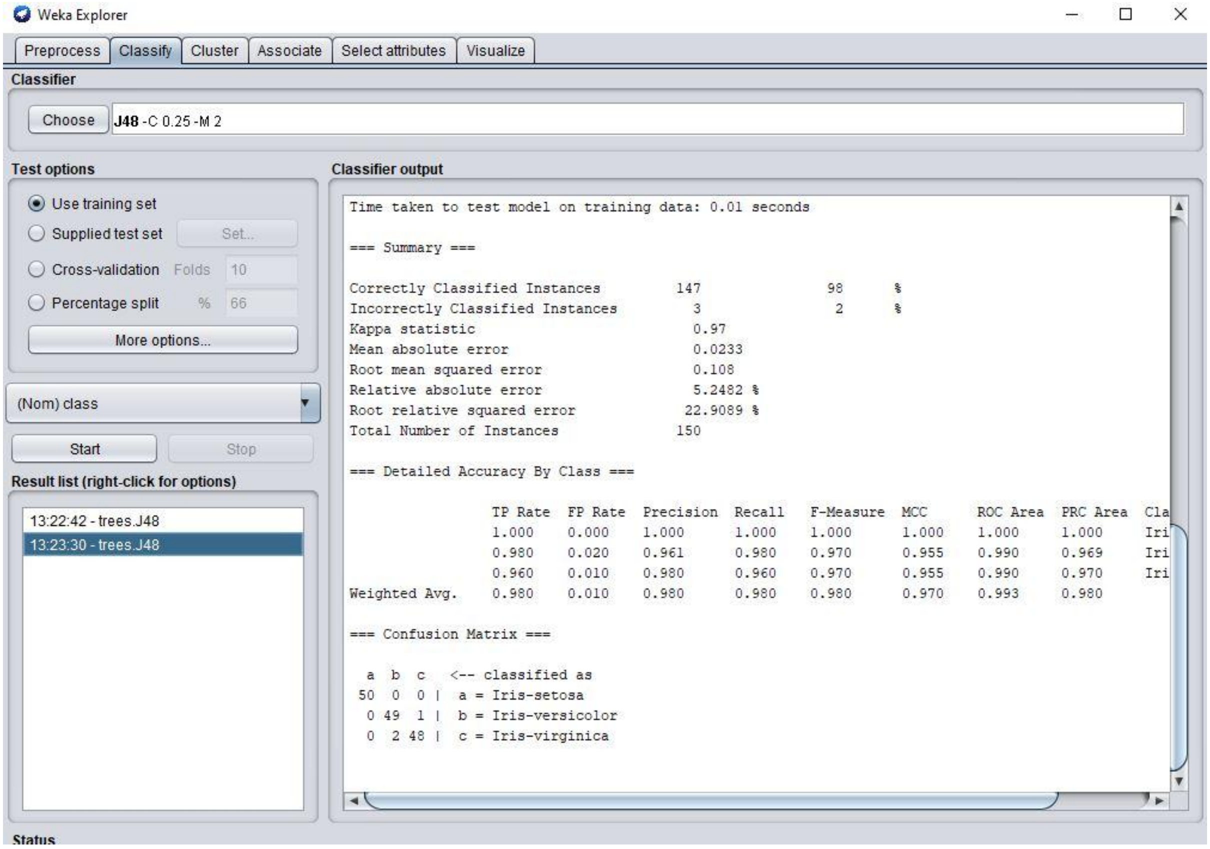
(Nom) class

Result list (right-click for options): 13:20:32 - Ranker + PrincipalComponent

Attribute selection output

```
=== Attribute Selection on all input data ===

Search Method:
        Attribute ranking.

Attribute Evaluator (unsupervised):
        Principal Components Attribute Transformer

Correlation matrix
  1      -0.11   0.87   0.82
 -0.11   1      -0.42  -0.36
  0.87  -0.42   1       0.96
  0.82  -0.36   0.96   1


eigenvalue      proportion      cumulative
  2.91082         0.7277          0.7277       -0.581petallength-0.566petalwidth-0.522sepallength+
  0.92122         0.23031         0.95801      0.926sepalwidth+0.372sepallength+0.065petalwidth+0.

Eigenvectors
 V1       V2
-0.5224  0.3723 sepallength
 0.2634  0.9256 sepalwidth
-0.5813  0.0211 petallength
```

4) Classifying the dataset - J48 – C 0.25 – M 2



Classifier: **J48** -C 0.25 -M 2

Test options: ● Use training set ○ Supplied test set ○ Cross-validation Folds 10 ○ Percentage split % 66

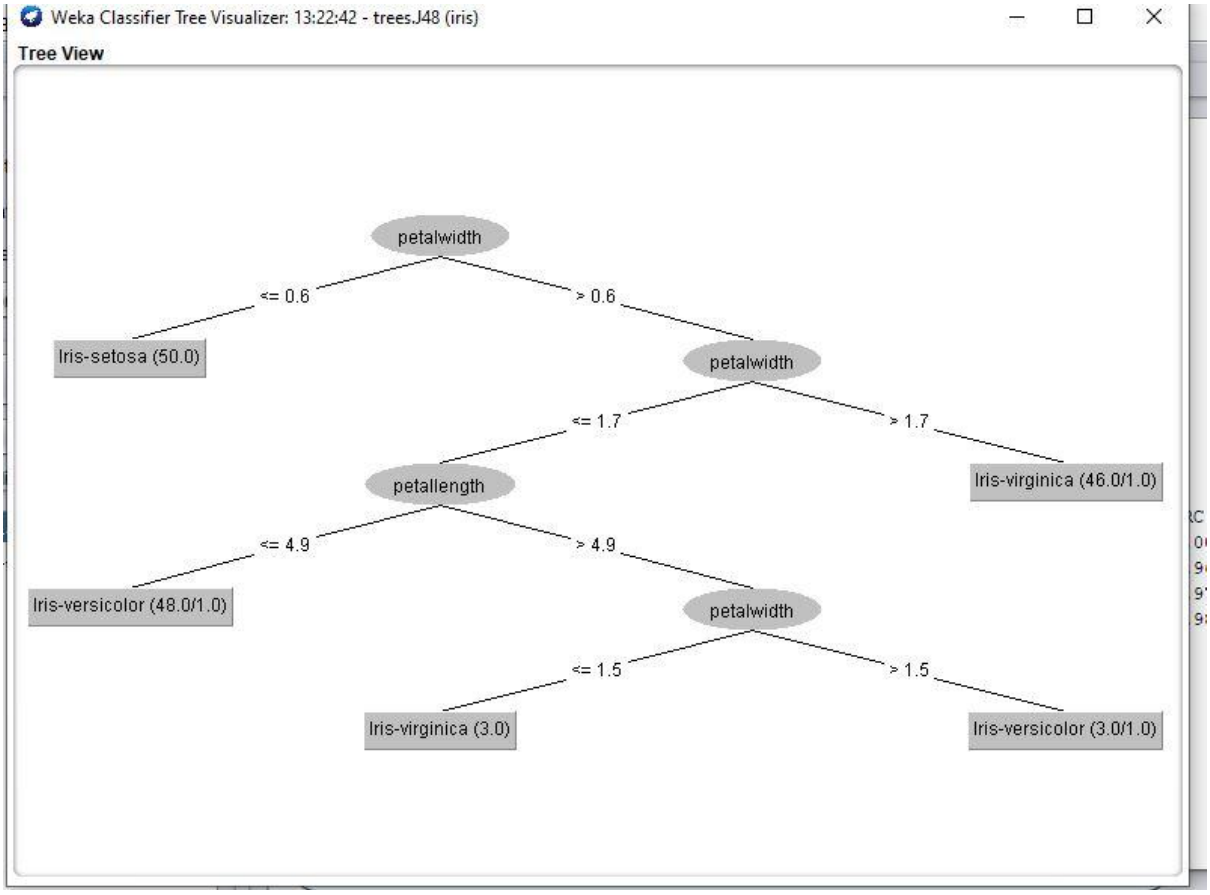
(Nom) class

Result list (right-click for options):
13:22:42 - trees.J48
13:23:30 - trees.J48

Classifier output

```
Time taken to test model on training data: 0.01 seconds

=== Summary ===

Correctly Classified Instances         147               98      %
Incorrectly Classified Instances         3                2      %
Kappa statistic                          0.97
Mean absolute error                      0.0233
Root mean squared error                  0.108
Relative absolute error                  5.2482 %
Root relative squared error             22.9089 %
Total Number of Instances              150

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Cla
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     Iri
                 0.980    0.020    0.961      0.980    0.970      0.955    0.990     0.969     Iri
                 0.960    0.010    0.980      0.960    0.970      0.955    0.990     0.970     Iri
Weighted Avg.    0.980    0.010    0.980      0.980    0.980      0.970    0.993     0.980

=== Confusion Matrix ===

  a  b  c   <-- classified as
 50  0  0 |  a = Iris-setosa
  0 49  1 |  b = Iris-versicolor
  0  2 48 |  c = Iris-virginica
```

5) Classification of Iris Dataset in Tree View


Weka Classifier Tree Visualizer: 13:22:42 - trees.J48 (iris)
Tree View
petalwidth
<= 0.6
> 0.6
Iris-setosa (50.0)
petalwidth
<= 1.7
> 1.7
petallength
Iris-virginica (46.0/1.0)
<= 4.9
> 4.9
Iris-versicolor (48.0/1.0)
petalwidth
<= 1.5
> 1.5
Iris-virginica (3.0)
Iris-versicolor (3.0/1.0)

Conclusion -

Thus, from this practical we conclude that neural network helps us to cluster and classify the dataset, they help to group unlabelled data according to similarities among the example inputs and they **classify** data when they have a labelled dataset to train on.

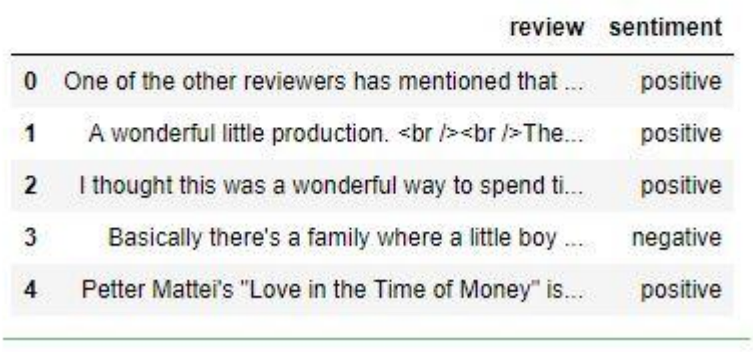
PRACTICAL 14

Aim: Perform Sentiment Analysis of movie reviews using nltk in python.

Sentiment Analysis of Movie Reviews

Program -

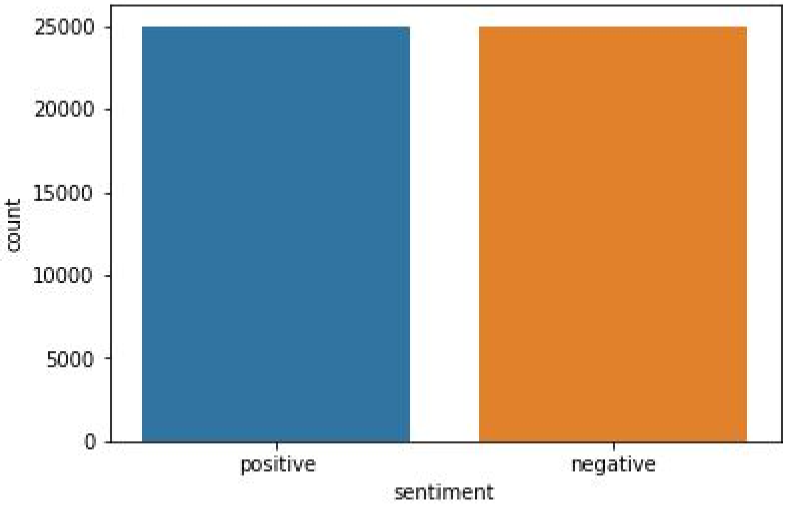
```
#Importing required libraries

import pandas as pd

import numpy as np

import re

import nltk

from nltk.corpus import stopwords

from numpy import array

from keras.preprocessing.text import one_hot

from keras.preprocessing.sequence import pad_sequences

from keras.models import Sequential

from keras.layers.core import Activation, Dropout, Dense

from keras.layers import Flatten

from keras.layers import GlobalMaxPooling1D

from keras.layers.embeddings import Embedding

from sklearn.model_selection import train_test_split

from keras.preprocessing.text import Tokenizer

movie_reviews = pd.read_csv("F:\5th Sem Study Material\AI\IMDB Dataset.csv")

movie_reviews.isnull().values.any()

movie_reviews.shape

movie_reviews.head()

#importing and analysing dataset

movie_reviews = pd.read_csv("E:\Datasets\IMDB Dataset.csv")

movie_reviews.isnull().values.any()

movie_reviews.shape
movie_reviews.head()
```



	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. <br /><br />The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive

#distribution of positive and negative sentiment in our dataset

```
import seaborn as sns

sns.countplot(x='sentiment', data=movie_reviews)
```



#Data Processing

```
def preprocess_text(sen):
    # Removing html tags
    sentence = remove_tags(sen)

    # Remove punctuations and numbers
    sentence = re.sub('[^a-zA-Z]', ' ', sentence)
    # Single character removal
    sentence = re.sub(r"\s+[a-zA-Z]\s+", ' ', sentence)

    # Removing multiple spaces
    sentence = re.sub(r'\s+', ' ', sentence)

    return sentence
TAG_RE = re.compile(r'<[^>]+>')

def remove_tags(text):
    return TAG_RE.sub('', text)
```

#Next, we will pre-process our reviews and will store them in a new list

```
X = []
sentences = list(movie_reviews['review'])
for sen in sentences:
    X.append(preprocess_text(sen))
```

#we will convert labels into digits

```
y = movie_reviews['sentiment']
y = np.array(list(map(lambda x: 1 if x=="positive" else 0, y)))
```

#dividing our dataset into train and test sets

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
```

#preparing the embedding layer

```
tokenizer = Tokenizer(num_words=5000)
tokenizer.fit_on_texts(X_train)
X_train = tokenizer.texts_to_sequences(X_train)
X_test = tokenizer.texts_to_sequences(X_test)
```

```
vocab_size = len(tokenizer.word_index) + 1

maxlen = 100

Xtrain = pad_sequences(X_train, padding='post', maxlen=maxlen)

X_test = pad_sequences(X_test, padding='post', maxlen=maxlen)
```

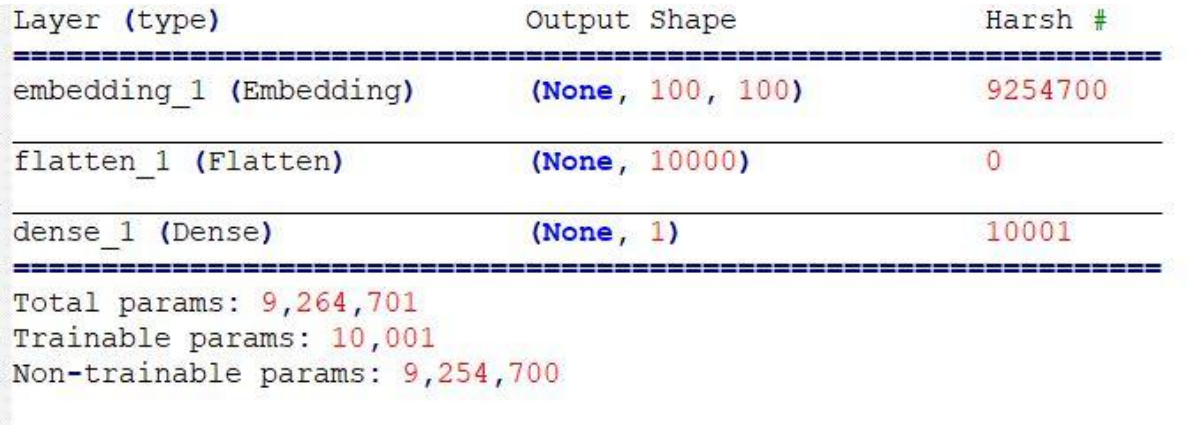
#GloVe embeddings to create our feature matrix

```
from numpy import array

from numpy import asarray

from numpy import zeros


embeddings_dictionary = dict()

glove_file = open('E:/Datasets/Word Embeddings/glove.6B.100d.txt', encoding="utf8")


for line in glove_file:

    records = line.split()

    word = records[0]

    vector_dimensions = asarray(records[1:], dtype='float32')

    embeddings_dictionary [word] = vector_dimensions

glove_file.close()

embedding_matrix = zeros((vocab_size, 100))

for word, index in tokenizer.word_index.items():

    embedding_vector = embeddings_dictionary.get(word)

    if embedding_vector is not None:

        embedding_matrix[index] = embedding_vector
```

#next, text classification with simple neural network

```
model = Sequential()
```

```
embedding_layer = Embedding(vocab_size, 100, weights=[embedding_matrix], input_length=maxlen , trainable=False)

model.add(embedding_layer)

model.add(Flatten())

model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])

print(model.summary())
```



```
Layer (type)                 Output Shape              Harsh #
=================================================================
embedding_1 (Embedding)      (None, 100, 100)          9254700
_________________________________________________________________
flatten_1 (Flatten)          (None, 10000)             0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 10001
=================================================================
Total params: 9,264,701
Trainable params: 10,001
Non-trainable params: 9,254,700
```

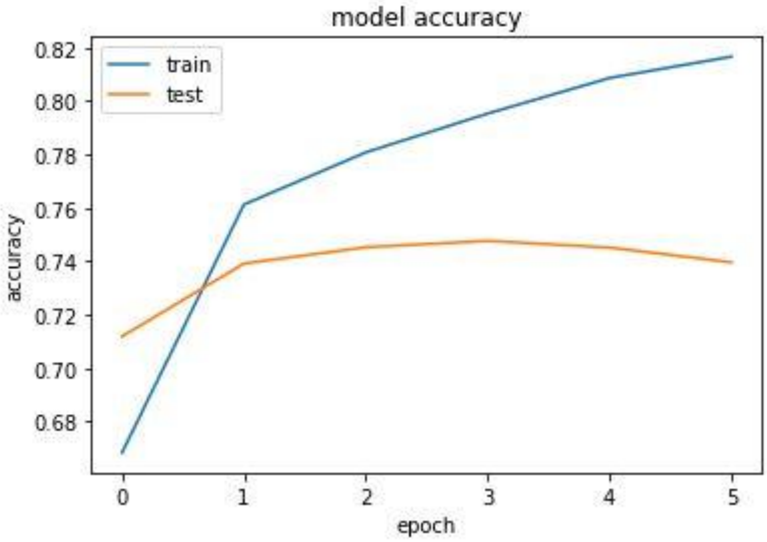
#train the model

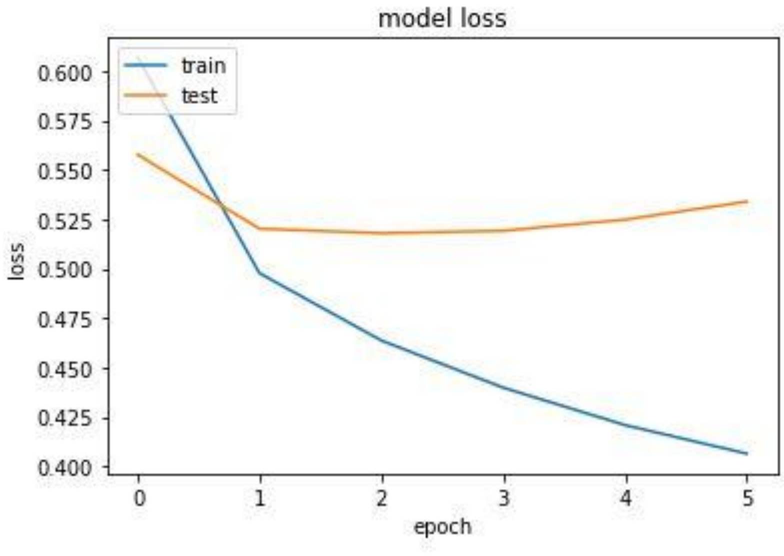
```
history = model.fit(X_train, y_train, batch_size=128, epochs=6, verbose=1, validation_split=0.2)

score = model.evaluate(X_test, y_test, verbose=1)

print("Test Score:", score[0])

print("Test Accuracy:", score[1])
```

#plotting the loss and accuracy differences for training and test sets

```
plt.title('model accuracy')

plt.ylabel('accuracy')

plt.xlabel('epoch')

plt.legend(['train','test'], loc='upper left')

plt.show()
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])

plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train','test'], loc='upper left')
plt.show()
```

Output





Conclusion -

Thus, from this practical we conclude that sentiment analysis tries to identify and extract opinions within a given text across blogs, reviews, social media, etc.