Anna University

for Affilated Engineering College - 2021 Regulation

PROGRESS THROUGH KNOWLEDGE

ECE Department

1st Semester | 2nd Semester | 3rd Semester | 4th Semester

5th Semester | 6th Semester | 7th Semester | 8th Semester

Click on Subject/Paper under Semester to enter.

1st Semester
- Professional English - I - HS3152
- Matrices and Calculus - MA3151
- Engineering Physics - PH3151
- Engineering Chemistry - CY3151
- Problem Solving and Python Programming - GE3151

2nd Semester
- Professional English - II - HS3252
- Statistics and Numerical Methods - MA3251
- Engineering Graphics - GE3251
- Physics for Electronics Engg - PH3254
- Basic Electrical & Instru Engg - BE3254
- Circuit Analysis - EC3251

3rd Semester
- Random Process and Linear Algebra - MA3355
- C Programming and Data Structures - CS3353
- Signals and Systems - EC3354
- Electronic Devices and Circuits - EC3353
- Control Systems - EC3351
- Digital Systems Design - EC3352

4th Semester
- Electromagnetic Fields - EC3452
- Networks and Security - EC3401
- Linear Integrated Circuits - EC3451
- Digital Signal Processing - EC3492
- Communication Systems - EC3491
- Environmental Sciences and Sustainability - GE3451

5th Semester
- Wireless Communication - EC3501
- VLSI and Chip Design - EC3552
- Transmission Lines and RF Systems - EC3551
- Elective 1
- Elective 2
- Elective 3

6th Semester
- Embedded Systems and IOT Design - ET3491
- Artificial Intelligence and Machine Learning - CS3491
- Open Elective-1
- Elective-4
- Elective-5
- Elective-6

7th Semester
- Human Values and Ethics - GE3791
- Open Elective 2
- Open Elective 3
- Open Elective 4

8th Semester
- Project Work / Intership

Ex.No:1(a)	**Implementation of Uniformed Search Algorithms (BFS& DFS)**
Date:	

Aim:

To implements the simple uniformed search algorithm breadth first search methods using python

Procedure:

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

Program:

```
from collections import deque

class Graph:

  def _init_(self, directed=True):

self.edges = {}

self.directed = directed

  def add_edge(self, node1, node2, __reversed=False):

    try: neighbors = self.edges[node1]

    except KeyError: neighbors = set()

neighbors.add(node2)

self.edges[node1] = neighbors

    if not self.directed and not __reversed: self.add_edge(node2, node1, True)

  def neighbors(self, node):

    try: return self.edges[node]

    except KeyError: return []

  def breadth_first_search(self, start, goal):
```

```
        found, fringe, visited, came_from = False, deque([start]), set([start]), {start: None}
        print('{:11s} | {}'.format('Expand Node', 'Fringe'))
print('-------------')
        print('{:11s} | {}'.format('-', start))
        while not found and len(fringe):
            current = fringe.pop()
            print('{:11s}'.format(current), end=' | ')
            if current == goal: found = True; break
            for node in self.neighbors(current):
                if node not in visited: visited.add(node); fringe.appendleft(node); came_from[node] =
current
print(', '.join(fringe))
        if found: print(); return came_from
        else: print('No path from {} to {}'.format(start, goal))
    @staticmethod
    def print_path(came_from, goal):
        parent = came_from[goal]
        if parent:
Graph.print_path(came_from, parent)
        else: print(goal, end='');return
print(' =>', goal, end='')
    def __str__(self):
        return str(self.edges)
graph = Graph(directed=False)
graph.add_edge('A', 'B')
graph.add_edge('A', 'S')
graph.add_edge('S', 'G')
```

```
graph.add_edge('S', 'C')

graph.add_edge('C', 'F')

graph.add_edge('G', 'F')

graph.add_edge('C', 'D')

graph.add_edge('C', 'E')

graph.add_edge('E', 'H')

graph.add_edge('G', 'H')

start, goal = 'A', 'H'

traced_path = graph.breadth_first_search(start, goal)

if (traced_path): print('Path:', end=' '); Graph.print_path(traced_path, goal);print()
```

Output:

```
====================== RESTART: F:/bfs.py ===============================

Expand Node | Fringe
--------------------
-           | A
A           | S, B
B           | S
S           | C, G
G           | H, F, C
C           | E, D, H, F
F           | E, D, H
H           |
Path: A => S => G => H
```

Result:

Thus, the program for breadth first search was executed and output is verified.

Ex.No:1(b)	**Implementation of uniformed search algorithms(DFS)**
Date:	

Aim:

To implements the simple uniformed search algorithm depth first search methods using python

Procedure:

1. Start by putting any one of the graph's vertex on top of the stack.
2. After that take the top item of the stack and add it to the visited list of the vertex.
3. Next, create a list of that adjacent node of the vertex. Add the ones which aren't in the visited list of vertexes to the top of the stack.
4. Lastly, keep repeating steps 2 and 3 until the stack is empty.

Program:

```
from collections import deque

class Graph:

    def _init_(self, directed=True):

self.edges = {}

self.directed = directed


    def add_edge(self, node1, node2, __reversed=False):

        try: neighbors = self.edges[node1]

        except KeyError: neighbors = set()

neighbors.add(node2)

self.edges[node1] = neighbors

        if not self.directed and not __reversed: self.add_edge(node2, node1, True)

    def neighbors(self, node):

        try: return self.edges[node]

        except KeyError: return []
```

```
    def breadth_first_search(self, start, goal):
        found, fringe, visited, came_from = False, deque([start]), set([start]), {start: None}
        print('{:11s} | {}'.format('Expand Node', 'Fringe'))
print('-------------')
        print('{:11s} | {}'.format('-', start))
        while not found and len(fringe):
            current = fringe.pop()
            print('{:11s}'.format(current), end=' | ')
            if current == goal: found = True; break
            for node in self.neighbors(current):
                if node not in visited: visited.add(node); fringe.append(node); came_from[node] =
current
print(', '.join(fringe))
        if found: print(); return came_from
        else: print('No path from {} to {}'.format(start, goal))


    @staticmethod
    def print_path(came_from, goal):
        parent = came_from[goal]
        if parent:
Graph.print_path(came_from, parent)
        else: print(goal, end='');return
print(' =>', goal, end='')
    def __str__(self):
        return str(self.edges)
graph = Graph(directed=False)
```

```
graph.add_edge('A', 'B')

graph.add_edge('A', 'S')

graph.add_edge('S', 'G')

graph.add_edge('S', 'C')

graph.add_edge('C', 'F')

graph.add_edge('G', 'F')

graph.add_edge('C', 'D')

graph.add_edge('C', 'E')

graph.add_edge('E', 'H')

graph.add_edge('G', 'H')

start, goal = 'A', 'H'

traced_path = graph.breadth_first_search(start, goal)

if (traced_path): print('Path:', end=' '); Graph.print_path(traced_path, goal);print()
```

Output:

```
        ====================== RESTART: F:/DFS.py ==================

Expand Node | Fringe
- - - - - - - - - - - - - -
-           | A
A           | B, S
S           | B, C, G
G           | B, C, F, H
H           |
Path: A => S => G => H
```

Result:

Thus, the program for depth first search was executed and output is verified.

Ex.No:2(a)	**Implementation of Informed search algorithms (A*, memory-bounded A*)**
Date:	

Aim:

To implements the simple informed search algorithm A* search methods using python

Procedure:

Step1: Place the starting node in the OPEN list.

Step 2: Check if the OPEN list is empty or not, if the list is empty then return failure and stops.

Step 3: Select the node from the OPEN list which has the smallest value of evaluation function (g+h), if node n is goal node then return success and stop, otherwise

Step 4: Expand node n and generate all of its successors, and put n into the closed list. For each successor n', check whether n' is already in the OPEN or CLOSED list, if not then compute evaluation function for n' and place into Open list.

Step 5: Else if node n' is already in OPEN and CLOSED, then it should be attached to the back pointer which reflects the lowest g(n') value.

Step 6: Return to Step 2.

Formula for A* Algorithm

h(n) = heuristic_value
g(n) = actual_cost
f(n) = actual_cost + heursitic_value

f(n) = g(n) + h(n)

Program:

```
def aStarAlgo(start_node, stop_node):
open_set = set(start_node) # {A}, len{open_set}=1
closed_set = set()
   g = {} # store the distance from starting node
   parents = {}
   g[start_node] = 0
   parents[start_node] = start_node # parents['A']='A"

   while len(open_set) >0 :
      n = None
```



```
        for v in open_set: # v='B'/'F'
            if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):
                n = v # n='A'

        if n == stop_node or Graph_nodes[n] == None:
            pass
        else:
            for (m, weight) in get_neighbors(n):
             # nodes 'm' not in first and last set are added to first
             # n is set its parent
                if m not in open_set and m not in closed_set:
open_set.add(m)      # m=B weight=6 {'F','B','A'} len{open_set}=2
                    parents[m] = n       # parents={'A':A,'B':A} len{parent}=2
                    g[m] = g[n] + weight # g={'A':0,'B':6, 'F':3} len{g}=2


            #for each node m,compare its distance from start i.e g(m) to the
            #from start through n node
                else:
                    if g[m] > g[n] + weight:
                    #update g(m)
                        g[m] = g[n] + weight
                    #change parent of m to n
                        parents[m] = n

                    #if m in closed set,remove and add to open
                        if m in closed_set:
closed_set.remove(m)
open_set.add(m)

        if n == None:
print('Path does not exist!')
            return None

        # if the current node is the stop_node
        # then we begin reconstructin the path from it to the start_node
        if n == stop_node:
            path = []

            while parents[n] != n:
path.append(n)
                n = parents[n]

path.append(start_node)

path.reverse()

print('Path found: {}'.format(path))
```

```
            return path


        # remove n from the open_list, and add it to closed_list
        # because all of his neighbors were inspected
    open_set.remove(n)# {'F','B'} len=2
    closed_set.add(n) #{A} len=1

    print('Path does not exist!')
        return None

#definefuction to return neighbor and its distance
#from the passed node
def get_neighbors(v):
    if v in Graph_nodes:
        return Graph_nodes[v]
    else:
        return None
#for simplicity we ll consider heuristic distances given
#and this function returns heuristic distance for all nodes

def heuristic(n):
H_dist = {
        'A': 10,
        'B': 8,
        'C': 5,
        'D': 7,
        'E': 3,
        'F': 6,
        'G': 5,
        'H': 3,
        'I': 1,
        'J': 0
    }

    return H_dist[n]

#Describe your graph here
Graph_nodes = {

    'A': [('B', 6), ('F', 3)],
    'B': [('C', 3), ('D', 2)],
    'C': [('D', 1), ('E', 5)],
    'D': [('C', 1), ('E', 8)],
    'E': [('I', 5), ('J', 5)],
    'F': [('G', 1),('H', 7)] ,
    'G': [('I', 3)],
    'H': [('I', 2)],
    'I': [('E', 5), ('J', 3)],
```

```
}
aStarAlgo('A', 'J')
```

Output:

```
===== RESTART: F:/AL&ML MANUAL/informed search1.py ==

Path found: ['A', 'F', 'G', 'I', 'J']
```

Result:

Thus, the program for A* search was executed and output is verified.

Ex.No:2(b)	**Implementation of informed search Algorithms (AO* Search)**
Date:	

Aim:

To implements the simple informed search algorithm AO* search methods using python

Procedure:

Step1: Proceeds life A*, expands best leaf until memory is full.
Step2: Cannot add new node without dropping an old one. (Always drops worst one)
Step3: Expands the best leaf and deletes the worst leaf.
Step4: If all have same f-value-selects same node for expansion and deletion.
Step4: SMA* is complete if any reachable solution.

Program:

```
class Graph:
   def __init__(self, graph, heuristicNodeList, startNode): #instantiate graph object with graph
topology, heuristic values, start node

self.graph = graph
self.H=heuristicNodeList
self.start=startNode
self.parent={}
self.status={}
self.solutionGraph={}

   def applyAOStar(self): # starts a recursive AO* algorithm
self.aoStar(self.start, False)

   def getNeighbors(self, v): # gets the Neighbors of a given node
      return self.graph.get(v,'')

   def getStatus(self,v): # return the status of a given node
      return self.status.get(v,0)

   def setStatus(self,v, val): # set the status of a given node
```



```
self.status[v]=val

    def getHeuristicNodeValue(self, n):
        return self.H.get(n,0) # always return the heuristic value of a given node

    def setHeuristicNodeValue(self, n, value):
self.H[n]=value # set the revised heuristic value of a given node


    def printSolution(self):
print("FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE
STARTNODE:",self.start)
print("------------------------------------------------------------")
        print(self.solutionGraph)
print("------------------------------------------------------------")

    def computeMinimumCostChildNodes(self, v): # Computes the Minimum Cost of child nodes
of a given node v
minimumCost=0
costToChildNodeListDict={}
costToChildNodeListDict[minimumCost]=[]
        flag=True
        for nodeInfoTupleList in self.getNeighbors(v): # iterate over all the set of child node/s
            cost=0
nodeList=[]
            for c, weight in nodeInfoTupleList:
                cost=cost+self.getHeuristicNodeValue(c)+weight
nodeList.append(c)

            if flag==True: # initialize Minimum Cost with the cost of first set of child node/s
minimumCost=cost
costToChildNodeListDict[minimumCost]=nodeList # set the Minimum Cost child node/s
                flag=False
            else: # checking the Minimum Cost nodes with the current Minimum Cost
                if minimumCost>cost:
minimumCost=cost
costToChildNodeListDict[minimumCost]=nodeList # set the Minimum Cost child node/s


        return minimumCost, costToChildNodeListDict[minimumCost] # return Minimum Cost
and Minimum Cost child node/s
```

```
    def aoStar(self, v, backTracking): # AO* algorithm for a start node and backTracking status
flag

print("HEURISTIC VALUES :", self.H)
print("SOLUTION GRAPH :", self.solutionGraph)
print("PROCESSING NODE :", v)

print("-----------------------------------------------------------------------------------------------")

        if self.getStatus(v) >= 0: # if status node v >= 0, compute Minimum Cost nodes of v
minimumCost, childNodeList = self.computeMinimumCostChildNodes(v)
self.setHeuristicNodeValue(v, minimumCost)
self.setStatus(v,len(childNodeList))

            solved=True # check the Minimum Cost nodes of v are solved

            for childNode in childNodeList:
self.parent[childNode]=v
                if self.getStatus(childNode)!=-1:
                    solved=solved & False

            if solved==True: # if the Minimum Cost nodes of v are solved, set the current node status
as solved(-1)
self.setStatus(v,-1)
self.solutionGraph[v]=childNodeList # update the solution graph with the solved nodes which
may be a part of solution


            if v!=self.start: # check the current node is the start node for backtracking the current
node value
self.aoStar(self.parent[v], True) # backtracking the current node value with backtracking status
set to true

             if backTracking==False: # check the current call is not for backtracking
                 for childNode in childNodeList: # for each Minimum Cost child node
  self.setStatus(childNode,0) # set the status of child node to 0(needs exploration)
self.aoStar(childNode, False) # Minimum Cost child node is further explored with backtracking
status as false
```

```
h1 = {'A': 1, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J':1, 'T': 3}
graph1 = {
    'A': [[('B', 1), ('C', 1)], [('D', 1)]],
    'B': [[('G', 1)], [('H', 1)]],
    'C': [[('J', 1)]],
    'D': [[('E', 1), ('F', 1)]],
    'G': [[('I', 1)]]
}
G1= Graph(graph1, h1, 'A')
G1.applyAOStar()
G1.printSolution()

h2 = {'A': 1, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7} # Heuristic values of Nodes
graph2 = { # Graph of Nodes and Edges
    'A': [[('B', 1), ('C', 1)], [('D', 1)]], # Neighbors of Node 'A', B, C & D with repective weights
    'B': [[('G', 1)], [('H', 1)]], # Neighbors are included in a list of lists
    'D': [[('E', 1), ('F', 1)]] # Each sublist indicate a "OR" node or "AND" nodes
}

G2 = Graph(graph2, h2, 'A') # Instantiate Graph object with graph, heuristic values and start
Node
G2.applyAOStar() # Run the AO* algorithm
G2.printSolution() # print the solution graph as AO* Algorithm search
```

OUTPUT:

```
==== RESTART: F:/AL&ML MANUAL/AO Search.py =======

HEURISTIC VALUES : {'A': 1, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1, 'T': 3}
SOLUTION GRAPH : {}
PROCESSING NODE : A
-----------------------------------------------------------
HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1, 'T':
3}
SOLUTION GRAPH : {}
PROCESSING NODE : B
-----------------------------------------------------------
HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1, 'T':
3}
SOLUTION GRAPH : {}
```

PROCESSING NODE : A

HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1, 'T': 3}
SOLUTION GRAPH : {}
PROCESSING NODE : G

HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J': 1, 'T': 3}
SOLUTION GRAPH : {}
PROCESSING NODE : B

HEURISTIC VALUES : {'A': 10, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J': 1, 'T': 3}
SOLUTION GRAPH : {}
PROCESSING NODE : A

HEURISTIC VALUES : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J': 1, 'T': 3}
SOLUTION GRAPH : {}
PROCESSING NODE : I

HEURISTIC VALUES : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 0, 'J': 1, 'T': 3}
SOLUTION GRAPH : {'I': []}
PROCESSING NODE : G

HEURISTIC VALUES : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1, 'T': 3}
SOLUTION GRAPH : {'I': [], 'G': ['I']}
PROCESSING NODE : B

HEURISTIC VALUES : {'A': 12, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1, 'T': 3}
SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}
PROCESSING NODE : A

HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1, 'T': 3}
SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}
PROCESSING NODE : C

HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1, 'T': 3}

SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}
PROCESSING NODE : A

HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1, 'T': 3}
SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}
PROCESSING NODE : J

HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 0, 'T': 3}
SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G'], 'J': []}
PROCESSING NODE : C

HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 1, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 0, 'T': 3}
SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G'], 'J': [], 'C': ['J']}
PROCESSING NODE : A

FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE STARTNODE: A

{'I': [], 'G': ['I'], 'B': ['G'], 'J': [], 'C': ['J'], 'A': ['B', 'C']}

HEURISTIC VALUES : {'A': 1, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH : {}
PROCESSING NODE : A

HEURISTIC VALUES : {'A': 11, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH : {}
PROCESSING NODE : D

HEURISTIC VALUES : {'A': 11, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH : {}
PROCESSING NODE : A

HEURISTIC VALUES : {'A': 11, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH : {}
PROCESSING NODE : E

HEURISTIC VALUES : {'A': 11, 'B': 6, 'C': 12, 'D': 10, 'E': 0, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH : {'E': []}
PROCESSING NODE : D

HEURISTIC VALUES : {'A': 11, 'B': 6, 'C': 12, 'D': 6, 'E': 0, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH : {'E': []}
PROCESSING NODE : A


```
------------------------------------------------------------
HEURISTIC VALUES : {'A': 7, 'B': 6, 'C': 12, 'D': 6, 'E': 0, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH : {'E': []}
PROCESSING NODE : F
------------------------------------------------------------
HEURISTIC VALUES : {'A': 7, 'B': 6, 'C': 12, 'D': 6, 'E': 0, 'F': 0, 'G': 5, 'H': 7}
SOLUTION GRAPH : {'E': [], 'F': []}
PROCESSING NODE : D
------------------------------------------------------------
HEURISTIC VALUES : {'A': 7, 'B': 6, 'C': 12, 'D': 2, 'E': 0, 'F': 0, 'G': 5, 'H': 7}
SOLUTION GRAPH : {'E': [], 'F': [], 'D': ['E', 'F']}
PROCESSING NODE : A
------------------------------------------------------------
FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE STARTNODE: A
----------------------------------------
{'E': [], 'F': [], 'D': ['E', 'F'], 'A': ['D']}
```

Result:

Thus, the program for AO* search was executed and output is verified.

Ex.No:3	**Implement Navie Bayes Models**
Date:	

Aim:

To implement navie bayes using navie classifier methods

Procedure:

Step 1 - Import basic libraries.
Step 2 - Importing the dataset.
Step 3 - Data preprocessing.
Step 4 - Training the model.
Step 5 - Testing and evaluation of the model.
Step 6 - Visualizing the model.

Program:

```
# import necessary libraries
importpandasaspd
fromsklearnimporttree
fromsklearn.preprocessingimportLabelEncoder
fromsklearn.naive_bayesimportGaussianNB

# Load Data from CSV
data=pd.read_csv('tennisdata.csv')
print("The first 5 Values of data is :\n", data.head())
```

```
The first 5 Values of data is :
   Outlook Temperature Humidity  Windy PlayTennis
0    Sunny       Hot     High    Weak        No

1    Sunny       Hot     High  Strong        No

2  Overcast      Hot     High    Weak       Yes

3     Rain      Mild     High    Weak       Yes

4     Rain      Cool   Normal    Weak       Yes
```

```
# obtain train data and train output
X=data.iloc[:, :-1]
```

```
print("\nThe First 5 values of the train data is\n", X.head())
```

```
The First 5 values of the train data is
    Outlook Temperature Humidity   Windy
0     Sunny         Hot     High    Weak
1     Sunny         Hot     High  Strong
2  Overcast         Hot     High    Weak
3      Rain        Mild     High    Weak
4      Rain        Cool   Normal    Weak
```

```
y=data.iloc[:, -1]
print("\nThe First 5 values of train output is\n", y.head())
```

```
The First 5 values of train output is
 0     No
1     No
2    Yes
3    Yes
4    Yes
Name: PlayTennis, dtype: object
```

```
# convert them in numbers
le_outlook=LabelEncoder()
X.Outlook=le_outlook.fit_transform(X.Outlook)

le_Temperature=LabelEncoder()
X.Temperature=le_Temperature.fit_transform(X.Temperature)

le_Humidity=LabelEncoder()
X.Humidity=le_Humidity.fit_transform(X.Humidity)

le_Windy=LabelEncoder()
X.Windy=le_Windy.fit_transform(X.Windy)

print("\nNow the Train output is\n", X.head())
```

```
Now the Train output is
Outlook  Temperature  Humidity  Windy
0       2          1        0      1
1       2          1        0      0
2       0          1        0      1
3       1          2        0      1
4       1          0        1      1
```

```
le_PlayTennis=LabelEncoder()
y=le_PlayTennis.fit_transform(y)
print("\nNow the Train output is\n",y)
Now the Train output is
 [0 0 1 1 1 0 1 0 1 1 1 1 1 0]
fromsklearn.model_selectionimporttrain_test_split
X_train, X_test, y_train, y_test=train_test_split(X,y, test_size=0.20)

classifier=GaussianNB()
classifier.fit(X_train, y_train)

fromsklearn.metricsimportaccuracy_score
print("Accuracy is:", accuracy_score(classifier.predict(X_test), y_test))
```

Output:

Accuracy is: 0.3333333333333333

Result:

Thus, the naive baye program was executed and output is verified.

Ex.No:4	**Implement Bayesian Networks**
Date:	

Aim:

To write a python program to find Bayesian networks

Procedure:

1. age: age in years
2. sex: sex (1 = male; 0 = female)
3. cp: chest pain type
    - Value 1: typical angina
    - Value 2: atypical angina
    - Value 3: non-anginal pain
    - Value 4: asymptomatic
4. trestbps: resting blood pressure (in mm Hg on admission to the hospital)
5. chol: serum cholestoral in mg/dl
6. fbs: (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
7. restecg: resting electrocardiographic results
    - Value 0: normal
    - Value 1: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV)
    - Value 2: showing probable or definite left ventricular hypertrophy by Estes' criteria
8. thalach: maximum heart rate achieved
9. exang: exercise induced angina (1 = yes; 0 = no)
10. oldpeak = ST depression induced by exercise relative to rest
11. slope: the slope of the peak exercise ST segment
    - Value 1: upsloping
    - Value 2: flat
    - Value 3: downsloping
12. ca = number of major vessels (0-3) colored by flourosopy
13. thal: 3 = normal; 6 = fixed defect; 7 = reversable defect
14. Heartdisease: It is integer valued from 0 (no presence) to 4. Diagnosis of heart disease (angiographic disease status)

Program:

```
import pandas as pd
data=pd.read_csv("heartdisease.csv")
heart_disease=pd.DataFrame(data)
print(heart_disease)
```

	age	Gender	Family	diet	Lifestyle	cholestrol	heartdisease
0	0	0	1	1	3	0	1
1	0	1	1	1	3	0	1
2	1	0	0	0	2	1	1
3	4	0	1	1	3	2	0
4	3	1	1	0	0	2	0
5	2	0	1	1	1	0	1
6	4	0	1	0	2	0	1
7	0	0	1	1	3	0	1
8	3	1	1	0	0	2	0
9	1	1	0	0	0	2	1
10	4	1	0	1	2	0	1
11	4	0	1	1	3	2	0
12	2	1	0	0	0	0	0
13	2	0	1	1	1	0	1
14	3	1	1	0	0	1	0
15	0	0	1	0	0	2	1
16	1	1	0	1	2	1	1
17	3	1	1	1	0	1	0
18	4	0	1	1	3	2	0

In [2]:

```
frompgmpy.modelsimportBayesianModel
model=BayesianModel([
('age','Lifestyle'),
('Gender','Lifestyle'),
('Family','heartdisease'),
('diet','cholestrol'),
('Lifestyle','diet'),
('cholestrol','heartdisease'),
('diet','cholestrol')
])

frompgmpy.estimatorsimportMaximumLikelihoodEstimator
model.fit(heart_disease, estimator=MaximumLikelihoodEstimator)

frompgmpy.inferenceimportVariableElimination
HeartDisease_infer=VariableElimination(model)
```

In [3]:

```
print('For age Enter { SuperSeniorCitizen:0, SeniorCitizen:1, MiddleAged:2, Youth:3, Teen:4 }')
print('For Gender Enter { Male:0, Female:1 }')
print('For Family History Enter { yes:1, No:0 }')
print('For diet Enter { High:0, Medium:1 }')
```

```
print('For lifeStyle Enter { Athlete:0, Active:1, Moderate:2, Sedentary:3 }')
print('For cholesterol Enter { High:0, BorderLine:1, Normal:2 }')

q =HeartDisease_infer.query(variables=['heartdisease'], evidence={
    'age':int(input('Enter age :')),
    'Gender':int(input('Enter Gender :')),
    'Family':int(input('Enter Family history :')),
    'diet':int(input('Enter diet :')),
    'Lifestyle':int(input('Enter Lifestyle :')),
    'cholestrol':int(input('Enter cholestrol :'))
    })

print(q['heartdisease'])
```

Output:

```
For age Enter { SuperSeniorCitizen:0, SeniorCitizen:1, MiddleAged:2, Youth:3, Teen:4 }
For Gender Enter { Male:0, Female:1 }
For Family History Enter { yes:1, No:0 }
For diet Enter { High:0, Medium:1 }
For lifeStyle Enter { Athlete:0, Active:1, Moderate:2, Sedentary:3 }
For cholesterol Enter { High:0, BorderLine:1, Normal:2 }
Enter age :1
Enter Gender :1
Enter Family history :0
Enter diet :1
Enter Lifestyle :0
Enter cholestrol :1
+-----------+--------------+
| heartdisease   |   phi(heartdisease) |
+===============+=====================+
| heartdisease_0 |            0.0000 |
+-----------+--------------+
| heartdisease_1 |            1.0000 |
+-----------+--------------+
```

Result:

Thus, the Bayesian networks program was executed and output is verified.

Ex.No:5(a)	**Build regression models**
Date:	

Aim:

To write a python program regression model using linear regression model

Procedure:

Step 1: Data Pre Processing

- Importing The Libraries.
- Importing the Data Set.
- Encoding the Categorical Data.
- Avoiding the Dummy Variable Trap.
- Splitting the Data set into Training Set and Test Set.

Step 2: Fitting Multiple Linear Regression to the Training set

Step 3: Predict the Test set results.

Program:

```
import pandas as pd
importnumpyas np
fromsklearnimportlinear_model
```

In [2]:

```
df=pd.read_csv('homeprices.csv')
df
```

Out[2]:

	area	bedrooms	age	price
0	2600	3.0	20	550000
1	3000	4.0	15	565000
2	3200	NaN	18	610000
3	3600	3.0	30	595000

	area	bedrooms	age	price
4	4000	5.0	8	760000
5	4100	6.0	8	810000

Data Preprocessing: Fill NA values with median value of a column

In [3]:
```
df.bedrooms.median()
```

Out[3]:
4.0

In [5]:
```
df.bedrooms=df.bedrooms.fillna(df.bedrooms.median())
df
```

Out[5]:

	area	bedrooms	age	price
0	2600	3.0	20	550000
1	3000	4.0	15	565000
2	3200	4.0	18	610000
3	3600	3.0	30	595000
4	4000	5.0	8	760000
5	4100	6.0	8	810000

In [6]:
```
reg =linear_model.LinearRegression()
reg.fit(df.drop('price',axis='columns'),df.price)
```

Out[6]:
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,

```
normalize=False)
```

In [7]:
```
reg.coef_
```

Out[7]:
```
array([  112.06244194, 23388.88007794, -3231.71790863])
```

In [8]:
```
reg.intercept_
```

Out[8]:
```
221323.00186540408
```

Find price of home with 3000 sqr ft area, 3 bedrooms, 40 year old

In [9]:
```
reg.predict([[3000, 3, 40]])
```

Out[9]:
```
array([498408.25158031])
```

In [10]:
```
112.06244194*3000 + 23388.88007794*3 +-3231.71790863*40 + 221323.00186540384
```

Out[10]:
```
498408.25157402386
```

Find price of home with 2500 sqr ft area, 4 bedrooms, 5 year old

In [11]:
```
reg.predict([[2500, 4, 5]])
```

Out[11]:
```
array([578876.03748933])
```

Result:

Thus, the python program for regression model was executed successfully.

Ex.No:5(b)	**Build regression models (Logistics Regression Model)**
Date:	

Aim:

To write a python program regression model using logistics regression model

Procedure:

Step 1: Data Pre Processing

- Importing The Libraries.
- Importing the Data Set.

Step 2:Extracting Independent and dependent Variable

Step3:Splitting the dataset into training and test set.

Step4:feature Scaling

Step5:Fitting Logistic Regression to the training set

Step6:Predicting the test set result

Program:

```
import pandas as pd
from matplotlib importpyplotasplt
%matplotlib inline
```

In [16]:

```
df=pd.read_csv("insurance_data.csv")
df.head()
```

Out[16]:

	age	bought_insurance
0	22	0
1	25	0
2	47	1

	age	bought_insurance
3	52	0
4	46	1

In [17]:

```
plt.scatter(df.age,df.bought_insurance,marker='+',color='red')
```

Out[17]:

<matplotlib.collections.PathCollection at 0x20a8cb15d30>

In [18]:

```
from sklearn.model_selection import train_test_split
```

In [29]:

```
X_train, X_test, y_train, y_test=train_test_split(df[['age']],df.bought_insurance,train_size=0.8)
```

In [30]:

```
X_test
```

Out[30]:

	age
4	46
8	62
26	23
17	58
24	50
25	54

In [31]:

```
from sklearn.linear_model import LogisticRegression
model =LogisticRegression()
```

In [66]:

```
model.fit(X_train, y_train)
```


C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning. FutureWarning)

Out[66]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
intercept_scaling=1, max_iter=100, multi_class='warn',
n_jobs=None, penalty='l2', random_state=None, solver='warn',
tol=0.0001, verbose=0, warm_start=False)
```

In [9]:

```
X_test
```

Out[9]:

	age
16	25
21	26
2	47

In [39]:

```
y_predicted=model.predict(X_test)
```

In [33]:

```
model.predict_proba(X_test)
```

Out[33]:

```
array([[0.40569485, 0.59430515],
       [0.26002994, 0.73997006],
       [0.63939494, 0.36060506],
       [0.29321765, 0.70678235],
       [0.36637568, 0.63362432],
       [0.32875922, 0.67124078]])
```

In [34]:

```
model.score(X_test,y_test)
```

Out[34]:

1.0

In [40]:

```
y_predicted
```

Out[40]:

```
array([1, 1, 0, 1, 1, 1], dtype=int64)
```

In [37]:

```
X_test
```

Out[37]:

	age
4	46
8	62
26	23
17	58
24	50
25	54

model.coef_ indicates value of m in y=m*x + b equation

In [67]:
```
model.coef_
```

Out[67]:
```
array([[0.04150133]])
```

model.intercept_ indicates value of b in y=m*x + b equation

In [68]:
```
model.intercept_
```

Out[68]:
```
array([-1.52726963])
```

Lets defined sigmoid function now and do the math with hand

In [43]:
```
import math
def sigmoid(x):
return 1 / (1 +math.exp(-x))
```

In [75]:
```
defprediction_function(age):
    z = 0.042 * age - 1.53 # 0.04150133 ~ 0.042 and -1.52726963 ~ -1.53
    y = sigmoid(z)
return y
```

In [76]:
```
age = 35
prediction_function(age)
```

Out[76]:
```
0.4850044983805899
```

0.485 is less than 0.5 which means person with 35 age will *not* buy insurance

In [77]:

```
age = 43
prediction_function(age)
```

Out[77]:

0.568565299077705

0.485 is more than 0.5 which means person with 43 will buy the insurance

Result:

Thus, the python program for logistics regression model was executed successfully.

Ex.No:6 (a)	**Build Decision Trees**
Date:	

Aim:

To write a python program Decision tree using Gaussian classifier.

Procedure:

1. import Python library packages
2. reading the dataset from the local folder
3. printing first 5 rows
4. As all the columns are categorical, check for unique values of each column
5. Check how these unique categories are distributed among the columns
6. Heatmap of the columns on dataset with each other. It shows Pearson's correlation coefficient of column w.r.t other columns.
7. As scikit-learn algorithms do not generally work with string values, I've converted string categories to integers.
8. printing the first 5 rows
9. X is the dataframe containing input data / features
10. y is the series which has results which are to be predicted.
11. Import train_test_split function
12. Split dataset into training set and test set
13. Create a Gaussian Classifier
14. Train the model using the training sets y_pred=model.predict(X_test
15. Import scikit-learn metrics module for accuracy calculation
16. Model Accuracy, how often is the classifier correct?

Program:

```
#import Python library packages

    import numpy as np

    import matplotlib.pyplot as plt

    import pandas as pd

    from sklearn.metrics import classification_report
```

```
from sklearn.metrics import confusion_matrix

from sklearn import preprocessing

from sklearn.preprocessing import LabelEncoder

from sklearn.model_selection import cross_val_score

import seaborn as sns


#reading the dataset from the local folder

data=pd.read_csv('covid19.csv')
```

In[2]:

```
#printing first 5 rows

data.head()
```

Output[2]:

	patient_id	global_num	sex	birth_year	age	country	province	city	latitude	longitude	infection_case	infection_order	state	Label
0	1000000001	2	male	1964	50s	Korea	Seoul	Gangseo-gu	37.460459	126.440680	overseas inflow	1.0	released	0
1	1000000002	5	male	1987	30s	Korea	Seoul	Jungnang-gu	37.478832	126.668558	overseas inflow	1.0	released	0
2	1000000003	6	male	1964	50s	Korea	Seoul	Jongno-gu	37.562143	126.801884	contact with patient	2.0	released	0

	patient_id	global_num	sex	birth_year	age	country	province	city	latitude	longitude	infection_case	infection_order	state	Label
3	1000000004	7	male	1991	20s	Korea	Seoul	Mapo-gu	37.567454	127.005627	overseas inflow	1.0	released	0
4	1000000005	9	female	1992	20s	Korea	Seoul	Seongbuk-gu	37.460459	126.440680	contact with patient	2.0	released	0

In[3]

```
#As all the columns are categorical, check for unique values of each column
for i in data.columns:
    print(data[i].unique(),"\t",data[i].nunique())
```

Output[3]:

```
[1000000001 1000000002 1000000003 1000000004 1000000005 1000000006
 1000000007 1000000008 1000000009 1000000010 1000000011 1000000012
 1000000013 1000000014 1000000015 1000000016 1000000017 1000000018
 1000000019 1000000020 1000000021 1000000022 1000000023 1000000024
 1000000025 1000000026 1000000027 1000000028 1000000029 1000000030
 1000000031 1000000032 1000000033 1000000034 1000000035 1000000036
 1000000037 1000000038 1000000039 1000000040 1000000041 1000000042
 1000000043 1000000044 1000000045 1000000046 1000000047 1000000048
 1000000049 1000000050 1000000051 1000000052 1000000053 1000000054
 1000000055 1000000056 1000000057 1000000058 1000000059 1000000060
 1000000061 1000000062 1000000063 1000000064 1000000065 1000000066
 1000000067 1000000068 1000000069 1000000070 1000000071 1000000072
 1000000073 1000000074 1000000075 1000000076 1000000077 1000000078
 1000000079 1000000080 1000000081 1000000082 1000000083 1000000084
 1000000085 1000000086 1000000087 1000000088 1000000089 1000000090
 1000000091 1000000092 1000000093 1000000094 1000000095 1000000096
 1000000097 1000000098 1000000099 1000000100 1000000101 1000000102
 1000000103 1000000104 1000000105 1000000106 1000000107 1000000108
 1000000109 1000000110 1000000111 1000000112 1000000113 1000000114
```

1000000115 1000000116 1000000117 1000000118 1000000119 1000000120
1000000121 1000000122 1000000123 1000000124 1000000125 1000000126

In[4]

data.info()

Out[4]:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 160 entries, 0 to 159
Data columns (total 14 columns):
patient_id        160 non-null int64
global_num        160 non-null int64
sex               160 non-null object
birth_year        160 non-null int64
age               160 non-null object
country           160 non-null object
province          160 non-null object
city              160 non-null object
latitude          160 non-null float64
longitude         160 non-null float64
infection_case    160 non-null object
infection_order   18 non-null float64
state             160 non-null object
Label             160 non-null int64
dtypes: float64(3), int64(4), object(7)
memory usage: 17.6+ KB
```

In[5]

```
#Check how these unique categories are distributed among the columns
for i in data.columns:
    print(data[i].value_counts())
print()
```

Out[5]:

```
1000000160    1
1000000159    1
1000000058    1
1000000057    1
1000000056    1
1000000055    1
1000000054    1
```



```
1000000053    1
1000000052    1
1000000051    1
1000000050    1
1000000049    1
1000000048    1
1000000047    1
1000000046    1
1000000045    1
1000000044    1
1000000043    1
1000000042    1
1000000059    1
1000000060    1
1000000061    1
1000000071    1
1000000078    1
1000000077    1
1000000076    1
1000000075    1
1000000074    1
1000000073    1
1000000072    1
           ..
1000000090    1
1000000100    1
1000000089    1
1000000088    1
1000000087    1
1000000086    1
1000000085    1
1000000084    1
1000000083    1
1000000099    1
```

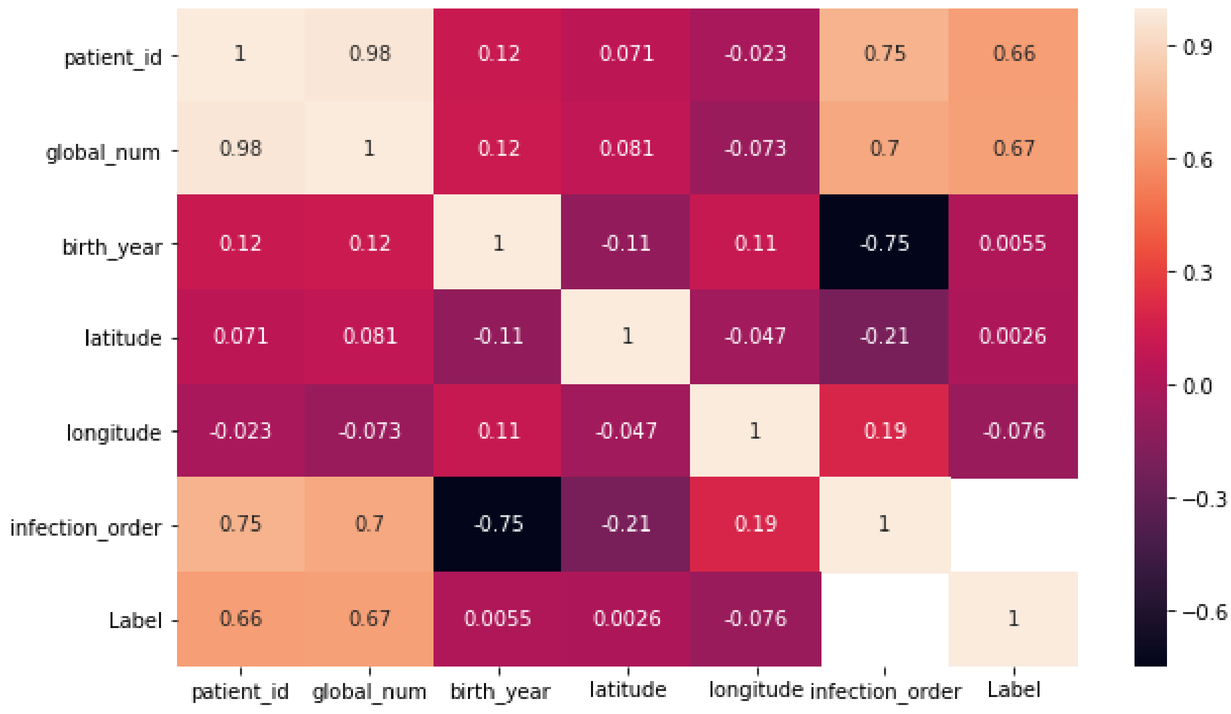
In[6]

#Heatmap of the columns on dataset with each other. It shows Pearson's correlation coefficient of column w.r.t other columns.

```
fig=plt.figure(figsize=(10,6))

sns.heatmap(data.corr(),annot=True)
```

Out[6]:

<matplotlib.axes._subplots.AxesSubplot at 0x1e3673ac748>




```
<matplotlib.axes._subplots.AxesSubplot at 0x1e3673ac748>
```

In[7]

```
#As scikit-learn algorithms do not generally work with string values, I've converted string categories to integers.
le=LabelEncoder()
for sex indata.columns:
    data[sex]=le.fit_transform(data[sex])
for age indata.columns:
    data[age]=le.fit_transform(data[age])
for country indata.columns:
    data[country]=le.fit_transform(data[country])
for province indata.columns:
    data[province]=le.fit_transform(data[province])
for city indata.columns:
    data[city]=le.fit_transform(data[city])
forinfection_caseindata.columns:
    data[infection_case]=le.fit_transform(data[infection_case])
for state indata.columns:
    data[state]=le.fit_transform(data[state])

#printing the first 5 rows
data.head()
```

Out[7]:

Out[11]:

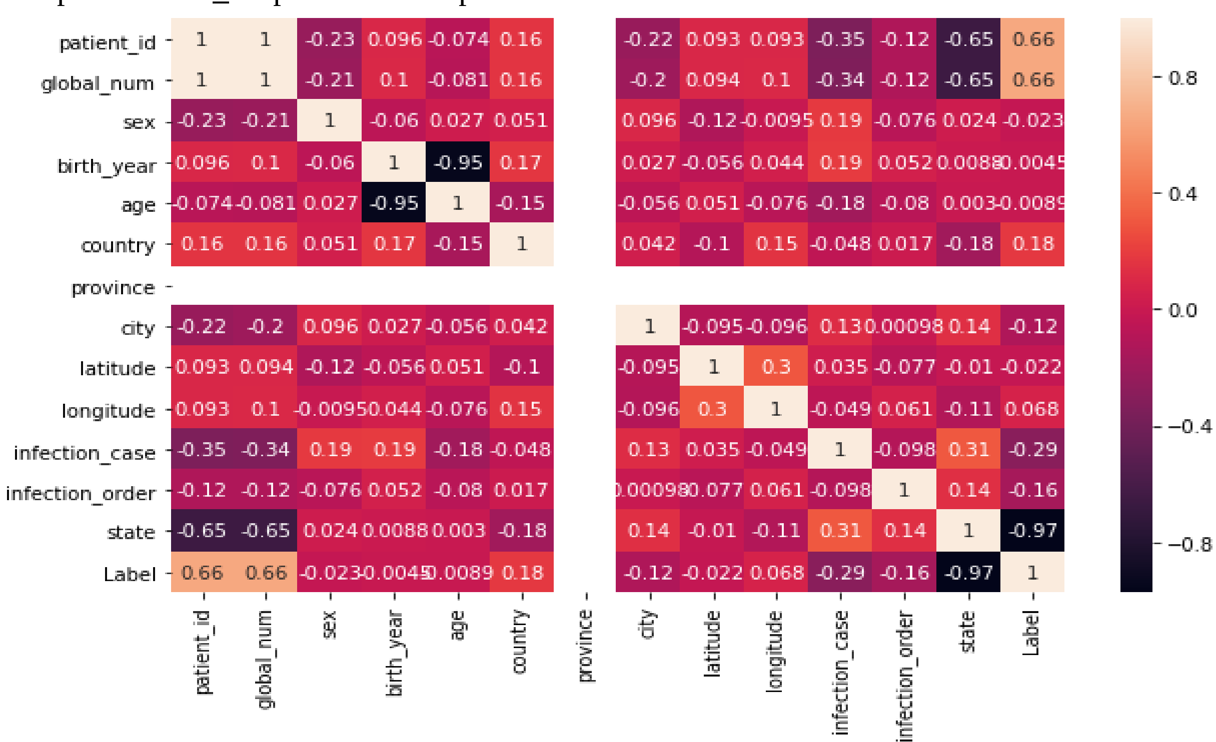
	patient_id	global_num	sex	birth_year	age	country	province	city	latitude	longitude	infection_case	infection_order	state	Label
0	0	0	1	22	4	1	0	7	37	2	7	0	2	0
1	1	1	1	45	2	1	0	14	39	5	7	0	2	0
2	2	2	1	22	4	1	0	12	64	24	5	1	2	0
3	3	3	1	49	1	1	0	15	72	54	7	0	2	0
4	4	4	0	50	1	1	0	19	37	2	5	1	2	0

In[8]

```
fig=plt.figure(figsize=(10,6))
sns.heatmap(data.corr(),annot=True)
```

Out[8]:

<matplotlib.axes._subplots.AxesSubplot at 0x1e3674c6d30>



In[9]
#X is the dataframe containing input data / features

#y is the series which has results which are to be predicted.
```
X=data[data.columns[:-1]]
y=data['Label']
X.head(2)
```

Out[9]:

	patient_id	global_num	sex	birth_year	age	country	province	city	latitude	longitude	infection_case	infection_order	state
0	0	0	1	22	4	1	0	7	37	2	7	0	2
1	1	1	1	45	2	1	0	14	39	5	7	0	2

In[10]
```
# Import train_test_split function
from sklearn.model_selection import train_test_split
# Split dataset into training set and test set
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.33) # 70% training and 30% test
from sklearn.tree import DecisionTreeClassifier
#Create a Gaussian Classifier
model=DecisionTreeClassifier()
#Train the model using the training sets y_pred=model.predict(X_test)
model.fit(X_train,y_train)
y_pred=model.predict(X_test)
#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
# Model Accuracy, how often is the classifier correct?
print("Accuracy for Decision Tree:",metrics.accuracy_score(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test,  y_pred))
```

Output[10]:
```
Accuracy for Decision Tree: 0.9433962264150944
[[18  0  0]
 [ 1 32  2]
 [ 0  0  0]]
              precision    recall  f1-score   support
```

```
           0       0.95      1.00      0.97        18
           1       1.00      0.91      0.96        35
           2       0.00      0.00      0.00         0

    accuracy                           0.94        53
   macro avg       0.65      0.64      0.64        53
weighted avg       0.98      0.94      0.96        53

print(cross_val_score(model,X,y,cv=10))
[0.94117647 0.875      0.9375     1.         0.9375     1.
 0.9375     1.         1.         0.93333333]
```

Result:

Thus, the python program for decision tree was executed successfully.

Ex.No:6 (b)	**Build Random Forest Tree**
Date:	

Aim:

To write a python program random forest tree using Gaussian classifier.

Procedure:

1. import Python library packages
2. reading the dataset from the local folder
3. printing first 5 rows
4. As all the columns are categorical, check for unique values of each column
5. Check how these unique categories are distributed among the columns
6. Heatmap of the columns on dataset with each other. It shows Pearson's correlation coefficient of column w.r.t other columns.
7. As scikit-learn algorithms do not generally work with string values, I've converted string categories to integers.
8. printing the first 5 rows
9. X is the dataframe containing input data / features
10. y is the series which has results which are to be predicted.
11. Import train_test_split function
12. Split dataset into training set and test set
13. Create a Gaussian Classifier
14. Train the model using the training sets y_pred=model.predict(X_test
15. Import scikit-learn metrics module for accuracy calculation
16. Model Accuracy, how often is the classifier correct?

Program:

```
#import Python library packages

        import numpy as np

        import matplotlib.pyplot as plt

        import pandas as pd

        from sklearn.metrics import classification_report

        from sklearn.metrics import confusion_matrix
```

```
from sklearn import preprocessing

from sklearn.preprocessing import LabelEncoder

from sklearn.model_selection import cross_val_score

import seaborn as sns


#reading the dataset from the local folder

data=pd.read_csv('covid19.csv')
```

In[2]:

```
#printing first 5 rows

data.head()
```

Output[2]:

	patient_id	global_num	sex	birth_year	age	country	province	city	latitude	longitude	infection_case	infection_order	state	Label
0	1000000001	2	male	1964	50s	Korea	Seoul	Gangseo-gu	37.460459	126.440680	overseas inflow	1.0	released	0
1	1000000002	5	male	1987	30s	Korea	Seoul	Jungnang-gu	37.478832	126.668558	overseas inflow	1.0	released	0
2	1000000003	6	male	1964	50s	Korea	Seoul	Jongno-gu	37.562143	126.801884	contact with patient	2.0	released	0
3	10000000	7	male	1991	20	Korea	Seoul	Mapo-gu	37.5674	127.0056	overseas	1.0	releas	0

	patient_id	global_num	sex	birth_year	age	country	province	city	latitude	longitude	infection_case	infection_order	state	Label
	04				s				54	27	inflow		ed	
4	1000000005	9	female	1992	20s	Korea	Seoul	Seongbuk-gu	37.460459	126.440680	contact with patient	2.0	released	0

In[3]

```
#As all the columns are categorical, check for unique values of each column

for i in data.columns:

    print(data[i].unique(),"\t",data[i].nunique())
```

Output[3]:

```
[1000000001 1000000002 1000000003 1000000004 1000000005 1000000006
 1000000007 1000000008 1000000009 1000000010 1000000011 1000000012
 1000000013 1000000014 1000000015 1000000016 1000000017 1000000018
 1000000019 1000000020 1000000021 1000000022 1000000023 1000000024
 1000000025 1000000026 1000000027 1000000028 1000000029 1000000030
 1000000031 1000000032 1000000033 1000000034 1000000035 1000000036
 1000000037 1000000038 1000000039 1000000040 1000000041 1000000042
 1000000043 1000000044 1000000045 1000000046 1000000047 1000000048
 1000000049 1000000050 1000000051 1000000052 1000000053 1000000054
 1000000055 1000000056 1000000057 1000000058 1000000059 1000000060
 1000000061 1000000062 1000000063 1000000064 1000000065 1000000066
 1000000067 1000000068 1000000069 1000000070 1000000071 1000000072
 1000000073 1000000074 1000000075 1000000076 1000000077 1000000078
 1000000079 1000000080 1000000081 1000000082 1000000083 1000000084
 1000000085 1000000086 1000000087 1000000088 1000000089 1000000090
 1000000091 1000000092 1000000093 1000000094 1000000095 1000000096
 1000000097 1000000098 1000000099 1000000100 1000000101 1000000102
 1000000103 1000000104 1000000105 1000000106 1000000107 1000000108
 1000000109 1000000110 1000000111 1000000112 1000000113 1000000114
 1000000115 1000000116 1000000117 1000000118 1000000119 1000000120
 1000000121 1000000122 1000000123 1000000124 1000000125 1000000126
```


In[4]

data.info()

Out[4]:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 160 entries, 0 to 159
Data columns (total 14 columns):
patient_id        160 non-null int64
global_num        160 non-null int64
sex               160 non-null object
birth_year        160 non-null int64
age               160 non-null object
country           160 non-null object
province          160 non-null object
city              160 non-null object
latitude          160 non-null float64
longitude         160 non-null float64
infection_case    160 non-null object
infection_order   18 non-null float64
state             160 non-null object
Label             160 non-null int64
dtypes: float64(3), int64(4), object(7)
memory usage: 17.6+ KB
```

In[5]

```
#Check how these unique categories are distributed among the columns

for i in data.columns:

   print(data[i].value_counts())

print()
```

Out[5]:

```
1000000160    1
1000000159    1
1000000058    1
1000000057    1
1000000056    1
1000000055    1
1000000054    1
1000000053    1
1000000052    1
1000000051    1
1000000050    1
```

```
1000000049    1
1000000048    1
1000000047    1
1000000046    1
1000000045    1
1000000044    1
1000000043    1
1000000042    1
1000000059    1
1000000060    1
1000000061    1
1000000071    1
1000000078    1
1000000077    1
1000000076    1
1000000075    1
1000000074    1
1000000073    1
1000000072    1
          ..
1000000090    1
1000000100    1
1000000089    1
1000000088    1
1000000087    1
1000000086    1
1000000085    1
1000000084    1
1000000083    1
1000000099    1
```

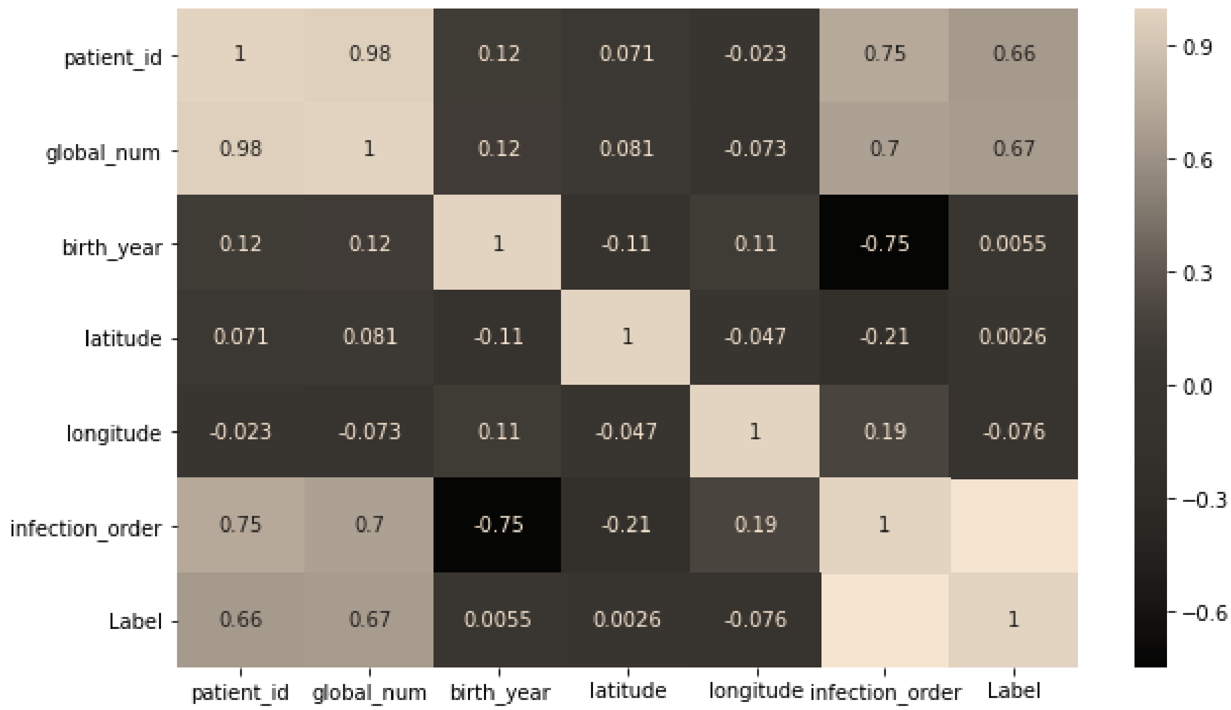
In[6]

#Heatmap of the columns on dataset with each other. It shows Pearson's correlation coefficient of column w.r.t other columns.

```
fig=plt.figure(figsize=(10,6))

sns.heatmap(data.corr(),annot=True)
```

Out[6]:

<matplotlib.axes._subplots.AxesSubplot at 0x1e3673ac748>




```
<matplotlib.axes._subplots.AxesSubplot at 0x1e3673ac748>
```

In[7]

```
#As scikit-learn algorithms do not generally work with string values, I've converted string categories to integers.
le=LabelEncoder()
for sex indata.columns:
    data[sex]=le.fit_transform(data[sex])
for age indata.columns:
    data[age]=le.fit_transform(data[age])
for country indata.columns:
    data[country]=le.fit_transform(data[country])
for province indata.columns:
    data[province]=le.fit_transform(data[province])
for city indata.columns:
    data[city]=le.fit_transform(data[city])
forinfection_caseindata.columns:
    data[infection_case]=le.fit_transform(data[infection_case])
for state indata.columns:
    data[state]=le.fit_transform(data[state])

#printing the first 5 rows
data.head()
```

Out[7]:

Out[11]:

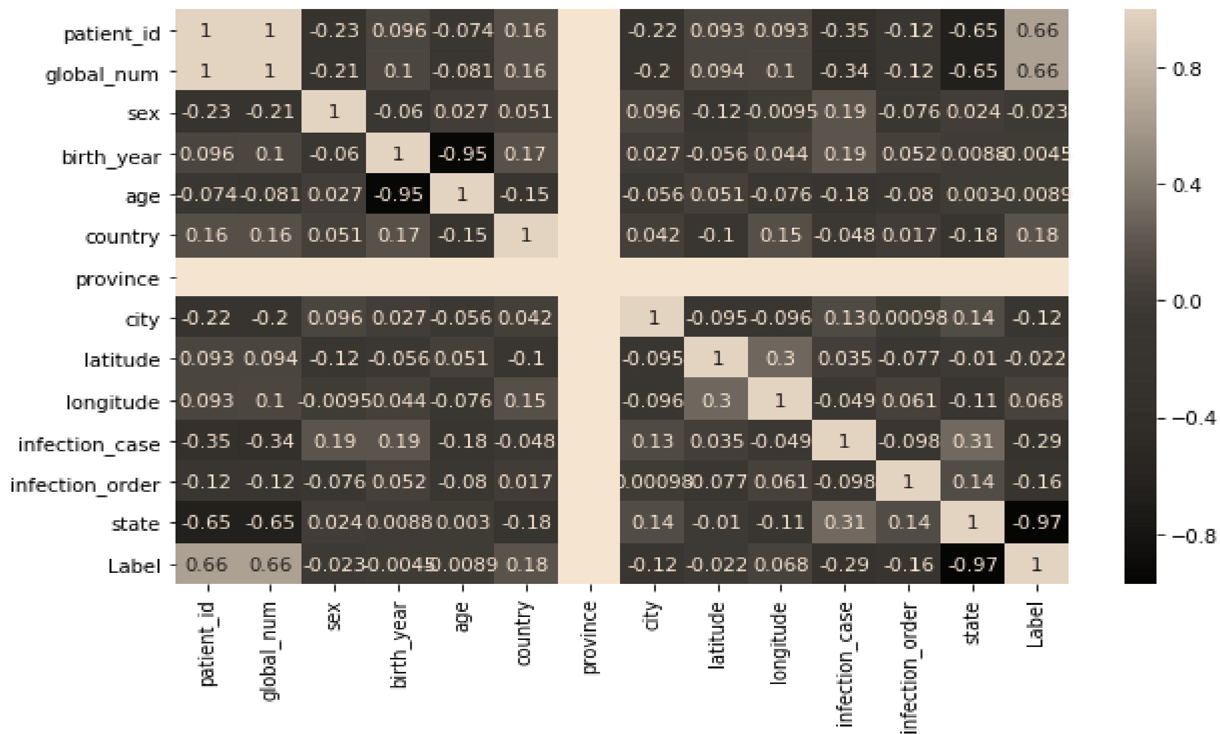
	patient_id	global_num	sex	birth_year	age	country	province	city	latitude	longitude	infection_case	infection_order	state	Label
0	0	0	1	22	4	1	0	7	37	2	7	0	2	0
1	1	1	1	45	2	1	0	14	39	5	7	0	2	0
2	2	2	1	22	4	1	0	12	64	24	5	1	2	0
3	3	3	1	49	1	1	0	15	72	54	7	0	2	0
4	4	4	0	50	1	1	0	19	37	2	5	1	2	0

In[8]

```
fig=plt.figure(figsize=(10,6))
sns.heatmap(data.corr(),annot=True)
```

Out[8]:

<matplotlib.axes._subplots.AxesSubplot at 0x1e3674c6d30>



In[9]

```
#X is thedataframe containing input data / features

#y is the series which has results which are to be predicted.
X=data[data.columns[:-1]]
y=data['Label']
X.head(2)
```

Out[9]:

	patient_id	global_num	sex	birth_year	age	country	province	city	latitude	longitude	infection_case	infection_order	state
0	0	0	1	22	4	1	0	7	37	2	7	0	2
1	1	1	1	45	2	1	0	14	39	5	7	0	2

In[10]

```
from sklearn.ensemble import RandomForestClassifier

#Create a Gaussian Classifier

clf=RandomForestClassifier(n_estimators=100)


#Train the model using the training sets y_pred=clf.predict(X_test)

clf.fit(X_train,y_train)

y_pred=clf.predict(X_test)

#Import scikit-learn metrics module for accuracy calculation

from sklearn import metrics

# Model Accuracy, how often is the classifier correct?

print("Accuracy for random forest:",metrics.accuracy_score(y_test, y_pred))

print(confusion_matrix(y_test, y_pred))

print(classification_report(y_test,  y_pred))
```


Output[10]:

```
Accuracy for random forest: 0.9056603773584906
[[15  4  0]
 [ 0 33  0]
 [ 0  1  0]]
              precision    recall  f1-score   support

           0       1.00      0.79      0.88        19
           1       0.87      1.00      0.93        33
           2       0.00      0.00      0.00         1

    accuracy                           0.91        53
   macro avg       0.62      0.60      0.60        53
weighted avg       0.90      0.91      0.90        53
```

Result:

Thus, the python program for random forest tree was executed successfully.

Ex.No:7	Build SVM models
Date:	

Aim:

To write a python program to build Support vector machine models.

Procedure:

1. import Python library packages
2. reading the dataset from the local folder
3. printing first 5 rows
4. As all the columns are categorical, check for unique values of each column
5. Check how these unique categories are distributed among the columns
6. Heatmap of the columns on dataset with each other. It shows Pearson's correlation coefficient of column w.r.t other columns.
7. As scikit-learn algorithms do not generally work with string values, I've converted string categories to integers.
8. printing the first 5 rows
9. X is the dataframe containing input data / features
10. y is the series which has results which are to be predicted.
11. Import train_test_split function
12. Split dataset into training set and test set
13. Create a svm Classifier
14. Train the model using the training sets
15. Predict the response for test dataset
16. normalizer
17. Import scikit-learn metrics module for accuracy calculation
18. Model Accuracy: how often is the classifier correct?
19. summarize scores
20. calculate roc curves
21. plot the roc curve for the model
22. axis labels
23. show the legend
24. show the plot

Program:

```
#import Python library packages
    import numpy as np
    import matplotlib.pyplot as plt
    import pandas as pd
    from sklearn.metrics import classification_report
    from sklearn.metrics import confusion_matrix
    from sklearn import preprocessing
    from sklearn.preprocessing import LabelEncoder
    from sklearn.model_selection import cross_val_score
    import seaborn as sns

    #reading the dataset from the local folder
    data=pd.read_csv('covid19.csv')
```

In[2]:

```
#printing first 5 rows
data.head()
```

Output[2]:

	patient_id	global_num	sex	birth_year	age	country	province	city	latitude	longitude	infection_case	infection_order	state	Label
0	1000000001	2	male	1964	50s	Korea	Seoul	Gangseo-gu	37.460459	126.440680	overseas inflow	1.0	released	0

	patient_id	global_num	sex	birth_year	age	country	province	city	latitude	longitude	infection_case	infection_order	state	Label
1	1000000002	5	male	1987	30s	Korea	Seoul	Jungnang-gu	37.478832	126.668558	overseas inflow	1.0	released	0
2	1000000003	6	male	1964	50s	Korea	Seoul	Jongno-gu	37.562143	126.801884	contact with patient	2.0	released	0
3	1000000004	7	male	1991	20s	Korea	Seoul	Mapo-gu	37.567454	127.005627	overseas inflow	1.0	released	0
4	1000000005	9	female	1992	20s	Korea	Seoul	Seongbuk-gu	37.460459	126.440680	contact with patient	2.0	released	0

In[3]

```
#As all the columns are categorical, check for unique values of each column

for i in data.columns:

    print(data[i].unique(),"\t",data[i].nunique())
```

Output[3]:

```
[1000000001 1000000002 1000000003 1000000004 1000000005 1000000006
 1000000007 1000000008 1000000009 1000000010 1000000011 1000000012
 1000000013 1000000014 1000000015 1000000016 1000000017 1000000018
 1000000019 1000000020 1000000021 1000000022 1000000023 1000000024
 1000000025 1000000026 1000000027 1000000028 1000000029 1000000030
 1000000031 1000000032 1000000033 1000000034 1000000035 1000000036
 1000000037 1000000038 1000000039 1000000040 1000000041 1000000042
```

```
1000000043 1000000044 1000000045 1000000046 1000000047 1000000048
1000000049 1000000050 1000000051 1000000052 1000000053 1000000054
1000000055 1000000056 1000000057 1000000058 1000000059 1000000060
1000000061 1000000062 1000000063 1000000064 1000000065 1000000066
1000000067 1000000068 1000000069 1000000070 1000000071 1000000072
1000000073 1000000074 1000000075 1000000076 1000000077 1000000078
1000000079 1000000080 1000000081 1000000082 1000000083 1000000084
1000000085 1000000086 1000000087 1000000088 1000000089 1000000090
1000000091 1000000092 1000000093 1000000094 1000000095 1000000096
1000000097 1000000098 1000000099 1000000100 1000000101 1000000102
1000000103 1000000104 1000000105 1000000106 1000000107 1000000108
1000000109 1000000110 1000000111 1000000112 1000000113 1000000114
1000000115 1000000116 1000000117 1000000118 1000000119 1000000120
1000000121 1000000122 1000000123 1000000124 1000000125 1000000126
```

In[4]

data.info()

Out[4]:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 160 entries, 0 to 159
Data columns (total 14 columns):
patient_id        160 non-null int64
global_num        160 non-null int64
sex               160 non-null object
birth_year        160 non-null int64
age               160 non-null object
country           160 non-null object
province          160 non-null object
city              160 non-null object
latitude          160 non-null float64
longitude         160 non-null float64
infection_case    160 non-null object
infection_order   18 non-null float64
state             160 non-null object
Label             160 non-null int64
dtypes: float64(3), int64(4), object(7)
memory usage: 17.6+ KB
```

In[5]

```
#Check how these unique categories are distributed among the columns
for i in data.columns:
    print(data[i].value_counts())
```

print()

Out[5]:

```
1000000160    1
1000000159    1
1000000058    1
1000000057    1
1000000056    1
1000000055    1
1000000054    1
1000000053    1
1000000052    1
1000000051    1
1000000050    1
1000000049    1
1000000048    1
1000000047    1
1000000046    1
1000000045    1
1000000044    1
1000000043    1
1000000042    1
1000000059    1
1000000060    1
1000000061    1
1000000071    1
1000000078    1
1000000077    1
1000000076    1
1000000075    1
1000000074    1
1000000073    1
1000000072    1
          ..
1000000090    1
1000000100    1
1000000089    1
1000000088    1
1000000087    1
1000000086    1
1000000085    1
1000000084    1
1000000083    1
1000000099    1
```

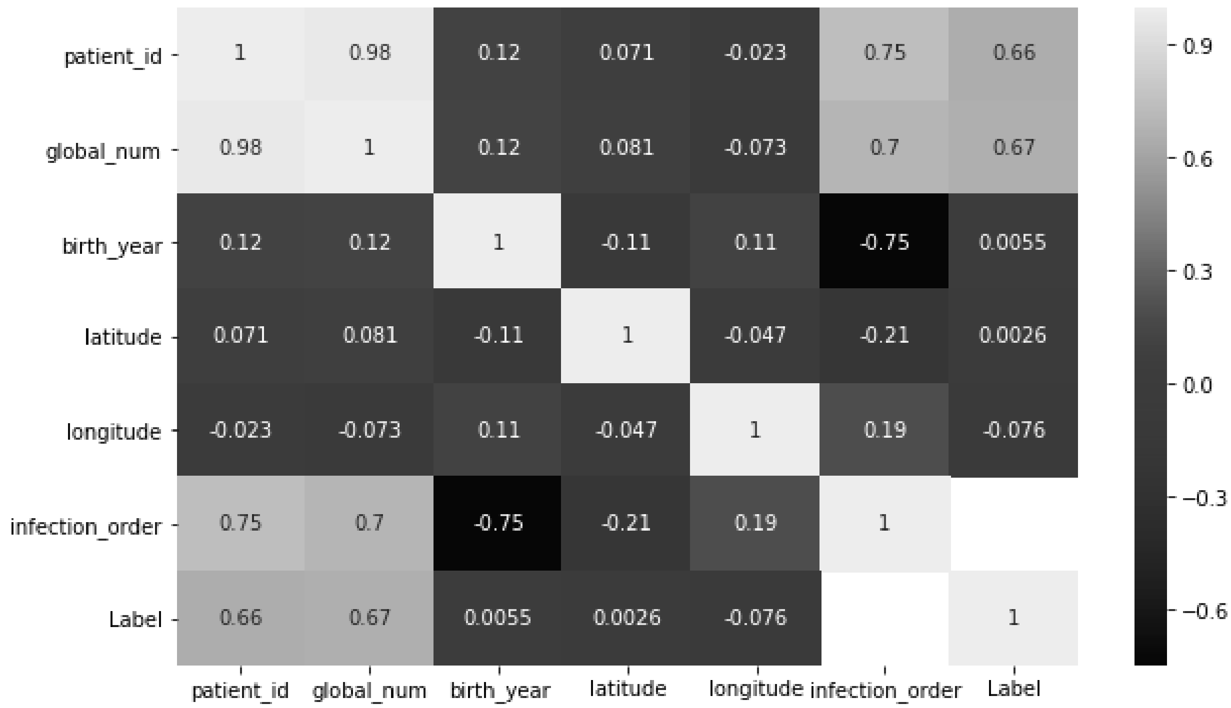
In[6]

#Heatmap of the columns on dataset with each other. It shows Pearson's correlation coefficient of column w.r.t other columns.

```
fig=plt.figure(figsize=(10,6))

sns.heatmap(data.corr(),annot=True)
```

Out[6]:

<matplotlib.axes._subplots.AxesSubplot at 0x1e3673ac748>



<matplotlib.axes._subplots.AxesSubplot at 0x1e3673ac748>

In[7]

```
#As scikit-learn algorithms do not generally work with string values, I've converted string categories to integers.
le=LabelEncoder()
for sex indata.columns:
   data[sex]=le.fit_transform(data[sex])
for age indata.columns:
   data[age]=le.fit_transform(data[age])
for country indata.columns:
   data[country]=le.fit_transform(data[country])
for province indata.columns:
   data[province]=le.fit_transform(data[province])
for city indata.columns:
```

```
    data[city]=le.fit_transform(data[city])
forinfection_caseindata.columns:
    data[infection_case]=le.fit_transform(data[infection_case])
for state indata.columns:
    data[state]=le.fit_transform(data[state])

#printing the first 5 rows
data.head()
```

Out[7]:

Out[11]:

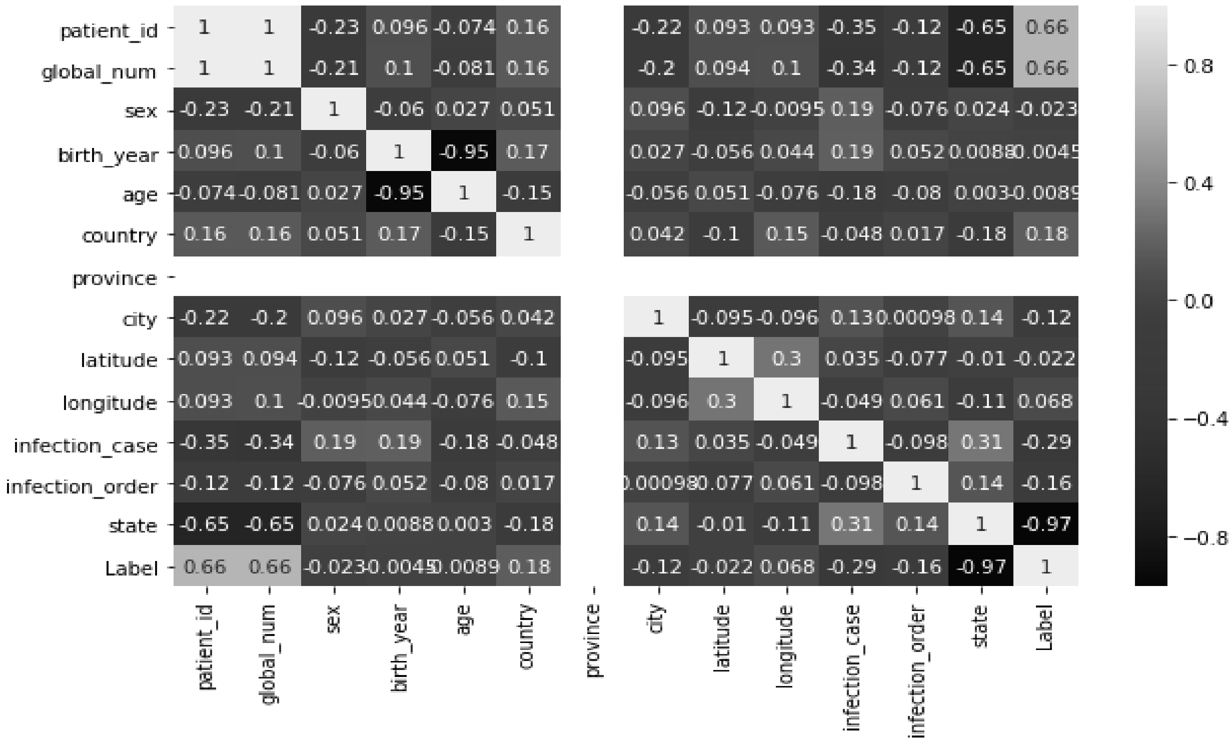
	patient_id	global_num	sex	birth_year	age	country	province	city	latitude	longitude	infection_case	infection_order	state	Label
0	0	0	1	22	4	1	0	7	37	2	7	0	2	0
1	1	1	1	45	2	1	0	14	39	5	7	0	2	0
2	2	2	1	22	4	1	0	12	64	24	5	1	2	0
3	3	3	1	49	1	1	0	15	72	54	7	0	2	0
4	4	4	0	50	1	1	0	19	37	2	5	1	2	0

In[8]

```
fig=plt.figure(figsize=(10,6))
sns.heatmap(data.corr(),annot=True)
```

Out[8]:

<matplotlib.axes._subplots.AxesSubplot at 0x1e3674c6d30>



In[9]

***#X is the**dataframe containing input data / features*

#y is the series which has results which are to be predicted.

```
X=data[data.columns[:-1]]
y=data['Label']
X.head(2)
```

Out[9]:

	patient_id	global_num	sex	birth_year	age	country	province	city	latitude	longitude	infection_case	infection_order	state
0	0	0	1	22	4	1	0	7	37	2	7	0	2
1	1	1	1	45	2	1	0	14	39	5	7	0	2

In[10]

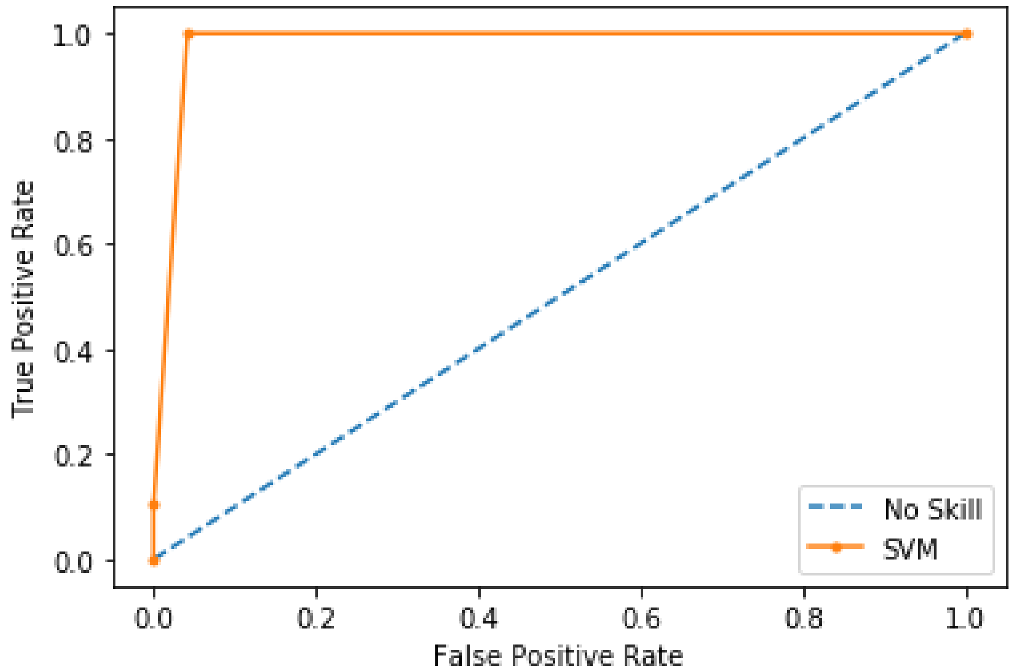
```
# Import train_test_split function

from sklearn.model_selection import train_test_split

# Split dataset into training set and test set
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33) # 70% training and 30%
test

from sklearn import svm

#Create a svm Classifier

clf = svm.SVC(kernel='linear') # Linear Kernel

#Train the model using the training sets

clf.fit(X_train, y_train)

#Predict the response for test dataset

y_pred = clf.predict(X_test)

#normalizer

ns_probs = [0 for _ in range(len(y_test))]

#Import scikit-learn metrics module for accuracy calculation

from sklearn import metrics

# Model Accuracy: how often is the classifier correct?

print("Accuracy for Runlengthsvm:",metrics.accuracy_score(y_test, y_pred))

print(confusion_matrix(y_test, y_pred))

print(classification_report(y_test, y_pred))

ns_auc = roc_auc_score(y_test, ns_probs)

lr_auc = roc_auc_score(y_test, y_pred)

print(roc_auc_score(y_test, y_pred))

# summarize scores

print('No Skill: ROC AUC=%.3f' % (ns_auc))

print('SVM: ROC AUC=%.3f' % (lr_auc))

# calculate roc curves

ns_fpr, ns_tpr, _ = roc_curve(y_test, ns_probs)

lr_fpr, lr_tpr, _ = roc_curve(y_test, y_pred)

# plot the roc curve for the model
```

```
pyplot.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')

pyplot.plot(lr_fpr, lr_tpr, marker='.', label='SVM')

# axis labels

pyplot.xlabel('False Positive Rate')

pyplot.ylabel('True Positive Rate')

# show the legend

pyplot.legend()

# show the plot

pyplot.show()
```

Output[10]:

```
Accuracy for Runlengthsvm: 0.9245283018867925
[[23  1  0]
 [ 0 26  3]
 [ 0  0  0]]
              precision    recall  f1-score   support

           0       1.00      0.96      0.98        24
           1       0.96      0.90      0.93        29
           2       0.00      0.00      0.00         0

    accuracy                           0.92        53
   macro avg       0.65      0.62      0.64        53
weighted avg       0.98      0.92      0.95        53

0.9813218390804598
No Skill: ROC AUC=0.500
SVM: ROC AUC=0.981
C:\Users\acer\Anaconda3\lib\site-
packages\sklearn\metrics\classification.py:1439: UndefinedMetricWarning:
Recall and F-score are ill-defined and being set to 0.0 in labels with no
true samples.
  'recall', 'true', average, warn_for)
```



Result:

Thus, the python program for buildsvm models was executed successfully.

Ex.No:8	**Implement Ensembling Techniques (K- Means)**
Date:	

Aim:

To write a python program to implement ensembling techniques using k-means

Procedure:

Step 1 - Import basic libraries.
Step 2 - Importing the dataset.
Step 3 - Data preprocessing.
Step 4 - Training the model.
Step 5 - Testing and evaluation of the model.
Step 6 - Visualizing the model.

Program:

```
fromsklearn.clusterimportKMeans
fromsklearnimportpreprocessing
fromsklearn.mixtureimportGaussianMixture
fromsklearn.datasetsimportload_iris
importsklearn.metricsassm
import pandas as pd
importnumpyas np
importmatplotlib.pyplotasplt
```

In [2]:

```
dataset=load_iris()
# print(dataset)
```

In [3]:

```
X=pd.DataFrame(dataset.data)
X.columns=['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
y=pd.DataFrame(dataset.target)
y.columns=['Targets']
# print(X)
```

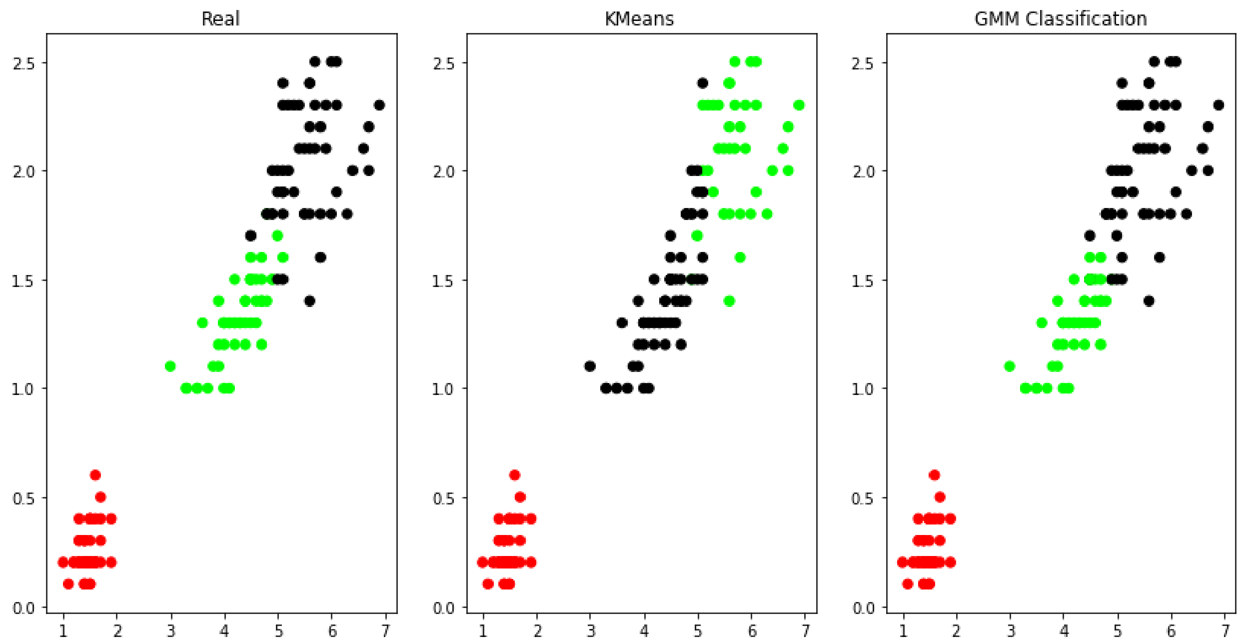
In [4]:

```
plt.figure(figsize=(14,7))
colormap=np.array(['red','lime','black'])

# REAL PLOT
plt.subplot(1,3,1)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y.Targets],s=40)
plt.title('Real')

# K-PLOT
plt.subplot(1,3,2)
```

```
model=KMeans(n_clusters=3)
model.fit(X)
predY=np.choose(model.labels_,[0,1,2]).astype(np.int64)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[predY],s=40)
plt.title('KMeans')

# GMM PLOT
scaler=preprocessing.StandardScaler()
scaler.fit(X)
xsa=scaler.transform(X)
xs=pd.DataFrame(xsa,columns=X.columns)
gmm=GaussianMixture(n_components=3)
gmm.fit(xs)
y_cluster_gmm=gmm.predict(xs)
plt.subplot(1,3,3)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y_cluster_gmm],s=40)
plt.title('GMM Classification')
```

Out[4]:

```
Text(0.5, 1.0, 'GMM Classification')
```



Result:

Thus, the python program for the ensemble techniques using k means plotting was executed successfully.

Ex.No:9	**Implement Clustering Algorithms**
Date:	

Aim:

To write a python program to implement clustering algorithm using k-means method.

Procedure:

Step 1 - Import basic libraries.
Step 2 - Importing the dataset.
Step 3 - Data preprocessing.
Step 4 - Training the model using k-Means.
Step 5 - Testing and evaluation of the model.
Step 6 - Visualizing the model.

Program:

```
fromsklearn.clusterimportKMeans
importpandasaspd
fromsklearn.preprocessingimportMinMaxScaler
frommatplotlibimportpyplotasplt
%matplotlibinline
```

In [2]:

```
df=pd.read_csv("income.csv")

df.head()
```

Out[2]:

	Name	Age	Income($)
0	Rob	27	70000
1	Michael	29	90000
2	Mohan	29	61000

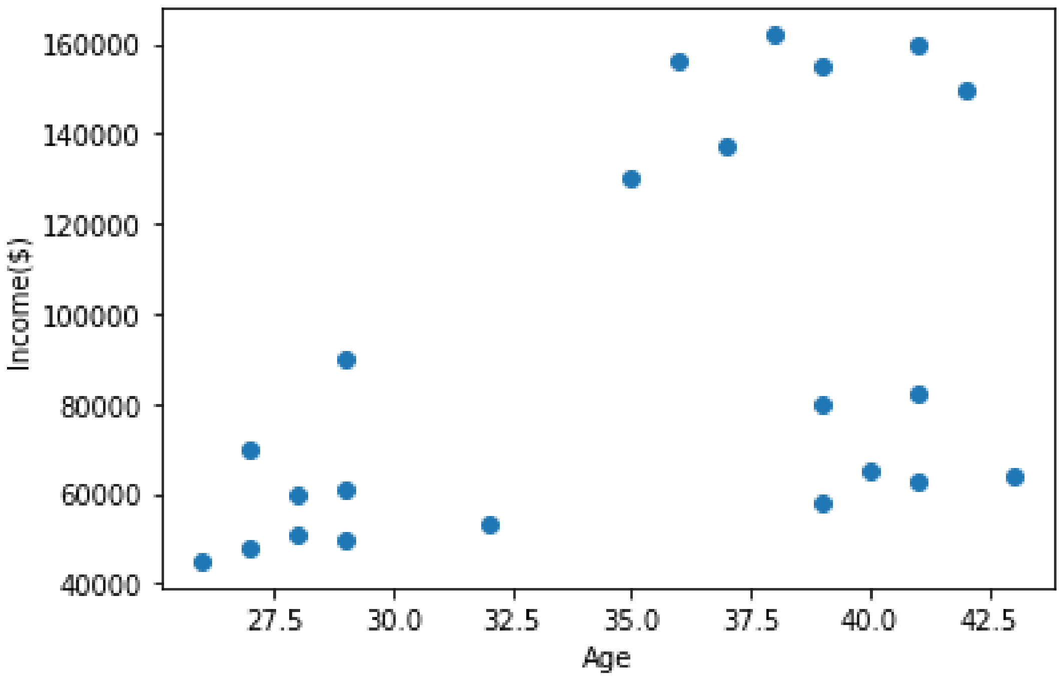
	Name	Age	Income($)
3	Ismail	28	60000
4	Kory	42	150000

In [3]:

```
plt.scatter(df.Age,df['Income($)'])
plt.xlabel('Age')
plt.ylabel('Income($)')
```

Out[3]:

```
Text(0, 0.5, 'Income($)')
```



In [4]:

```
km=KMeans(n_clusters=3)
y_predicted=km.fit_predict(df[['Age','Income($)']])
y_predicted
```

Out[4]:

```
array([0, 0, 2, 2, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 2])
```

In [5]:

```
df['cluster']=y_predicted
```

df.head()

Out[5]:

	Name	Age	Income($)	cluster
0	Rob	27	70000	0
1	Michael	29	90000	0
2	Mohan	29	61000	2
3	Ismail	28	60000	2
4	Kory	42	150000	1

In [6]:

```
km.cluster_centers_
```

Out[6]:

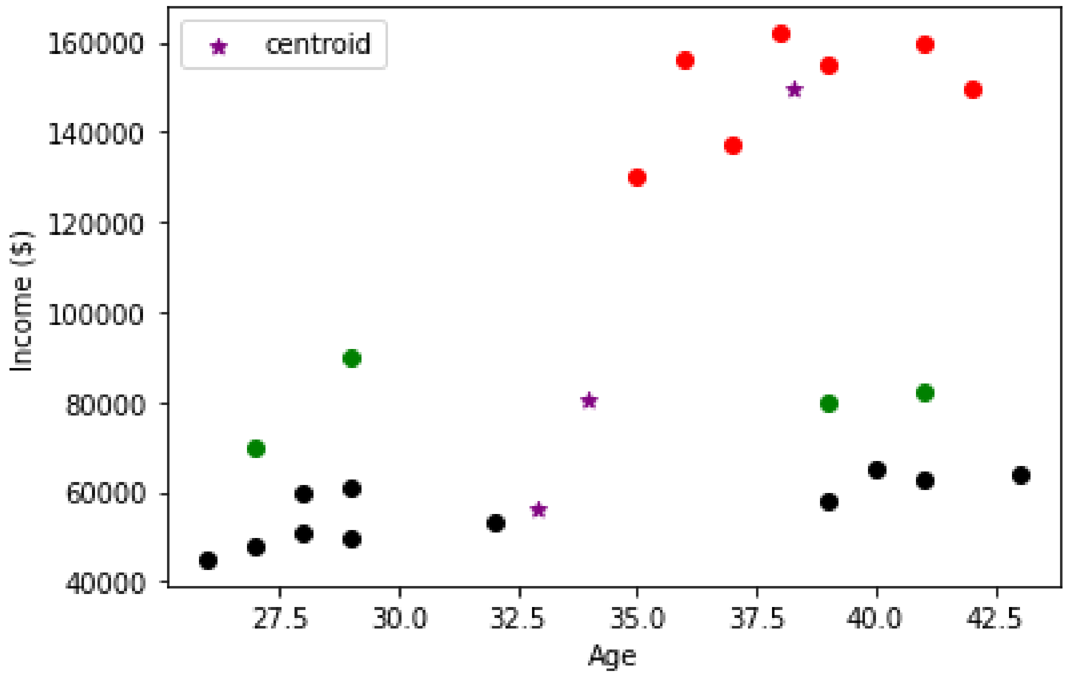
```
array([[3.40000000e+01, 8.05000000e+04],
       [3.82857143e+01, 1.50000000e+05],
       [3.29090909e+01, 5.61363636e+04]])
```

In [7]:

```
df1=df[df.cluster==0]
df2=df[df.cluster==1]
df3=df[df.cluster==2]
plt.scatter(df1.Age,df1['Income($)'],color='green')
plt.scatter(df2.Age,df2['Income($)'],color='red')
plt.scatter(df3.Age,df3['Income($)'],color='black')
plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],color='purple',marker='*',label='centroid')
plt.xlabel('Age')
plt.ylabel('Income ($)')
plt.legend()
```

Out[7]:

```
<matplotlib.legend.Legend at 0x13943cc89d0>
```



Preprocessing using min max scaler

In [8]:

```
scaler=MinMaxScaler()

scaler.fit(df[['Income($)']])
df['Income($)'] =scaler.transform(df[['Income($)']])

scaler.fit(df[['Age']])
df['Age'] =scaler.transform(df[['Age']])
```

In [9]:

```
df.head()
```

Out[9]:

	Name	Age	Income($)	cluster
0	Rob	0.058824	0.213675	0
1	Michael	0.176471	0.384615	0
2	Mohan	0.176471	0.136752	2

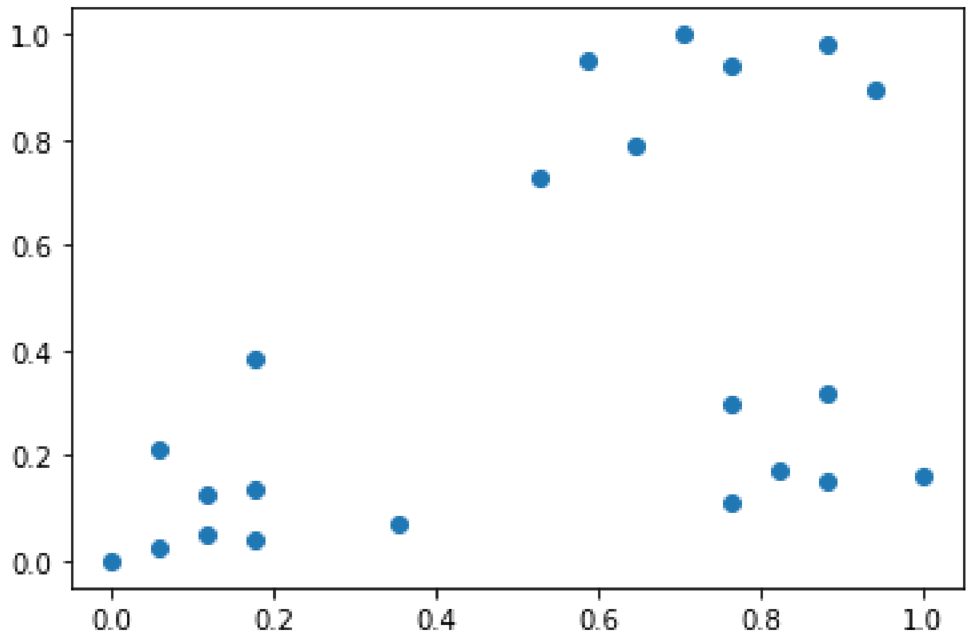
	Name	Age	Income($)	cluster
3	Ismail	0.117647	0.128205	2
4	Kory	0.941176	0.897436	1

In [10]:

```
plt.scatter(df.Age,df['Income($)'])
```

Out[10]:

```
<matplotlib.collections.PathCollection at 0x13943d5be20>
```



In [11]:

```
km=KMeans(n_clusters=3)
y_predicted=km.fit_predict(df[['Age','Income($)']])
y_predicted
```

Out[11]:

```
array([1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2])
```

In [12]:

```
df['cluster']=y_predicted
df.head()
```

Out[12]:

	Name	Age	Income($)	cluster
0	Rob	0.058824	0.213675	1
1	Michael	0.176471	0.384615	1
2	Mohan	0.176471	0.136752	1
3	Ismail	0.117647	0.128205	1
4	Kory	0.941176	0.897436	0

In [13]:

```
km.cluster_centers_
```

Out[13]:

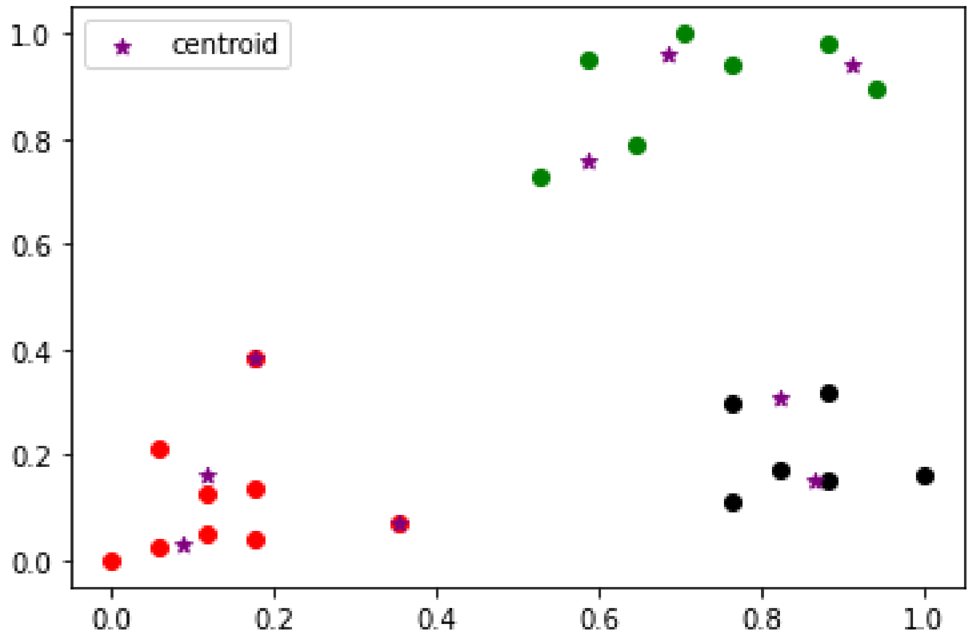
```
array([[0.72268908, 0.8974359 ],
       [0.1372549 , 0.11633428],
       [0.85294118, 0.2022792 ]])
```

In [16]:

```
df1=df[df.cluster==0]
df2=df[df.cluster==1]
df3=df[df.cluster==2]
plt.scatter(df1.Age,df1['Income($)'],color='green')
plt.scatter(df2.Age,df2['Income($)'],color='red')
plt.scatter(df3.Age,df3['Income($)'],color='black')
plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],color='purple',marker='*',label='centroid')
plt.legend()
```

Out[16]:

```
<matplotlib.legend.Legend at 0x13943d23190>
```



Elbow Plot

In [17]:

```
sse= []
k_rng=range(1,10)
forkink_rng:
km=KMeans(n_clusters=k)
km.fit(df[['Age','Income($)']])
sse.append(km.inertia_)
```

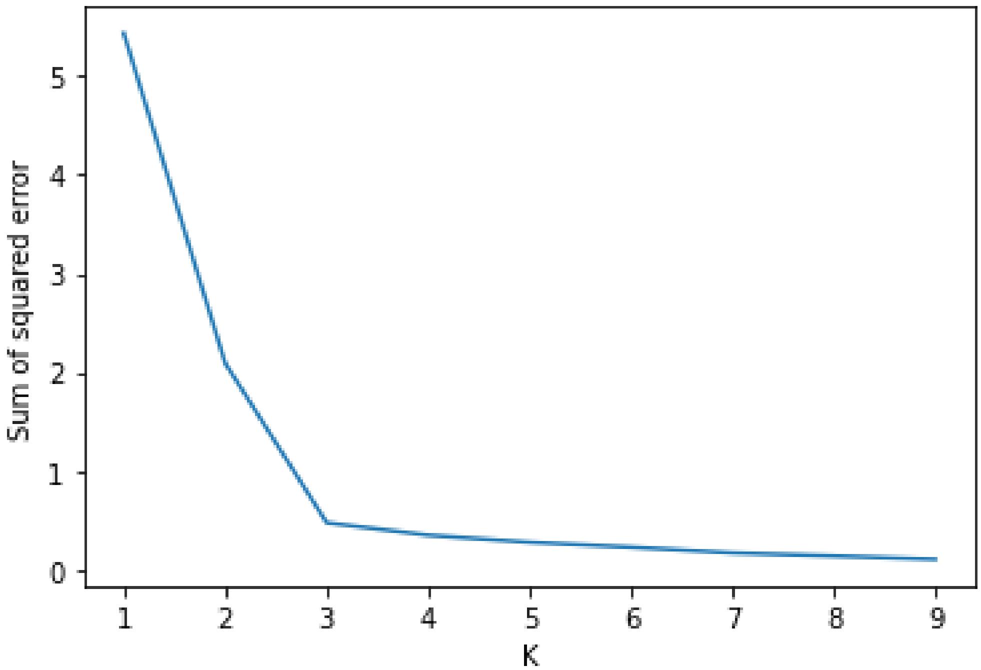
```
C:\Users\janaj\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:881: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=1.
warnings.warn(
```

In [18]:

```
plt.xlabel('K')
plt.ylabel('Sum of squared error')
plt.plot(k_rng,sse)
```

Out[18]:

```
[<matplotlib.lines.Line2D at 0x13945f3d940>]
```

Result:

Thus, the python program for clustering algorithm using k-means was executed successfully.

Ex.No:10	**Implement EM for Bayesian Networks**
Date:	

Aim:

To write a python program to implement EM for Bayesian Networks

Procedure:

Step 1 - Import basic libraries.
Step 2 - Importing the dataset.
Step 3 - Data preprocessing.
Step 4 - Training the model using k-Means.
Step 5 - Testing and evaluation of the model.
Step 6 - Visualizing the model.

Program:

In [1]:
```
importmath
importnumpyasnp
importmatplotlib.pyplotasplt
```

Helper Functions

In [2]:

```
defnCr(n,r):
"""
    A naive implementation for calculating the combination number C_n^r.

Args:
        n: int, the total number
        r: int, the number of selected

    Returns:
C_n^r, the combination number
    """
f=math.factorial
returnf(n)/f(r)/f(n-r)


defbinomial(x,n,p):
"""
    The bionomial distribution C_n^xp^x (1-p)^(n-x)

Args:
        n: int, the total number
```



```
        x: int, the number of selected
        p: float, the probability

    Returns:
        The bionomial probability C_n^xp^x (1-p)^(n-x)
    """
returnnCr(n,x)*(p**x)*(1-p)**(n-x)


deflog_likelihood(X,n,theta):
"""
    Calculates the log likelihood function for the two coins problem.

Args:
        X: np.array of shape (n_trials,), dtype int,
            the observations (number of heads at each trial).
        n: int, total number of tosses per trial.
        theta: tuple of (lambda, pA, pB), where
            - lambda: float, the prior probability of selecting coin A (=1/2)
            - pA: float, coin A's probability of showing head
            - pB: float, coin B's probability of showing head

    Returns:
        log-likelihood
            f(theta) = sum_i log sum_ziP(xi, zi; theta)
                     = sum_ilog[ lam ( nCr(10, xi) pA^xi (1-pA)^(10-xi) )
                                + (1-lam) ( nCr(10, xi) pB^xi (1-pB)^(10-xi)
) ]
    """
(lam,p1,p2)=theta
ll=0
forxinX:
ll+=np.log(
lam*binomial(x,n,p1)+(1-lam)*binomial(x,n,p2)
)
returnll


defELBO(X,n,Q,theta):
"""
    Calculates the ELBO for the two coins problem.

Args:
        X: np.array of shape (n_trials,), dtype int,
            the observations (number of heads at each trial).
        n: int, total number of tosses per trial.
        Q: np.array of shape (n_trials, 2), dtype float,
            the hidden posterior q(z) (z = A, B) computed in the E-step.
        theta: tuple of (lambda, pA, pB), where
            - lambda: float, the prior probability of selecting coin A (=1/2)
            - pA: float, coin A's probability of showing head
            - pB: float, coin B's probability of showing head

    Returns:
        ELBO (Evidence Lower Bound)
```

```
            g(theta) = sum_isum_zi Q(zi) log( P(xi, zi; theta) / Q(zi) )
    """
(lam,p1,p2)=theta
elbo=0
fori,xinenumerate(X):
elbo+=Q[i,0]*np.log(lam*binomial(x,n,p1)/Q[i,0])
elbo+=Q[i,1]*np.log((1-lam)*binomial(x,n,p2)/Q[i,1])
returnelbo


defplot_coin_function(grid_fn,title,path=[]):
"""
    Plots a function wrtpA, pB using 2D contours.
    Reference: https://nbviewer.jupyter.org/github/eecs445-f16/umich-eecs445-
f16/blob/master/handsOn_lecture17_clustering-mixtures-
em/handsOn_lecture17_clustering-mixtures-em.ipynb#Problem:-implement-EM-for-
Coin-Flips

Args:
grid_fn: callable, a function that takes pA, pB as inputs
            and returns the function value at that point.
        title: string, title of the plot.
        path: (optional) A list of tuple of (pA, pB) that are visited in the
EM iterations.
            Visualized as line segments if not empty.

    Returns:
        Shows the figure and returns None.
    """
xvals=np.linspace(0.01,0.99,100)
yvals=np.linspace(0.01,0.99,100)
xx,yy=np.meshgrid(xvals,yvals)

grid=np.zeros([len(xvals),len(yvals)])
foriinrange(len(xvals)):
forjinrange(len(yvals)):
grid[j,i]=grid_fn(xvals[i],yvals[j])

plt.figure(figsize=(6,4.5),dpi=100)
C=plt.contour(xx,yy,grid,1000)
cbar=plt.colorbar(C)
plt.title(title,fontsize=15)
plt.xlabel(r"$p_A$",fontsize=12)
plt.ylabel(r"$p_B$",fontsize=12)

ifpath:
p1,p2=zip(*path)
plt.plot(p1,p2,'g+-')
plt.text(p1[0]-0.15,p2[0]-
0.05,'start:\n$p_A={}$\n$p_B={}$'.format(p1[0],p2[0]),color='green',size=10)
plt.text(p1[-1]+0.0,p2[-
1]+0.02,'end:\n$p_A={:.3f}$\n$p_B={:.3f}$'.format(p1[-1],p2[-
1]),color='green',size=10)

plt.show()
```

```
defplot_coin_likelihood(X,n,path=[]):
"""
    Plots the coin likelihood wrtpA, pB using 2D contours.

Args:
        X: np.array of shape (n_trials,), dtype int,
            the observations (number of heads at each trial).
        n: int, total number of tosses per trial.
        path: (optional) A list of tuple of (pA, pB) that are visited in the
EM iterations.
            Visualized as line segments if not empty.

    Returns:
        Shows the figure and returns None.
    """
grid_fn=lambdapA,pB:log_likelihood(X,n,(lam,pA,pB))
returnplot_coin_function(
grid_fn=grid_fn,
title=r"Log-Likelihood $\log p(\mathcal{X}| \{p_A, p_B\})$",
path=path
)


defplot_coin_ELBO(X,n,Q,path=None):
"""
    Plots the coin ELBO wrtpA, pB using 2D contours.
    Reference: https://nbviewer.jupyter.org/github/eecs445-f16/umich-eecs445-
f16/blob/master/handsOn_lecture17_clustering-mixtures-
em/handsOn_lecture17_clustering-mixtures-em.ipynb#Problem:-implement-EM-for-
Coin-Flips

Args:
        X: np.array of shape (n_trials,), dtype int,
            the observations (number of heads at each trial).
        n: int, total number of tosses per trial.
        Q: np.array of shape (n_trials, 2), dtype float,
            the hidden posterior q(z) (z = A, B) computed in the E-step.
        path: (optional) A list of tuple of (pA, pB) that are visited in the
EM iterations.
            Visualized as line segments if not empty.

    Returns:
        Shows the figure and returns None.
    """
grid_fn=lambdapA,pB:ELBO(X,n,Q,(lam,pA,pB))
returnplot_coin_function(grid_fn=grid_fn,title="ELBO",path=path)
```

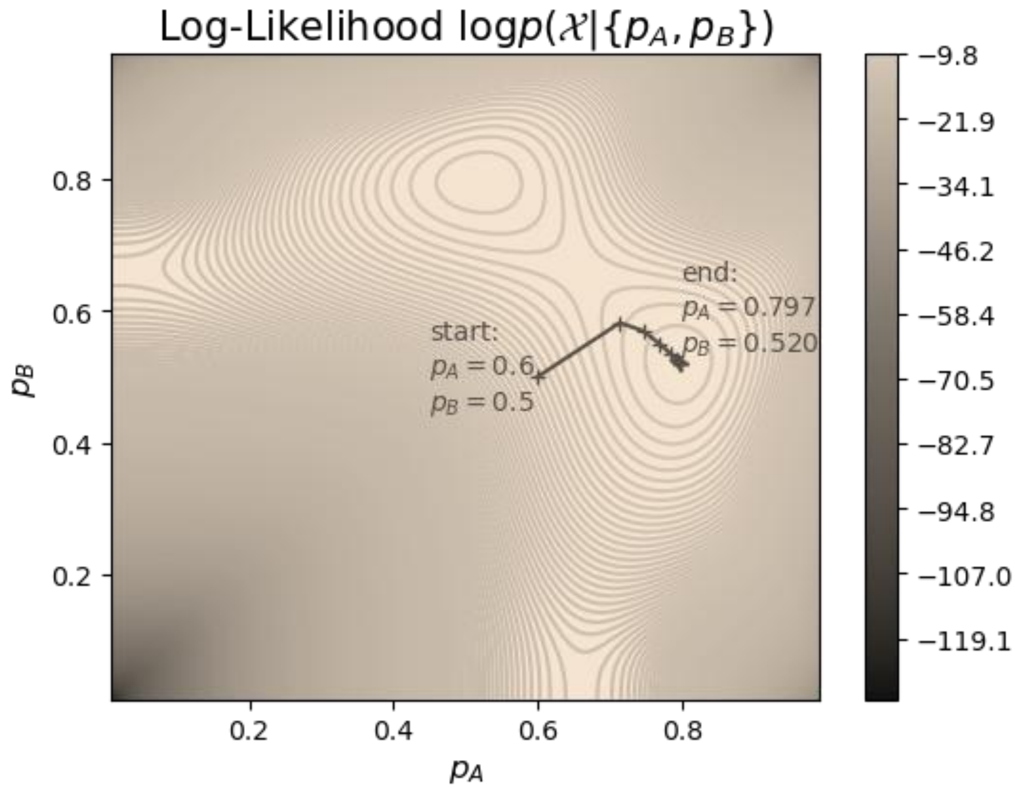
In [3]:

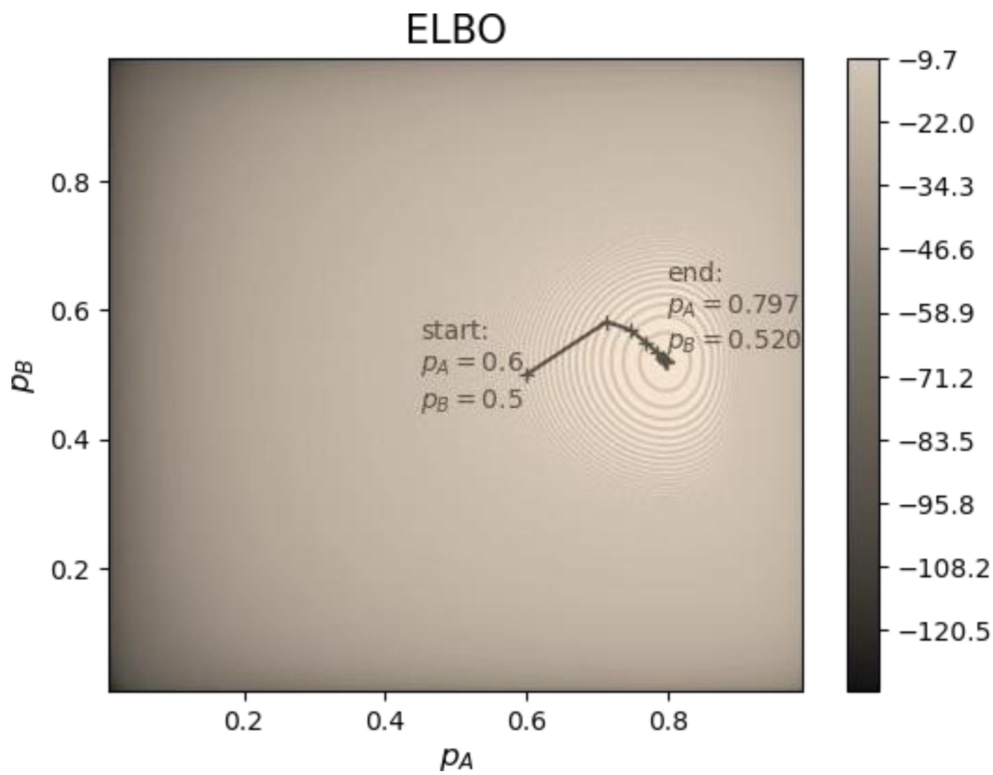
```
"""Starts the EM algorithm."""

n=10# number of tosses per trial
X=[5,9,8,4,7]# observation
lam=0.5# prior
```

```
p1=0.6# parameter: pA
p2=0.5# parameter: pB
n_trials=len(X)# number of trials
n_iters=10# number of EM iterations

path=[(p1,p2)]
print('Init: theta = ')
print(p1,p2)

foriinrange(n_iters):
print(f'==========EM Iter: {i+1}==========')

# E-step
q=np.zeros([n_trials,2])
fortrialinrange(n_trials):
x=X[trial]
q[trial,0]=lam*binomial(x,n,p1)
q[trial,1]=(1-lam)*binomial(x,n,p2)
q[trial,:]/=np.sum(q[trial,:])

print('E-step: q(z) = ')
print(q)

# M-step
p1=sum((np.array(X)/n)*q[:,0])/sum(q[:,0])
p2=sum((np.array(X)/n)*q[:,1])/sum(q[:,1])

path.append([p1,p2])

print('M-step: theta = ')
print(p1,p2)

plot_coin_likelihood(X,n,path)
plot_coin_ELBO(X,n,q,path)
Init: theta =
0.6 0.5
==========EM Iter: 1==========
E-step: q(z) =
[[0.44914893 0.55085107]
 [0.80498552 0.19501448]
 [0.73346716 0.26653284]
 [0.35215613 0.64784387]
 [0.64721512 0.35278488]]
M-step: theta =
0.713012235400516 0.5813393083136625
==========EM Iter: 2==========
E-step: q(z) =
[[0.29581932 0.70418068]
 [0.81151045 0.18848955]
 [0.70642201 0.29357799]
 [0.19014454 0.80985546]
 [0.57353393 0.42646607]]
M-step: theta =
0.7452920360819947 0.5692557501718727
==========EM Iter: 3==========
```

```
E-step: q(z) =
[[0.21759232 0.78240768]
 [0.86984852 0.13015148]
 [0.75115408 0.24884592]
 [0.11159059 0.88840941]
 [0.57686907 0.42313093]]
M-step: theta =
0.7680988343673212 0.5495359141383477
==========EM Iter: 4==========
E-step: q(z) =
[[0.16170261 0.83829739]
 [0.91290493 0.08709507]
 [0.79426368 0.20573632]
 [0.06633343 0.93366657]
 [0.58710461 0.41289539]]
M-step: theta =
0.7831645842999738 0.5346174541475203
==========EM Iter: 5==========
E-step: q(z) =
[[0.12902034 0.87097966]
 [0.93537835 0.06462165]
 [0.82155069 0.17844931]
 [0.04499518 0.95500482]
 [0.59420506 0.40579494]]
M-step: theta =
0.7910552458637526 0.5262811670299318
==========EM Iter: 6==========
E-step: q(z) =
[[0.11354215 0.88645785]
 [0.94527968 0.05472032]
 [0.83523177 0.16476823]
 [0.03622405 0.96377595]
 [0.59798906 0.40201094]]
M-step: theta =
0.7945325379936995 0.5223904375178746
==========EM Iter: 7==========
E-step: q(z) =
[[0.10708809 0.89291191]
 [0.94933575 0.05066425]
 [0.8412686  0.1587314 ]
 [0.03280939 0.96719061]
 [0.59985308 0.40014692]]
M-step: theta =
0.7959286672497985 0.5207298780860258
==========EM Iter: 8==========
E-step: q(z) =
[[0.10455728 0.89544272]
 [0.95095406 0.04904594]
 [0.84378118 0.15621882]
 [0.0315032  0.9684968 ]
 [0.60074317 0.39925683]]
M-step: theta =
0.7964656379225262 0.5200471890029875
==========EM Iter: 9==========
E-step: q(z) =
```

```
[[0.10359135 0.89640865]
 [0.95159456 0.04840544]
 [0.84480318 0.15519682]
 [0.03100653 0.96899347]
 [0.60115794 0.39884206]]
M-step: theta =
0.7966683078984395 0.5197703896938073
==========EM Iter: 10==========
E-step: q(z) =
[[0.10322699 0.89677301]
 [0.95184768 0.04815232]
 [0.8452156  0.1547844 ]
 [0.03081812 0.96918188]
 [0.60134719 0.39865281]]
M-step: theta =
0.7967441494752118 0.5196586622041123
```





Result:

Thus, the python program for EM for Bayesian networks was executed successfully.

Ex.No:11	**Build Simple Neural Network Models**
Date:	

Aim:

To write a python program to Build simple neural network models.

Procedure:

Step 1 - Import basic libraries.
Step 2 - Importing the dataset.
Step 3 - Data preprocessing.
Step 4 - Training the model.
Step 5 - Testing and evaluation of the model.
Step 6 - Visualizing the model.

Program:

Importing libraries

```
import warnings
warnings.filterwarnings('ignore')

import numpy as np
import matplotlib.pyplot as plt
import keras
from keras.datasets import mnist
from keras.models import Sequential,model_from_json
from keras.layers import Dense
from keras.optimizers import RMSprop
import pylab as plt
```

Using TensorFlow backend.

Keras is the deep learning library that helps you to code Deep Neural Networks with fewer lines of code

---

Import data

```
batch_size = 128
num_classes = 10
epochs = 2

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

# Normalize to 0 to 1 range
x_train /= 255
x_test /= 255

print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz

11132928/11490434 [============================>.] - ETA: 0s60000 train samples

10000 test samples

Visualize Data

```
print("Label:",y_test[2:3])
plt.imshow(x_test[2:3].reshape(28,28), cmap='gray')
plt.show()
```

Note: Images are also considered as numerical matrices

Design a model

```
first_layer_size = 32
model = Sequential()
model.add(Dense(first_layer_size, activation='sigmoid', input_shape=(784,)))
model.add(Dense(32, activation='sigmoid'))
model.add(Dense(32, activation='sigmoid'))
model.add(Dense(num_classes, activation='softmax'))

model.summary()
```

```
_________________________________________________________________
Layer (type)              Output Shape            Param #
=================================================================
dense_1 (Dense)           (None, 32)              25120
_________________________________________________________________
dense_2 (Dense)           (None, 32)              1056
_________________________________________________________________
dense_3 (Dense)           (None, 32)              1056
_________________________________________________________________
dense_4 (Dense)           (None, 10)              330
=================================================================
Total params: 27,562
Trainable params: 27,562
Non-trainable params: 0
```

Weights before Training

```
w = []
for layer in model.layers:
    weights = layer.get_weights()
    w.append(weights)

layer1 = np.array(w[0][0])
```

```
print("Shape of First Layer",layer1.shape)
print("Visualization of First Layer")
fig=plt.figure(figsize=(12, 12))
columns = 8
rows = int(first_layer_size/8)
for i in range(1, columns*rows +1):
    fig.add_subplot(rows, columns, i)
    plt.imshow(layer1[:,i-1].reshape(28,28),cmap='gray')
plt.show()
```

Compiling a Model

```
model.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(),
              metrics=['accuracy'])
```

Training

```
# Write the Training input and output variables, size of the batch, number of epochs
history = model.fit(x_train,y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1)
```

Epoch 1/2

60000/60000 [==============================] - 7s - loss: 1.5842 - acc: 0.5646

Epoch 2/2

60000/60000 [==============================] - 7s - loss: 0.6416 - acc: 0.8247

Testing

```
# Write the testing input and output variables
score = model.evaluate(x_test, y_test, verbose=0)
```



```
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

Test loss: 0.46789013657569883

Test accuracy: 0.8778

Weights after Training

```
w = []
for layer in model.layers:
    weights = layer.get_weights()
    w.append(weights)

layer1 = np.array(w[0][0])
print("Shape of First Layer",layer1.shape)
print("Visualization of First Layer")
fig=plt.figure(figsize=(12, 12))
columns = 8
rows = int(first_layer_size/8)
for i in range(1, columns*rows +1):
    fig.add_subplot(rows, columns, i)
    plt.imshow(layer1[:,i-1].reshape(28,28),cmap='gray')
plt.show()
```

Take away

This internal representation reflects Latent Variables

Each of the nodes will look for a specific pattern in the input

A node will get activated if input is similar to the feature it looks for

Each node is unique and often orthogonal to each other

Prediction

```
# Write the index of the test sample to test
prediction = model.predict(x_test[])
prediction = prediction[0]
print('Prediction\n',prediction)
print('\nThresholded output\n',(prediction>0.5)*1)
```

Ground truth

```
# Write the index of the test sample to show

plt.imshow(x_test[].reshape(28,28),cmap='gray')

plt.show()
```

User Input

```
# Load library
import cv2
import numpy as np
from matplotlib import pyplot as plt

# Load image in color
image_bgr = cv2.imread('digit.jpg', cv2.IMREAD_COLOR)
# Convert to RGB
image_rgb = cv2.cvtColor(image_bgr, cv2.COLOR_BGR2RGB)

# Show image
plt.imshow(image_rgb), plt.axis("off")
plt.show()
```

Convert to grayscale and resize

```
# Load image as grayscale

# Write the path to the image
```

```
image = cv2.imread('.jpg', cv2.IMREAD_GRAYSCALE)
image_resized = cv2.resize(image, (28, 28))
# Show image
plt.imshow(image_resized, cmap='gray'), plt.axis("off")
plt.show()
```

Prediction

```
prediction = model.predict(image_resized.reshape(1,784))
print('Prediction Score:\n',prediction[0])
thresholded = (prediction>0.5)*1
print('\nThresholded Score:\n',thresholded[0])
print('\nPredicted Digit:\n',np.where(thresholded == 1)[1][0])
```

Prediction Score:

[1.4401099e-05 5.0795502e-03 3.8524144e-03 9.4846159e-01 1.9869357e-04

3.1823639e-02 1.8906208e-04 1.7704907e-03 6.8866094e-03 1.7235276e-03]

Thresholded Score:

[0 0 0 1 0 0 0 0 0 0]

Predicted Digit:

3

Part 2: Saving, Loading and Retraining Models

Saving a model

```
# serialize model to JSON
model_json = model.to_json()
```



```
# Write the file name of the model

with open("model.json", "w") as json_file:
    json_file.write(model_json)

# serialize weights to HDF5
# Write the file name of the weights

model.save_weights("model.h5")
print("Saved model to disk")
```

Saved model to disk

Loading a model

```
# load json and create model

# Write the file name of the model

json_file = open('model.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json)

# load weights into new model
# Write the file name of the weights

loaded_model.load_weights("model.h5")
print("Loaded model from disk")
```

Loaded model from disk

Retraining a model

```
loaded_model.compile(loss='categorical_crossentropy', optimizer=RMSprop(), metrics=['accura
cy'])
```



```
history = loaded_model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, verbose=1)
score = loaded_model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

Epoch 1/2

60000/60000 [==============================] - 13s - loss: 0.3426 - acc: 0.9081

Epoch 2/2

60000/60000 [==============================] - 13s - loss: 0.2633 - acc: 0.9263

Test loss: 0.23180415102243423

Test accuracy: 0.9338

Saving a model and resuming the training later is the great relief in training large neural networks !

Part 3: Activation Functions

Sigmoid Activation Function

```
model = Sequential()
model.add(Dense(8, activation='sigmoid', input_shape=(784,)))
model.add(Dense(8, activation='sigmoid'))
model.add(Dense(num_classes, activation='softmax'))

model.summary()
model.compile(loss='categorical_crossentropy',
            optimizer=RMSprop(),
            metrics=['accuracy'])

history = model.fit(x_train, y_train,
                batch_size=batch_size,
                epochs=epochs,
                verbose=1,
                validation_data=(x_test, y_test))
```



```
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

____

Layer (type)   Output Shape   Param #

===

dense_5 (Dense)   (None, 8)   6280

____

dense_6 (Dense)   (None, 8)   72

____

dense_7 (Dense)   (None, 10)   90

===

Total params: 6,442

Trainable params: 6,442

Non-trainable params: 0

____

Train on 60000 samples, validate on 10000 samples

Epoch 1/2

60000/60000 [==============================] - 8s - loss: 2.0877 - acc: 0.4284 - val_loss: 1.8024 - val_acc: 0.6627

Epoch 2/2

60000/60000 [==============================] - 7s - loss: 1.5510 - acc: 0.7029 - val_loss: 1.3097 - val_acc: 0.7450

Test loss: 1.30969395942688

Test accuracy: 0.745

Relu Activation Function

```
# Write your code here

# Use the same model design from the above cell
```

What are your findings?

Other Activation Functions

```
model.add(Dense(8, activation='tanh'))

model.add(Dense(8, activation='linear'))

model.add(Dense(8, activation='hard_sigmoid'))
```

Tips

Relu is commonly used in most hidden layers

In case of dead neurons, use leaky Relu

Part 4: Design Choices in Neural Networks

Design a model with Low Number of Nodes. For Example 8

```
first_layer_size = 8

model = Sequential()
model.add(Dense(first_layer_size, activation='sigmoid', input_shape=(784,)))
model.add(Dense(32, activation='sigmoid'))
model.add(Dense(num_classes, activation='softmax'))

model.summary()
model.compile(loss='categorical_crossentropy',
            optimizer=RMSprop(),
            metrics=['accuracy'])

history = model.fit(x_train, y_train,
```



```
                batch_size=batch_size,
                epochs=epochs,
                verbose=1,
                validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])

w = []
for layer in model.layers:
    weights = layer.get_weights()
    w.append(weights)

layer1 = np.array(w[0][0])
print("Shape of First Layer",layer1.shape)
print("Visualization of First Layer")

import matplotlib.pyplot as plt
fig=plt.figure(figsize=(16, 16))
columns = 8
rows = int(first_layer_size/8)
for i in range(1, columns*rows +1):
    fig.add_subplot(rows, columns, i)
    plt.imshow(layer1[:,i-1].reshape(28,28),cmap='gray')
plt.show()
```

Design a model with Higher Number of Nodes. For example 128

```
# Write your code here

# Use the same layer design from the above cell
```

Lower number of Layers. For example 1 hidden layer

```
model = Sequential()
model.add(Dense(4, activation='relu', input_shape=(784,)))
model.add(Dense(num_classes, activation='softmax'))
```

```
model.add(Dense(num_classes,  activation='softmax'))

model.summary()
model.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(),
              metrics=['accuracy'])

history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_13 (Dense)             (None, 4)                 3140
_________________________________________________________________
dense_14 (Dense)             (None, 10)                50
_________________________________________________________________
dense_15 (Dense)             (None, 10)                110
=================================================================
Total params: 3,300

Trainable params: 3,300

Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples

Epoch 1/2

60000/60000 [==============================] - 5s - loss: 2.0010 - acc: 0.3352 - val_loss: 1.7679 - val_acc: 0.4473
```

Epoch 2/2

60000/60000 [==============================] - 4s - loss: 1.5709 - acc: 0.5450 - val_loss: 1.3916 - val_acc: 0.5919

Test loss: 1.3915847587585448

Test accuracy: 0.5919

Result:

Thus, the python program for simple NN models was executed successfully.

Ex.No:12	**Build Deep Learning Neural Network models**
Date:	

Aim:

To write a python program to Build deep learning neural network models.

Procedure:

Step 1 - Import basic libraries.

Step 2 - Importing the dataset.

Step 3 - Data preprocessing.

- From keras library we are going to use image preprocessing task, to normalize the image pixel values in between 0 to 1.
- Model is imported to load various Neural Network models such as Sequential.
- We are going to use Stochastic Gradient Descent (SGD) as a optimizer
- Keras layers such as Dense, Flatten, Conv2D and MaxPooling is used to implement the CNN model.

Step 4 - Training the model.

Step 5 - Testing and evaluation of the model.

Step 6 - Visualizing the model.

Program:

Convolution Neural Networks are mainly use for large size input data such as Image data.

- Convolution Neural Networks (CNNs) use parameter sharing.
- Small pattern detectors called filters are used to convolve over the entire image.
- These filters are learned through NN training in the same way as in fully connected networks.
- Just like a hidden layer in a fully connected layer, convolution layers are used in CNNs.
- To handle large size of image data, pooling layers are introduced.
- Normalization layers were used in early CNN architectures, but due to their minimal impact, they are not much used in the present CNNs.

Dataset Link : :https://github.com/spMohanty/PlantVillage-Dataset/tree/master/raw/color We split the dataset into training validation and testing sample

A. Data Preprocessing

In [1]:

```
importnumpyasnp
importkeras
```

```
fromkerasimportmodels
importmatplotlib.pyplotasplt
fromkeras.preprocessingimportimage
fromkeras.preprocessing.imageimportImageDataGenerator
fromkeras.modelsimportModel
fromkeras.optimizersimportSGD
fromkerasimportlayers
fromkeras.layersimportDense, Flatten, Conv2D, MaxPooling2D
fromkerasimportInput
```

Using TensorFlow backend.

** A2. Loading the training and testing data and defining the basic parameters **

- We are resizing the input image to 64 * 64
- In the dataset : Training Set : 70% Validation Set : 20% Test Set : 10%

In [2]:

```
# Normalize training and validation data in the range of 0 to 1
train_datagen=ImageDataGenerator(rescale=1./255)
validation_datagen=ImageDataGenerator(rescale=1./255)
test_datagen=ImageDataGenerator(rescale=1./255)

# Read the training sample and set the batch size
train_generator=train_datagen.flow_from_directory(
'cellimage/train/',
target_size=(64, 64),
batch_size=16,
class_mode='categorical')

# Read Validation data from directory and define target size with batch size
validation_generator=validation_datagen.flow_from_directory(
'cellimage/val/',
target_size=(64, 64),
batch_size=16,
class_mode='categorical',
shuffle=False)

test_generator=test_datagen.flow_from_directory(
'cellimage/test/',
target_size=(64, 64),
batch_size=1,
class_mode='categorical',
shuffle=False)
```

Found 2217 images belonging to 4 classes.
Found 635 images belonging to 4 classes.
Found 319 images belonging to 4 classes.

B. Model Building

- We are going to use 2 convolution layers with 3*3 filer and relu as an activation function
- Then max pooling layer with 2*2 filter is used
- After that we are going to use Flatten layer
- Then Dense layer is used with relu function
- In the output layer softmax function is used with 4 neurons as we have four class dataset.
- model.summary() is used to check the overall architecture of the model with number of learnable parameters in each

B1. Model Definition

In [3]:

```
# Create the model
model=models.Sequential()
# Add new layers
model.add(Conv2D(128, kernel_size=(3,3), activation='relu', input_shape=(64,64,3)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(64, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(64, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(layers.Flatten())
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(4, activation='softmax'))
model.summary()
```

```
WARNING:tensorflow:From C:\Users\sozhan\Anaconda2\lib\site-packages\tensorflow\python\framework\op_def_library.py:263: colocate_with (from tensorflow.python.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From C:\Users\sozhan\Anaconda2\lib\site-packages\keras\backend\tensorflow_backend.py:1264: calling reduce_prod_v1 (from tensorflow.python.ops.math_ops) with keep_dims is deprecated and will be removed in a future version.
Instructions for updating:
keep_dimsis deprecated, use keepdims instead
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 62, 62, 128)       3584
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 31, 31, 128)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 29, 29, 64)        73792
_________________________________________________________________
```

```
max_pooling2d_2 (MaxPooling2 (None, 14, 14, 64)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 12, 12, 64)       36928
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 6, 6, 64)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2304)             0
_________________________________________________________________
dense_1 (Dense)              (None, 32)               73760
_________________________________________________________________
dense_2 (Dense)              (None, 4)                132
=================================================================
Total params: 188,196
Trainable params: 188,196
Non-trainable params: 0
_________________________________________________________________
```

B2. Compile the model with SGD(Stochastic Gradient Descent) and train it with 10 epochs.

In [4]:

```
sgd=SGD(lr=0.01,decay=1e-6, momentum=0.9, nesterov=True)
# We are going to use accuracy metrics and cross entropy loss as performance parameters
model.compile(sgd, loss='categorical_crossentropy', metrics=['acc'])
# Train the model
history=model.fit_generator(train_generator,
steps_per_epoch=train_generator.samples/train_generator.batch_size,
epochs=10,
validation_data=validation_generator,
validation_steps=validation_generator.samples/validation_generator.batch_size,
verbose=1)
```

```
WARNING:tensorflow:From C:\Users\sozhan\Anaconda2\lib\site-packages\keras\backend\tensorflow_backend.py:2885: calling reduce_sum_v1 (from tensorflow.python.ops.math_ops) with keep_dims is deprecated and will be removed in a future version.
Instructions for updating:
keep_dimsis deprecated, use keepdims instead
WARNING:tensorflow:From C:\Users\sozhan\Anaconda2\lib\site-packages\tensorflow\python\ops\math_ops.py:3066: to_int32 (from tensorflow.python.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
WARNING:tensorflow:Variable *= will be deprecated. Use `var.assign(var * other)` if you want assignment to the variable value or `x = x * y` if you want a new python Tensor object.
Epoch 1/10
139/138 [==============================] - 51s 367ms/step - loss: 0.9863 - acc: 0.6043 - val_loss: 0.6350 - val_acc: 0.7591
Epoch 2/10
```

139/138 [==============================] - 47s 336ms/step - loss: 0.5411 - acc: 0.7947 - val_loss: 0.4170 - val_acc: 0.8441
Epoch 3/10
139/138 [==============================] - 48s 343ms/step - loss: 0.4594 - acc: 0.8278 - val_loss: 0.6648 - val_acc: 0.7307
Epoch 4/10
139/138 [==============================] - 46s 328ms/step - loss: 0.3686 - acc: 0.8642 - val_loss: 0.3508 - val_acc: 0.8709
Epoch 5/10
139/138 [==============================] - 46s 333ms/step - loss: 0.3162 - acc: 0.8855 - val_loss: 0.3843 - val_acc: 0.8661
Epoch 6/10
139/138 [==============================] - 48s 349ms/step - loss: 0.2712 - acc: 0.9039 - val_loss: 0.3046 - val_acc: 0.8929
Epoch 7/10
139/138 [==============================] - 46s 332ms/step - loss: 0.2601 - acc: 0.9005 - val_loss: 0.2986 - val_acc: 0.9039
Epoch 8/10
139/138 [==============================] - 46s 332ms/step - loss: 0.2168 - acc: 0.9214 - val_loss: 0.3035 - val_acc: 0.8945
Epoch 9/10
139/138 [==============================] - 44s 316ms/step - loss: 0.2203 - acc: 0.9213 - val_loss: 0.2454 - val_acc: 0.9134
Epoch 10/10
139/138 [==============================] - 54s 389ms/step - loss: 0.2047 - acc: 0.9308 - val_loss: 0.2947 - val_acc: 0.9008

B3. Saving the model

In [5]:

```
model.save('cnn_classification.h5')
```

B4. Loading the Model

In [6]:

```
model=models.load_model('cnn_classification.h5')
print(model)
<keras.models.Sequential object at 0x0000005BA45C1518>
```

B5. Saving weignts of model

In [7]:

```
model.save_weights('cnn_classification.h5')
```

B6. Loading the Model weights

In [8]:

```
model.load_weights('cnn_classification.h5')
```

C. Performance Measures

**Now we are going to plot the accuracy and loss*

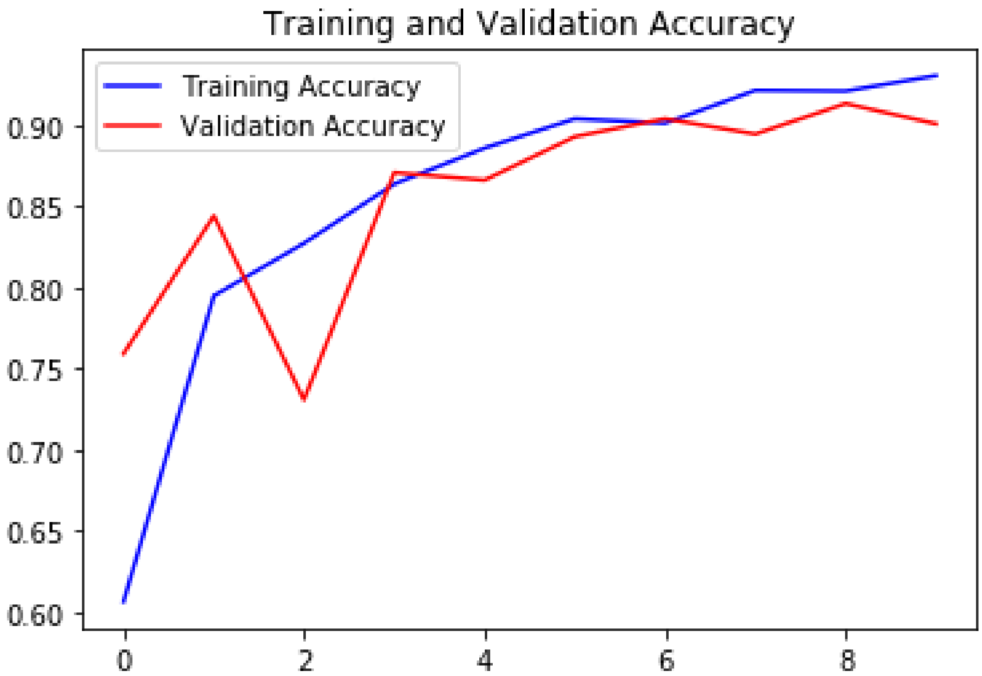
In [9]:

```
train_acc=history.history['acc']
val_acc=history.history['val_acc']
train_loss=history.history['loss']
val_loss=history.history['val_loss']
print(train_acc)
print(val_acc)
print(train_loss)
print(val_loss)
```

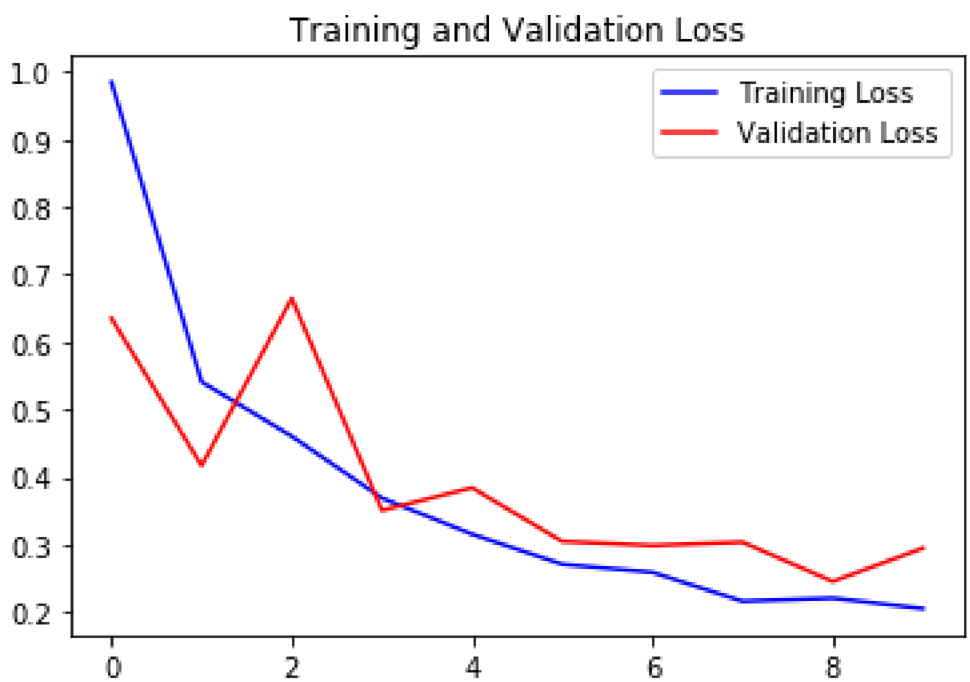
[0.6062246278755075, 0.7947677041584165, 0.8272440234551195, 0.8637798827244023, 0.8858818223361341, 0.9039242219484008, 0.9012178620562986, 0.921515561596574, 0.92106450157871, 0.9305367613892648]
[0.7590551185795641, 0.8440944886583043, 0.7307086613234572, 0.8708661422016114, 0.8661417322834646, 0.8929133858267716, 0.9039370078740158, 0.8944881889763779, 0.9133858267716536, 0.9007874015748032]
[0.9849027604415818, 0.5411215644190319, 0.4603504691849327, 0.3689646118656171, 0.3152961805429231, 0.27086145290663827, 0.25899119408323146, 0.21610905267684696, 0.220770583645578, 0.20524998439873293]
[0.6350463369230586, 0.4170022392836143, 0.664773770016948, 0.3508223737318685, 0.38431625602048214, 0.30464723109905645, 0.29862876056920823, 0.3035450204385547, 0.2454165400482538, 0.29468511782997237]

In [10]:

```
epochs=range(len(train_acc))
plt.plot(epochs, train_acc, 'b', label='Training Accuracy')
plt.plot(epochs, val_acc, 'r', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.figure()
plt.show()

plt.plot(epochs, train_loss, 'b', label='Training Loss')
plt.plot(epochs, val_loss, 'r', label='Validation Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.show()
```



<Figure size 432x288 with 0 Axes>



Model Testing

In [11]:

```
# Get the filenames from the generator
fnames=test_generator.filenames

# Get the ground truth from generator
```



```
ground_truth=test_generator.classes

# Get the label to class mapping from the generator
label2index=test_generator.class_indices

# Getting the mapping from class index to class label
idx2label=dict((v,k) fork,vinlabel2index.items())

# Get the predictions from the model using the generator
predictions=model.predict_generator(test_generator,
steps=test_generator.samples/test_generator.batch_size,verbose=1)
predicted_classes=np.argmax(predictions,axis=1)

errors=np.where(predicted_classes!=ground_truth)[0]
print("No of errors = {}/{}".format(len(errors),test_generator.samples))
```

319/319 [==============================] - 4s 14ms/step
No of errors = 29/319

Assignemnt

**You have to load the weights of previous model and with the help of previous weights try to create a CNN model with one more convolution layers. You have to train only after the newly added convolution layers of the neural network. **

Hint : Use model.load_weights('weights.h5', by_name=True)
In [15]:

```
new_model=models.Sequential()
model.load_weights('cnn_classification.h5', by_name=True)
new_model.add(Conv2D(128, kernel_size=(3,3), activation='relu', input_shape=(64,64,3)))
new_model.add(MaxPooling2D(pool_size=(2,2)))
new_model.add(Conv2D(64, kernel_size=(3,3), activation='relu'))
new_model.add(MaxPooling2D(pool_size=(2,2)))
new_model.add(Conv2D(64, kernel_size=(3,3), activation='relu'))
new_model.add(MaxPooling2D(pool_size=(2,2)))
new_model.add(Conv2D(32, kernel_size=(3,3), activation='relu'))
new_model.add(Conv2D(32, kernel_size=(3,3), activation='relu'))
new_model.add(layers.Flatten())
new_model.add(layers.Dense(32, activation='relu'))
new_model.add(layers.Dense(4, activation='softmax'))
new_model.summary()
```

```
_________________________________________________________________
Layer (type)              Output Shape            Param #
=================================================================
conv2d_4 (Conv2D)         (None, 62, 62, 128)     3584
_________________________________________________________________
```

```
max_pooling2d_4 (MaxPooling2 (None, 31, 31, 128)       0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 29, 29, 64)        73792
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 12, 12, 64)        36928
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 4, 4, 32)          18464
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 2, 2, 32)          9248
_________________________________________________________________
flatten_2 (Flatten)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 32)                4128
_________________________________________________________________
dense_4 (Dense)              (None, 4)                 132
=================================================================
Total params: 146,276
Trainable params: 146,276
Non-trainable params: 0
_________________________________________________________________
```

Training the model after 5rd layer

In[8]

```
forlayerinnew_model.layers[:6]:
layer.trainable=False

forlayerinnew_model.layers:
print(layer, layer.trainable)
new_model.summary()
<keras.layers.convolutional.Conv2D object at 0x0000005BA10D8F98> False
<keras.layers.pooling.MaxPooling2D object at 0x0000005BA2247C18> False
<keras.layers.convolutional.Conv2D object at 0x0000005BA22ED748> False
<keras.layers.pooling.MaxPooling2D object at 0x0000005BA45C1160> False
<keras.layers.convolutional.Conv2D object at 0x0000005BA460E860> False
<keras.layers.pooling.MaxPooling2D object at 0x0000005BA460EF28> False
<keras.layers.convolutional.Conv2D object at 0x0000005BA461FF28> True
<keras.layers.convolutional.Conv2D object at 0x0000005BA4633828> True
<keras.layers.core.Flatten object at 0x0000005BA46339B0> True
<keras.layers.core.Dense object at 0x0000005BA9C1A6A0> True
<keras.layers.core.Dense object at 0x0000005BA9C35B00> True
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
```

```
conv2d_4 (Conv2D)            (None, 62, 62, 128)       3584
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 31, 31, 128)       0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 29, 29, 64)        73792
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 12, 12, 64)        36928
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 4, 4, 32)          18464
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 2, 2, 32)          9248
_________________________________________________________________
flatten_2 (Flatten)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 32)                4128
_________________________________________________________________
dense_4 (Dense)              (None, 4)                 132
=================================================================
Total params: 146,276
Trainable params: 31,972
Non-trainable params: 114,304
_________________________________________________________________
```

In [ ]:

```
# Here we are changing the learning rate from 0.001 to 0.01

sgd=SGD(lr=0.01,decay=1e-6, momentum=0.9, nesterov=True)
# We are going to use accuracy metrics and cross entropy loss as performance parameters
new_model.compile(sgd, loss='categorical_crossentropy', metrics=['acc'])
#  Train  the  model
new_history=new_model.fit_generator(train_generator,
steps_per_epoch=train_generator.samples/train_generator.batch_size,
epochs=2,
validation_data=validation_generator,
validation_steps=validation_generator.samples/validation_generator.batch_size,
verbose=1)
```

```
Epoch 1/2
139/138 [==============================] - 25s 182ms/step - loss: 1.2130 - acc: 0.5108 - val_loss: 1.1815 - val_acc: 0.5181
Epoch 2/2
139/138 [==============================] - 24s 171ms/step - loss: 1.1256 - acc: 0.5256 - val_loss: 0.9504 - val_acc: 0.5685
```

C. Performance Measures

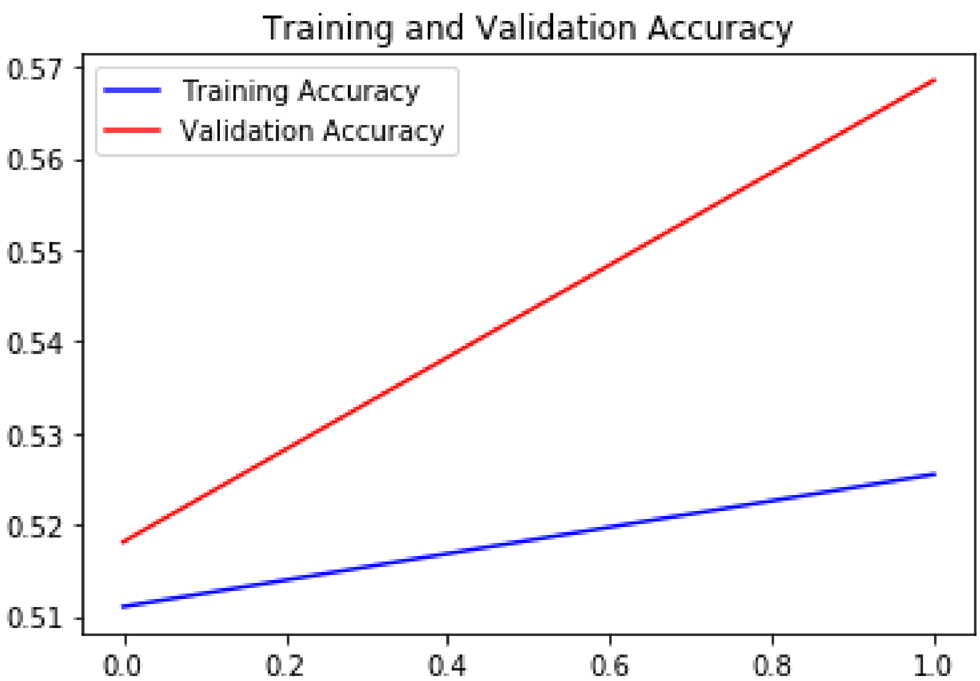
Now we are going to plot the accuracy and loss

In [18]:

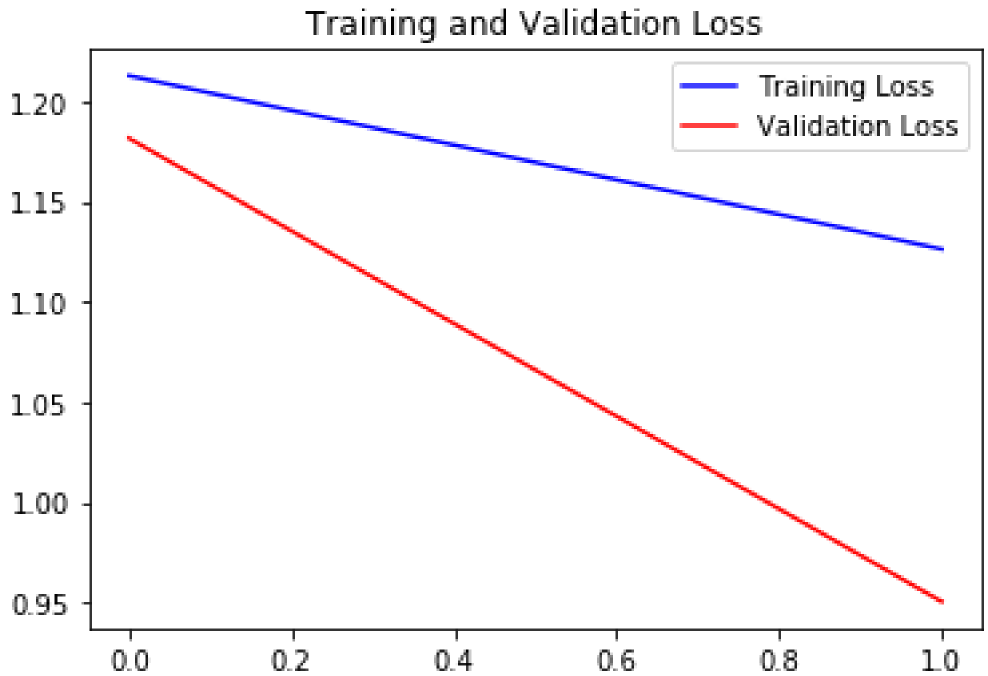
```
train_acc=new_history.history['acc']
val_acc=new_history.history['val_acc']
train_loss=new_history.history['loss']
val_loss=new_history.history['val_loss']
```

In [20]:

```
epochs=range(len(train_acc))
plt.plot(epochs, train_acc, 'b', label='Training Accuracy')
plt.plot(epochs, val_acc, 'r', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.figure()
plt.show()

plt.plot(epochs, train_loss, 'b', label='Training Loss')
plt.plot(epochs, val_loss, 'r', label='Validation Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.show()
```



<Figure size 432x288 with 0 Axes>



Model Testing

In [21]:

```
# Get the filenames from the generator
fnames=test_generator.filenames

# Get the ground truth from generator
ground_truth=test_generator.classes

# Get the label to class mapping from the generator
label2index=test_generator.class_indices

# Getting the mapping from class index to class label
idx2label=dict((v,k) fork,vinlabel2index.items())

# Get the predictions from the model using the generator
predictions=model.predict_generator(test_generator,
steps=test_generator.samples/test_generator.batch_size,verbose=1)
predicted_classes=np.argmax(predictions,axis=1)

errors=np.where(predicted_classes!=ground_truth)[0]
print("No of errors = {}/{}".format(len(errors),test_generator.samples))
```

```
319/319 [==============================] - 4s 12ms/step
No of errors = 29/319
```

Result:

Thus, the python program for build deep learning NN models was executed successfully.

ECE Department

1st Semester | 2nd Semester | 3rd Semester | 4th Semester

5th Semester | 6th Semester | 7th Semester | 8th Semester

Click on Subject/Paper under Semester to enter.

1st Semester
- Professional English - I - HS3152
- Matrices and Calculus - MA3151
- Engineering Physics - PH3151
- Engineering Chemistry - CY3151
- Problem Solving and Python Programming - GE3151

2nd Semester
- Professional English - II - HS3252
- Statistics and Numerical Methods - MA3251
- Engineering Graphics - GE3251
- Physics for Electronics Engg - PH3254
- Basic Electrical & Instru Engg - BE3254
- Circuit Analysis - EC3251

3rd Semester
- Random Process and Linear Algebra - MA3355
- C Programming and Data Structures - CS3353
- Signals and Systems - EC3354
- Electronic Devices and Circuits - EC3353
- Control Systems - EC3351
- Digital Systems Design - EC3352

4th Semester
- Electromagnetic Fields - EC3452
- Networks and Security - EC3401
- Linear Integrated Circuits - EC3451
- Digital Signal Processing - EC3492
- Communication Systems - EC3491
- Environmental Sciences and Sustainability - GE3451

5th Semester
- Wireless Communication - EC3501
- VLSI and Chip Design - EC3552
- Transmission Lines and RF Systems - EC3551
- Elective 1
- Elective 2
- Elective 3

6th Semester
- Embedded Systems and IOT Design - ET3491
- Artificial Intelligence and Machine Learning - CS3491
- Open Elective-1
- Elective-4
- Elective-5
- Elective-6

7th Semester
- Human Values and Ethics - GE3791
- Open Elective 2
- Open Elective 3
- Open Elective 4

8th Semester
- Project Work / Intership

Information Technology

1st Semester | 2nd Semester | 3rd Semester | 4th Semester

5th Semester | 6th Semester | 7th Semester | 8th Semester

Click on Subject/Paper under Semester to enter.

1st Semester

- Professional English - I - HS3152
- Matrices and Calculus - MA3151
- Engineering Physics - PH3151
- Engineering Chemistry - CY3151
- Problem Solving and Python Programming - GE3151

2nd Semester

- Professional English - II - HS3252
- Statistics and Numerical Methods - MA3251
- Engineering Graphics - GE3251
- Physics for Information Science - PH3256
- Basic Electrical and Electronics Engineering - BE3251
- Programming in C - CS3251

3rd Semester

- Discrete Mathematics - MA3354
- Digital Principles and Computer Organization - CS3351
- Foundation of Data Science - CS3352
- Data Structures and Algorithms - CD3291
- Object Oriented Programming - CS3391

4th Semester

- Environmental Sciences and Sustainability - GE3451
- Theory of Computation - CS3452
- Artificial Intelligence and Machine Learning - CS3491
- Database Management System - CS3492
- Web Essentials - IT3401
- Introduction to Operating Systems - CS3451

5th Semester

- Computer Networks - CS3591
- Full Stack Web Development - IT3501
- Distributed Computing - CS3551
- Embedded Systems and IoT - CS3691
- Elective 1
- Elective 2

6th Semester

- Object Oriented Software Engineering - CCS356
- Open Elective-1
- Elective-3
- Elective-4
- Elective-5
- Elective-6

7th Semester

- Human Values and Ethics - GE3791
- Open Elective 2
- Open Elective 3
- Open Elective 4
- Management Elective

8th Semester

- Project Work / Intership