SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

FACULTY OF ENGINEERING & TECHNOLOGY

(Formerly SRM University, Under section 3 of UGC Act, 1956)
S.R.M. NAGAR, KATTANKULATHUR –603 203,
KANCHEEPURAM DISTRICT

SCHOOL OF COMPUTING
DEPARTMENT OF COMPUTER SCIENCE

18CSE305J - ARTIFICIAL INTELLIGENCE LAB MANUAL

Name: Aryan Jalla
Reg No: RA1911003010729

INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

COLLEGE OF ENGINEERING & TECHNOLOGY
SRM INSTITUTE OF SCIENCE & TECHNOLOGY
S.R.M. NAGAR, KATTANKULATHUR - 603203
Chengalpattu District

BONAFIDE CERTIFICATE

Register No________________

Certified to be the bonafide record of work done by ______________________________ of ______________________ B.Tech Degree course in the Practical ____________________________________ in **SRM INSTITUTE OF SCIENCE & TECHNOLOGY,** Kattankulathur during the academic year ____________.

FACULTY INCHARGE

DATE:

HEAD OF THE DEPARTMENT

Submitted for University Examination held in ________________________, in ____ **SRM INSTITUTE OF SCIENCE & TECHNOLOGY,** Kattankulathur.

EXAMINER - I

EXAMINER - II

Aryan Jalla

RA1911003010729
CSE C2

Experiment 1

Implementation of Toy Problem

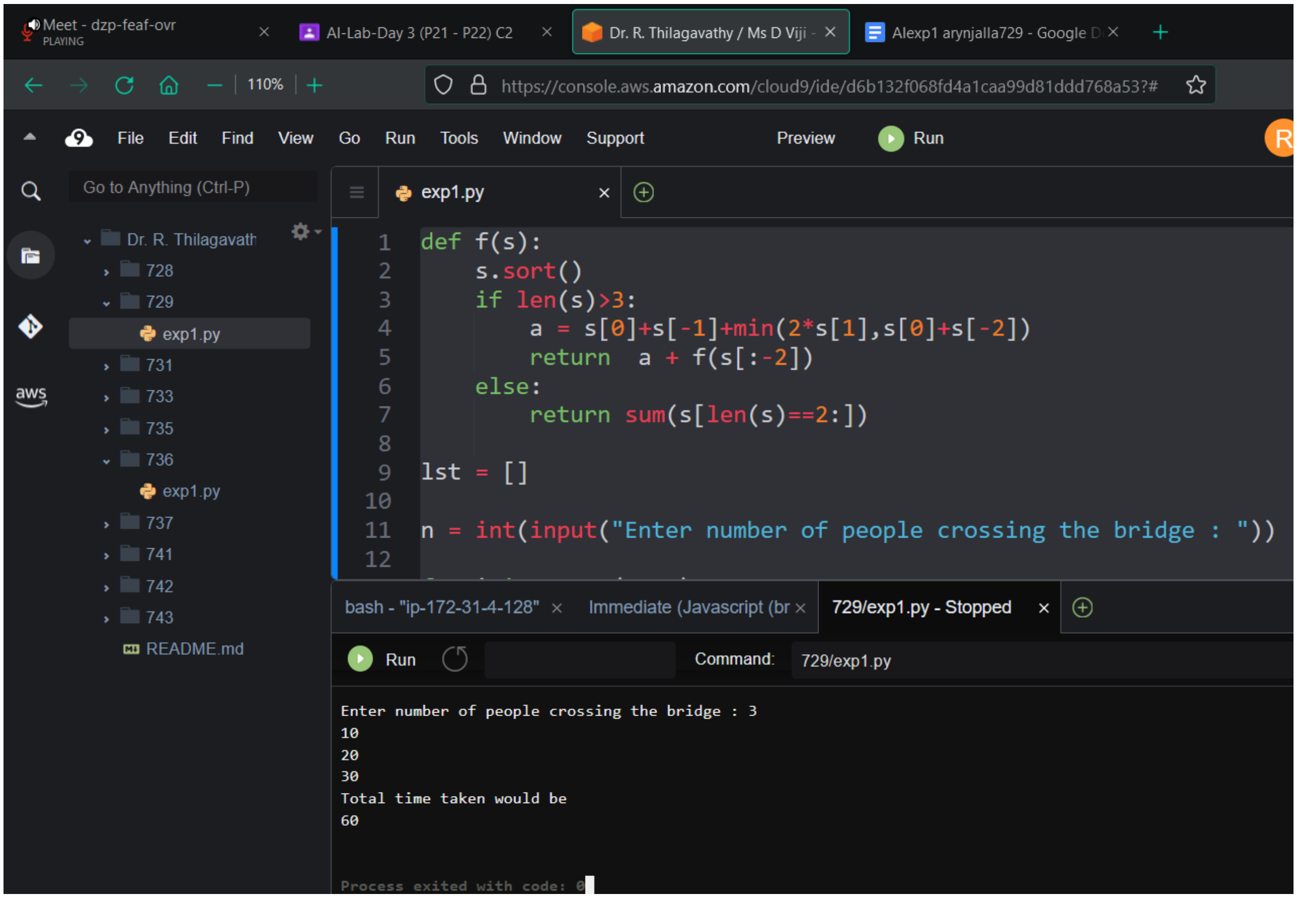
Aim:
To implement a toy problem using python programming language.

Problem Title:
torch and bridge puzzle problem

Problem Statement:
Given an array of positive distinct integer denoting the crossing time of 'n' people. These 'n' people are standing at one side of bridge. Bridge can hold at max two people at a time. When two people cross the bridge, they must move at the slower person's pace. Find the minimum total time in which all persons can cross the bridge.

Code:
```
def f(s):
     s.sort()
     if len(s)>3:
     a = s[0]+s[-1]+min(2*s[1],s[0]+s[-2])
     return  a + f(s[:-2])
     else:
     return sum(s[len(s)==2:])

lst = []

n = int(input("Enter number of people crossing the bridge : "))

for i in range(0, n):
     ele = int(input())

     lst.append(ele)

print("Total time taken would be ")
print(f(lst))
```

Input and Output:



Result:

Implementation of toy problem(torch and bridge puzzle problem) is studied and coded using python programming language successfully.

Aryan Jalla

RA1911003010729
CSE C2

Experiment 2

Minimum Spanning Tree

Aim:
To implement an agent problem using python programming language.

Problem Title:
Kruskal's minimum spanning tree

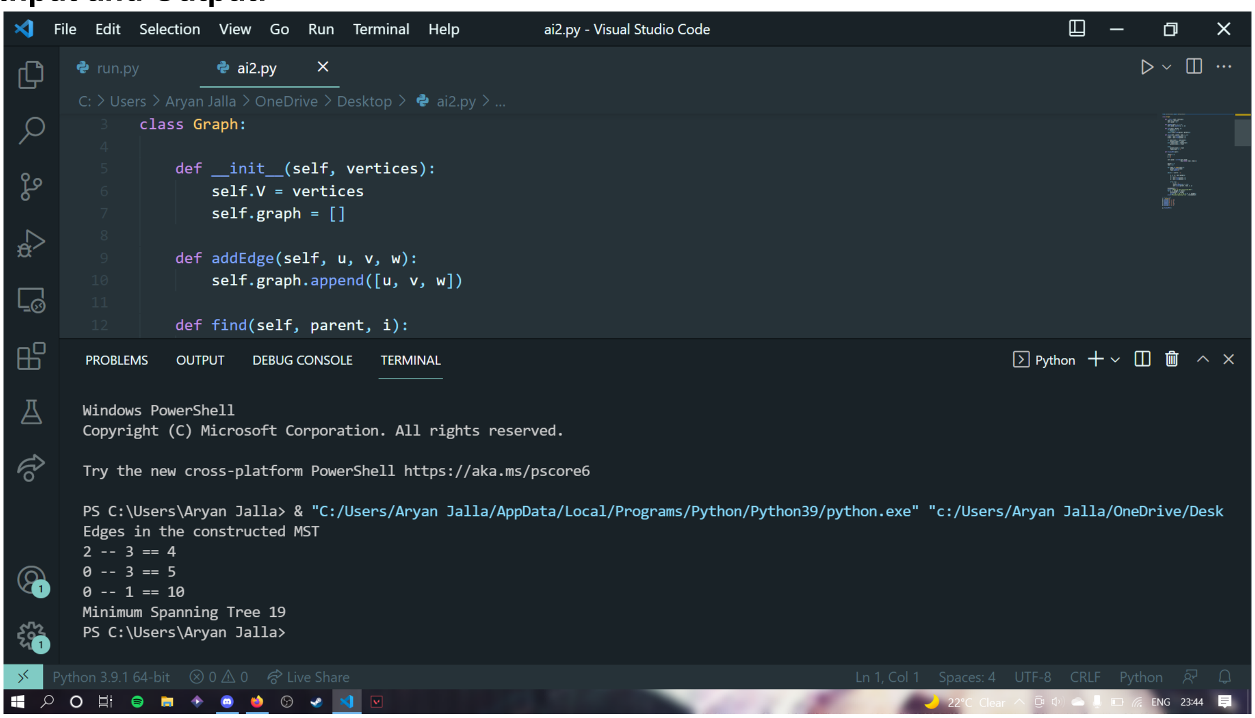
Problem Statement:
Given a connected and undirected graph, a *spanning tree* of that graph is a subgraph that is a tree and connects all the vertices together. A single graph can have many different spanning trees. A *minimum spanning tree (MST)* or minimum weight spanning tree for a weighted, connected, undirected graph is a spanning tree with a weight less than or equal to the weight of every other spanning tree. The weight of a spanning tree is the sum of weights given to each edge of the spanning tree.
How many edges does a minimum spanning tree has?
A minimum spanning tree has (V – 1) edges where V is the number of vertices in the given graph.

Code:
```
from collections import defaultdict
class Graph:

    def __init__(self, vertices):
        self.V = vertices
        self.graph = []

    def addEdge(self, u, v, w):
        self.graph.append([u, v, w])

    def find(self, parent, i):
        if parent[i] == i:
            return i
        return self.find(parent, parent[i])
    def union(self, parent, rank, x, y):
```

```python
        xroot = self.find(parent, x)
        yroot = self.find(parent, y)

        if rank[xroot] < rank[yroot]:
            parent[xroot] = yroot
        elif rank[xroot] > rank[yroot]:
            parent[yroot] = xroot
        else:
            parent[yroot] = xroot
            rank[xroot] += 1

    def KruskalMST(self):
        result = []
        i = 0
        e = 0

        self.graph = sorted(self.graph,
                            key=lambda item: item[2])

        parent = []
        rank = []

        for node in range(self.V):
            parent.append(node)
            rank.append(0)

        while e < self.V - 1:

            u, v, w = self.graph[i]
            i = i + 1
            x = self.find(parent, u)
            y = self.find(parent, v)

            if x != y:
                e = e + 1
                result.append([u, v, w])
                self.union(parent, rank, x, y)

        minimumCost = 0
```

```python
        print ("Edges in the constructed MST")
        for u, v, weight in result:
            minimumCost += weight
            print("%d -- %d == %d" % (u, v, weight))
        print("Minimum Spanning Tree" , minimumCost)

# Driver code
g = Graph(4)
g.addEdge(0, 1, 10)
g.addEdge(0, 2, 6)
g.addEdge(0, 3, 5)
g.addEdge(1, 3, 15)
g.addEdge(2, 3, 4)

g.KruskalMST()
```

Input and Output:



Result:
Implementation of agent problem (minimum spanning) is studied and coded using python programming language successfully.

Aryan Jalla
RA1911003010729
CSE C2

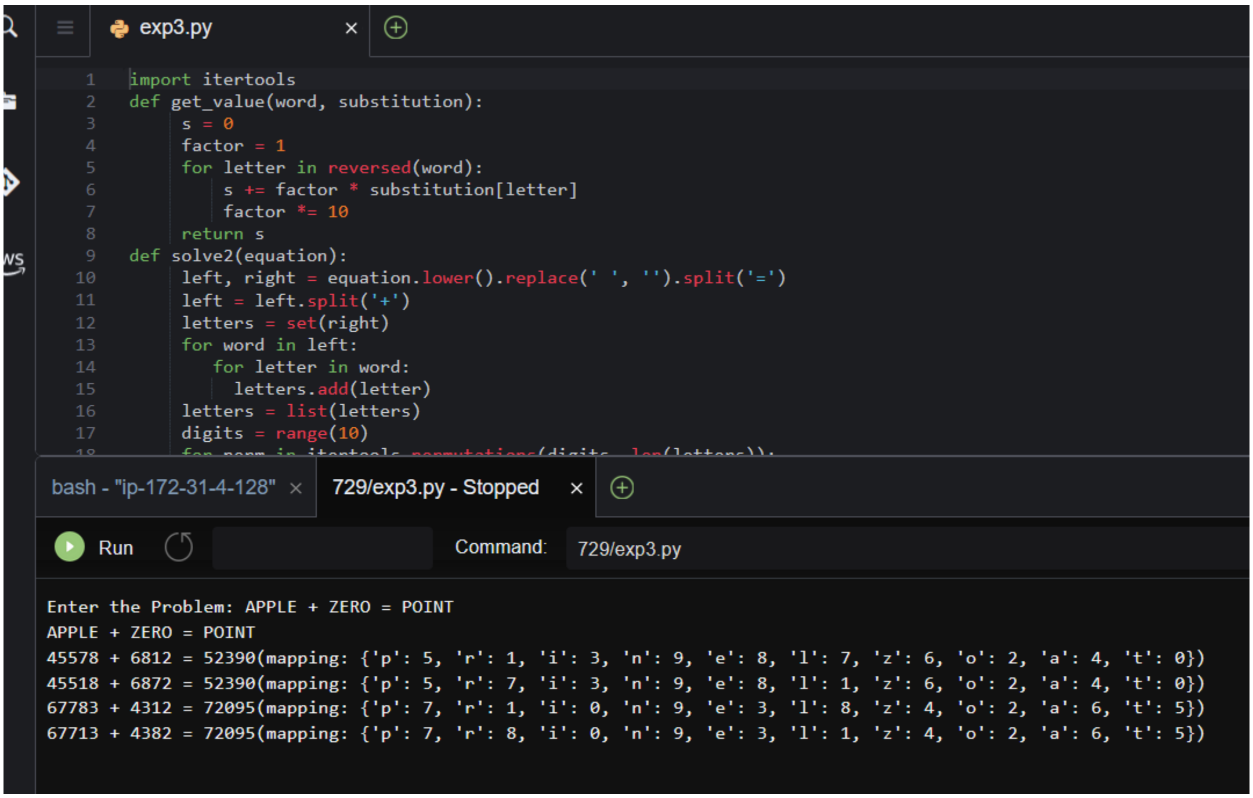
Experiment 3
Constraint Satisfaction Problem

Aim:
Write a generic program to solve the crypt arithmetic puzzle for any given input strings.

Code:
```
import itertools
def get_value(word, substitution):
     s = 0
     factor = 1
     for letter in reversed(word):
         s += factor * substitution[letter]
         factor *= 10
     return s

def solve2(equation):
     left, right = equation.lower().replace(' ',
'').split('=')
     left = left.split('+')
     letters = set(right)
     for word in left:
        for letter in word:
          letters.add(letter)
     letters = list(letters)
     digits = range(10)
     for perm in itertools.permutations(digits, len(letters)):
        sol = dict(zip(letters, perm))
        if sum(get_value(word, sol) for word in left) ==
get_value(right,sol):
```

```
              print(' + '.join(str(get_value(word, sol)) for
word in left) + " = {}(mapping: {})".format(get_value(right,
sol), sol))

a=input("Enter the Problem: ")
print(a)
solve2(a)
```

Input and Output:



Result:

A generic program to solve the crypt arithmetic puzzle for any given input strings has been coded and executed in python successfully.

Aryan Jalla
RA1911003010729

Experiment 4

AIM:

To implement Breadth-First Search and Depth-First Search and find the shortest path for an unweighted graph and compare both algorithms.

a) Breadth-First Search

CODE:

```
from collections import defaultdict

# This class represents a directed graph
# using adjacency list representation
class Graph:

    # Constructor
    def __init__(self):

        # default dictionary to store graph
        self.graph = defaultdict(list)

    # function to add an edge to graph
    def addEdge(self,u,v):
        self.graph[u].append(v)

    # Function to print a BFS of graph
    def BFS(self, s):

        # Mark all the vertices as not visited
        visited = [False] * (max(self.graph) + 1)
```

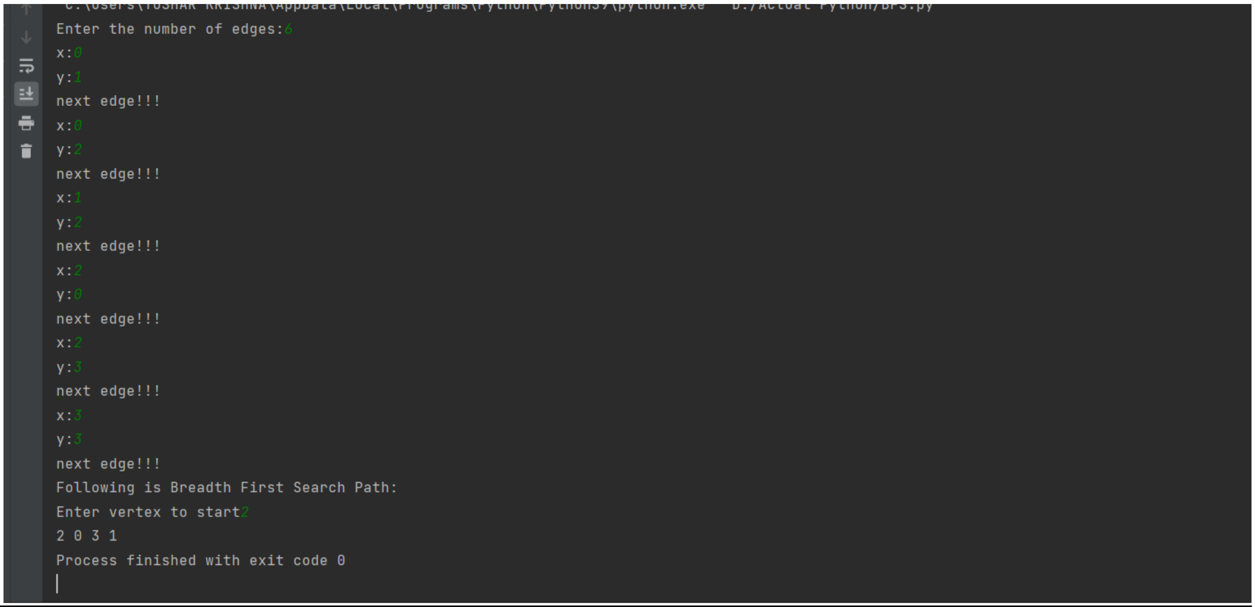
```
        # Create a queue for BFS
        queue = []

        # Mark the source node as
        # visited and enqueue it
        queue.append(s)
        visited[s] = True

        while queue:

            # Dequeue a vertex from
            # queue and print it
            s = queue.pop(0)
            print (s, end = " ")

            # Get all adjacent vertices of the
            # dequeued vertex s. If a adjacent
            # has not been visited, then mark it
            # visited and enqueue it
            for i in self.graph[s]:
                if visited[i] == False:
                    queue.append(i)
                    visited[i] = True

# Create a graph given in
# the above diagram
g = Graph()
n = int(input("Enter the number of edges:"))
for i in range(1,n+1):
    g.addEdge(int(input("x:")),int(input("y:")))
    print("next edge!!!")

print ("Following is Breadth First Search Path:")
g.BFS(int(input("Enter vertex to start")))
```

Code screenshots:

```
from collections import defaultdict

# This class represents a directed graph
# using adjacency list representation
class Graph:

    # Constructor
    def __init__(self):

        # default dictionary to store graph
        self.graph = defaultdict(list)

    # function to add an edge to graph
    def addEdge(self,u,v):
        self.graph[u].append(v)

    # Function to print a BFS of graph
    def BFS(self, s):

        # Mark all the vertices as not visited
        visited = [False] * (max(self.graph) + 1)

        # Create a queue for BFS
        queue = []

        # Mark the source node as
        # visited and enqueue it
        queue.append(s)
        visited[s] = True

        while queue:

            # Dequeue a vertex from
            # queue and print it
            s = queue.pop(0)
```

```
            print (s, end = " ")

            # Get all adjacent vertices of the
            # dequeued vertex s. If a adjacent
            # has not been visited, then mark it
            # visited and enqueue it
            for i in self.graph[s]:
                if visited[i] == False:
                    queue.append(i)
                    visited[i] = True



# Create a graph given in
# the above diagram
g = Graph()
n = int(input("Enter the number of edges:"))
for i in range(1,n+1):
    g.addEdge(int(input("x:")),int(input("y:")))
    print("next edge!!!")


print ("Following is Breadth First Search Path:")
g.BFS(int(input("Enter vertex to start")))
```

Output screenshots:



b) Depth-First Search:

CODE:

```
from collections import defaultdict

# This class represents a directed graph using
# adjacency list representation


class Graph:

    # Constructor
    def __init__(self):

        # default dictionary to store graph
        self.graph = defaultdict(list)
```

```
    def addEdge(self, u, v):
        self.graph[u].append(v)

    def DFSUtil(self, v, visited):

        visited.add(v)
        print(v, end=' ')

        # Recur for all the vertices
        # adjacent to this vertex
        for neighbour in self.graph[v]:
            if neighbour not in visited:
                self.DFSUtil(neighbour, visited)

    # The function to do DFS traversal. It uses
    # recursive DFSUtil()
    def DFS(self, v):

        # Create a set to store visited vertices
        visited = set()

        # Call the recursive helper function
        # to print DFS traversal
        self.DFSUtil(v, visited)

g = Graph()
n = int(input("Enter the number of edges:"))
for i in range(1,n+1):
    g.addEdge(int(input("x:")),int(input("y:")))
    print("next edge!!!")

print("Following is Depth First Search Path: ")
g.DFS(int(input("Enter vertex to start")))
```

Code screenshots:

```
from collections import defaultdict

# This class represents a directed graph using
# adjacency list representation


class Graph:

    # Constructor
    def __init__(self):

        # default dictionary to store graph
        self.graph = defaultdict(list)

    # function to add an edge to graph
    def addEdge(self, u, v):
        self.graph[u].append(v)

    # A function used by DFS
    def DFSUtil(self, v, visited):

        # Mark the current node as visited
        # and print it
        visited.add(v)
        print(v, end=' ')

        # Recur for all the vertices
        # adjacent to this vertex
        for neighbour in self.graph[v]:
            if neighbour not in visited:
                self.DFSUtil(neighbour, visited)

    # The function to do DFS traversal. It uses
    # recursive DFSUtil()
    def DFS(self, v):
```

```
        # Create a set to store visited vertices
        visited = set()

        # Call the recursive helper function
        # to print DFS traversal
        self.DFSUtil(v, visited)

# Driver code


# Create a graph given
# in the above diagram
g = Graph()
n = int(input("Enter the number of edges:"))
for i in range(1,n+1):
    g.addEdge(int(input("x:")),int(input("y:")))
    print("next edge!!!")

print("Following is Depth First Search Path: ")
g.DFS(int(input("Enter vertex to start")))
```

Output screenshots:

```
Enter the number of edges:6
x:0
y:1
next edge!!!
x:0
y:2
next edge!!!
x:1
y:2
next edge!!!
x:2
y:0
next edge!!!
x:2
y:3
next edge!!!
x:3
y:3
next edge!!!
Following is Depth First Search Path:
Enter vertex to start1
1 2 0 3
Process finished with exit code 0
```

Artificial Intelligence

Experiment – 5

Aryan Jalla

RA1911003010729

Best First Search

Aim: To find a path from source to destination using Best first search algorithm

Procedure:

1. Create 2 empty lists: OPEN and CLOSED
2. Start from the initial node (say N) and put it in the 'ordered' OPEN list
3. Repeat the next steps until GOAL node is reached
    1. If OPEN list is empty, then EXIT the loop returning 'False'
    2. Select the first/top node (say N) in the OPEN list and move it to the CLOSED list. Also capture the information of the parent node
    3. If N is a GOAL node, then move the node to the Closed list and exit the loop returning 'True'. The solution can be found by backtracking the path
    4. If N is not the GOAL node, expand node N to generate the 'immediate' next nodes linked to node N and add all those to the OPEN list
    5. Reorder the nodes in the OPEN list in ascending order according to an evaluation function f(n)

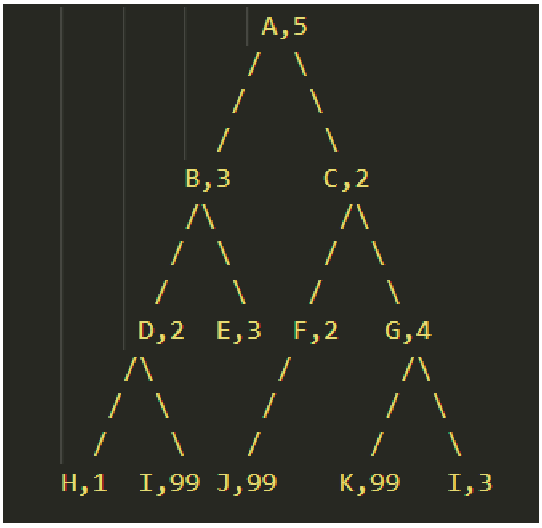
Program:

```
from collections import defaultdict


class Graph:
    def __init__(self, V):
        self.V = V
        self.adj = defaultdict(list)


    def addEdge(self, u, v, h2):
        self.adj[u].append((v, h2))


    def bestFirst(self, s, d, h1):
        parent = {}
```

```
success = False
open = [(s, h1)]
closed = []
parent[s] = None

while open and not success:
    t = open.pop(0)
    print(t[0])

    if t[0] == d:
        success = True
        closed.append(t)
    else:
        closed.append(t)
        for neighbor in self.adj[t[0]]:
            if neighbor not in open and neighbor not in closed:
                open.append(neighbor)
                parent[neighbor[0]] = t[0]
        open.sort(key = lambda  t: t[1])

if success:
    path = []
    n = d
    while parent[n] != None:
        path.append(n)
        n = parent[n]
    path.append(s)
    print("Path found: {}".format(path[::-1]))
else:
    print("No path found!!!")
```

```
v = int(input("Enter the no. vertices: "))
g = Graph(v)


heuristics = dict()
for i in range(v):
    ver_h = input("Enter vertex {} and its heuristic: ". format(i+1)).strip().split()
    heuristics[ver_h[0]] = int(ver_h[1])
    # print(ver_h[0], int(ver_h[1]))


e = int(input("Enter the no. edges: "))
for i in range(e):
    edge = input("Enter the vertices of edge {}: ". format(i+1)).strip().split()
    # print(heuristics[edge[0]], heuristics[edge[1]])
    g.addEdge(edge[0], edge[1], heuristics[edge[1]])


s = input("Enter the source: ")
d = input("Enter the destination: ")
g.bestFirst(s, d, heuristics[s])
```

Input Graph:



Output:

```
Enter the no. vertices: 12
Enter vertex 1 and its heuristic: A 5
Enter vertex 2 and its heuristic: B 3
Enter vertex 3 and its heuristic: C 2
Enter vertex 4 and its heuristic: D 2
Enter vertex 5 and its heuristic: E 3
Enter vertex 6 and its heuristic: F 2
Enter vertex 7 and its heuristic: G 4
Enter vertex 8 and its heuristic: H 1
Enter vertex 9 and its heuristic: I 99
Enter vertex 10 and its heuristic: J 99
Enter vertex 11 and its heuristic: K 99
Enter vertex 12 and its heuristic: I 3
Enter the no. edges: 11
Enter the vertices of edge 1: A B
Enter the vertices of edge 2: A C
Enter the vertices of edge 3: B D
Enter the vertices of edge 4: B E
Enter the vertices of edge 5: C F
Enter the vertices of edge 6: C G
Enter the vertices of edge 7: D H
Enter the vertices of edge 8: D I
Enter the vertices of edge 9: F J
Enter the vertices of edge 10: G K
Enter the vertices of edge 11: G I
Enter the source: A
Enter the destination: H
A
C
F
B
D
H
Path found: ['A', 'B', 'D', 'H']
```

Result:

Hence, best first search algorithm is implemented to find a path from source to destination

A* Search

Aim: To find a path from source to destination using Best first search algorithm

Procedure:

1. Create 2 empty lists: OPEN and CLOSED
2. Start from the initial node (say N) and put it in the 'ordered' OPEN list
3. Repeat the next steps until GOAL node is reached
    1. If OPEN list is empty, then EXIT the loop returning 'False'
    2. Select the first/top node (say N) in the OPEN list and move it to the CLOSED list. Also capture the information of the parent node

3. If N is a GOAL node, then move the node to the Closed list and exit the loop returning ‘True’. The solution can be found by backtracking the path
4. If N is not the GOAL node, expand node N to generate the ‘immediate’ next nodes linked to node N and add all those to the OPEN list
5. Reorder the nodes in the OPEN list in ascending order according to an evaluation function f(n)

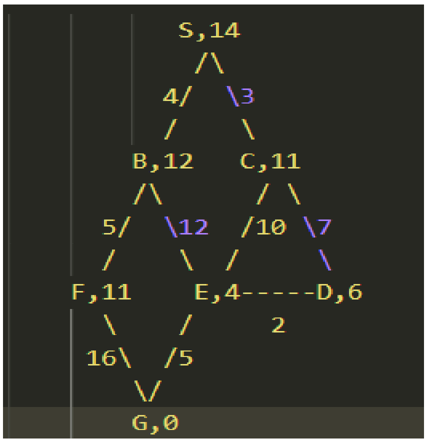
Program:

```
from collections import defaultdict


heuristic = dict()

Graph = defaultdict(list)


def aStar(start, des):
    openSet = [start]
    closedSet = []
    g = {}
    parent = {}
    g[start] = 0
    parent[start] = None

    while openSet:
        n = openSet[0]
        if len(openSet) > 1:
            for v in openSet[1:]:
                if g[v] + heuristic[v] < g[n] + heuristic[n]:
                    n = v
                if n == d or not Graph[n]:
                    pass
        print(n)
            # else:
        for m, w in Graph[n]:
            if m not in openSet and m not in closedSet:
```

```
        openSet.append(m)
        parent[m] = n
        g[m] = g[n] + w

    else:
        if g[m] > g[n] + w:
            g[m] = g[n] + w
            parent[m] = n

            if m in closedSet:
                closedSet.remove(m)
                openSet.append(m)

if n == None:
    print("Path doesn't exist!!!")
    return

if n == des:

    path = []

    while parent[n] != None:
        path.append(n)
        n = parent[n]

    path.append(s)
    print("Path found: {}".format(path[::-1]))
    # print(parent)
    return
openSet.remove(n)
closedSet.append(n)
```

```
v = int(input("Enter the no. vertices: "))
for i in range(v):
    ver_h = input("Enter vertex {} and its heuristic: ". format(i+1)).strip().split()
    heuristic[ver_h[0]] = int(ver_h[1])

e = int(input("Enter the no. edges: "))
for i in range(e):
    edge = input("Enter the vertices of edge {} along with the weight: ". format(i+1)).strip().split()
    Graph[edge[0]].append((edge[1], int(edge[2])))

# print(Graph)

s = input("Enter the source: ")
d = input("Enter the destination: ")
aStar(s, d)
```

Input Graph:



Output:

```
Enter the no. vertices: 7
Enter vertex 1 and its heuristic: S 14
Enter vertex 2 and its heuristic: B 12
Enter vertex 3 and its heuristic: C 11
Enter vertex 4 and its heuristic: F 11
Enter vertex 5 and its heuristic: E 4
Enter vertex 6 and its heuristic: D 6
Enter vertex 7 and its heuristic: G 0
Enter the no. edges: 9
Enter the vertices of edge 1 along with the weight: S B 4
Enter the vertices of edge 2 along with the weight: S C 3
Enter the vertices of edge 3 along with the weight: B F 5
Enter the vertices of edge 4 along with the weight: B E 12
Enter the vertices of edge 5 along with the weight: C E 10
Enter the vertices of edge 6 along with the weight: C D 7
Enter the vertices of edge 7 along with the weight: F G 16
Enter the vertices of edge 8 along with the weight: E G 5
Enter the vertices of edge 9 along with the weight: D E 2
Enter the source: S
Enter the destination: G
S
B
C
B
F
E
D
F
E
F
G
Path found: ['S', 'C', 'D', 'E', 'G']
```

Result:

Hence, A* search algorithm is implemented to find a path from source to destination.

SRM INSTITUTE OF SCIENCE & TECHNOLOGY

DEPARTMENT OF NETWORKING & COMMUNICATIONS

18CSC305J-ARTIFICIAL INTELLIGENCE

SEMESTER – 6

BATCH-2

REGISTRATION NUMBER	**RA1911003010729**
NAME	**Aryan Jalla**

INDEX

Ex No	DATE	Title	Page No	Marks
6		**Implementation of unification and resolution for real world problems.**		

Experiment No: 6

IMPLEMENTATION OF UNIFICATION AND RESOLUTION

PROBLEM STATEMENT : Developing an optimized technique using an appropriate artificial intelligence algorithm to solve the Unification and Resolution.

ALGORITHM :

1. function PL-RESOLUTION (KB, Q) returns true or false inputs: KB,
2. the knowledge base, group of sentences/facts in propositional logic
3. Q, the query, a sentence in propositional logic
4. clauses → the set of clauses in the CNF representation of KB ^ Q new → { }
5. loop do for each Ci, Cj in clauses do
6. resolvents → PL-RESOLVE (Ci, Cj)
7. if resolvents contains the empty clause the return true
8. new → new union resolvents
9. if new is a subset of clauses then return false
10. clauses → clauses union true

OPTIMIZATION TECHNIQUE:

Resolution basically works by using the principle of proof by contradiction. To find the conclusion we should negate the conclusion. Then the resolution rule is applied to the resulting clauses. Each clause that contains complementary literals is resolved to produce a2. new clause, which can be added to the set of facts (if it is not already present). This process continues until one of the two things happen:•There are no new clauses that can be added. An application of the resolution rule derives the empty clauseAn empty clause shows that the negation of the conclusion is a complete contradiction,hence the negation of the conclusion is invalid or false or the assertion is completely valid or true.

1. Convert the given statements in Predicate/Propositional Logic

2. Convert these statements into Conjunctive Normal Form

3. Negate the Conclusion (Proof by Contradiction)

4. Resolve using a Resolution Tree (Unification)

CODE UNIFICATION :

```
def
      get_index_comma(string):
      index_list =  list()
      par_count = 0

      for i in range(len(string)):
          if string[i] == ',' and par_count == 0:
              index_list.append(i)
            elif string[i] ==
          '(': par_count += 1
               elif string[i] ==
                                ')':
              par_count -= 1

      return index_list



def is_variable(expr):
    for i in expr:
          if i == '(' or i == ')':
              return False

    return True def
process_expression(expr):
expr = expr.replace(' ', '')
```

```
index = None for i in
range(len(expr)):
        if expr[i] == '(':
            index = i
            break predicate_symbol =
    expr[:index] expr =
    expr.replace(predicate_symbol, '') expr =
    expr[1:len(expr) - 1]
      arg_list = list()
      indices = get_index_comma(expr)

      if len(indices) == 0:
          arg_list.append(expr)
      else:
          arg_list.append(expr[:indices[0]])
                    for i, j in zip(indices,
                           indices[1:]):
            arg_list.append(expr[i + 1:j])
        arg_list.append(expr[indices[len(indices) - 1] + 1:])

      return predicate_symbol, arg_list



   def get_arg_list(expr):
      _, arg_list = process_expression(expr)

      flag = True
    while flag:
```

```
        flag = False

        for i in arg_list:
         if not is_variable(i):
            flag = True
              _, tmp = process_expression(i)
            for j in tmp:
                if j not in arg_list:
                  arg_list.append(j)
              arg_list.remove(i)

     return arg_list



def check_occurs(var, expr):
   arg_list = get_arg_list(expr)
   if var in arg_list:
        return True

     return False


def unify(expr1, expr2):

    if is_variable(expr1) and is_variable(expr2):
        if expr1 == expr2:
         return 'Null'
      else:
         return False elif is_variable(expr1) and not
   is_variable(expr2): if check_occurs(expr1,
   expr2):
```

```
    return False
else:
```

```
        tmp = str(expr2) + '/' + str(expr1)
      return tmp elif not is_variable(expr1) and
is_variable(expr2):
  if check_occurs(expr2, expr1):
        return False
     else:
        tmp = str(expr1) + '/' + str(expr2)
        return tmp
  else:
  predicate_symbol_1, arg_list_1 = process_expression(expr1)
predicate_symbol_2, arg_list_2 = process_expression(expr2)

     # Step 2 if predicate_symbol_1 !=
   predicate_symbol_2:
      return False # Step 3 elif
   len(arg_list_1) != len(arg_list_2):
      return False
   else:
        # Step 4: Create substitution list
      sub_list = list()

        # Step 5: for i in range(len(arg_list_1)):
      tmp = unify(arg_list_1[i], arg_list_2[i])

           if not tmp:
            return False
         elif tmp == 'Null':
              pass
  else:
```

```
            if type(tmp) == list: for
                j in tmp:
                sub_list.append(j)

              else:
                  sub_list.append(tmp)


        # Step 6
        return sub_list

if      name___== ' main-':
       # f1 = 'Q(a, g(x, a), f(y))'
     # f2 = 'Q(a, g(f(b), a), x)'
     f1 = input('f1 : ') f2 =
     input('f2 : ')


       result = unify(f1, f2)
       if not result:
           print('The process of Unification failed!')
       else:
           print('The process of Unification successful!')
           print(result)
```

OUTPUT UNIFICATION :

```
f1 : 'Q(a, g(x, a), f(y))'
f2 : 'Q(a, g(f(b), a), x)'
The process of Unification successful!
['f(b)/x', 'f(y)/x']
```

SRM INSTITUTE OF SCIENCE & TECHNOLOGY

DEPARTMENT OF NETWORKING &

COMMUNICATIONS **18CSC305J-ARTIFICIAL**

INTELLIGENCE

SEMESTER – 6 BATCH-2

REGISTRATION NUMBER	**RA1911003010729**
NAME	**Aryan Jalla**

INDEX

Ex No	DATE	Title	Page No	Marks
7	21/03/22	**Implementation of uncertain methods for an application (Fuzzy logic/ Dempster Shafer Theory)**		

Experiment No : 7

IMPLEMENTATION OF UNCERTAIN METHODS OF AN APPLICATION

Problem Statement:

To implement Fuzzy logic using matplotlib in python and find the graph of temperature, humidity and speed in different conditions.

Algorithm:

1. Locate the input, output, and state variables of the plane under consideration.
2. Split the complete universe of discourse spanned by each variable into a number of fuzzy subsets, assigning each with a linguistic label. The subsets include all the elements in the universe.
3. Obtain the membership function for each fuzzy subset.
4. Assign the fuzzy relationships between the inputs or states of fuzzy subsets on one side and the output of fuzzy subsets on the other side, thereby forming the rule base.
5. Choose appropriate scaling factors for the input and output variables for normalizing the variables between [0, 1] and [-1, I] interval.
6. Carry out the fuzzification process.
7. Identify the output contributed from each rule using fuzzy approximate reasoning.
8. Combine the fuzzy outputs obtained from each rule.
9. Finally, apply defuzzification to form a crisp output.

Optimization Technique:

1. Decomposing the large-scale system into a collection of various subsystems.
2. Varying the plant dynamics slowly and linearizing the nonlinear plane dynamics about a set of operating points.
3. Organizing a set of state variables, control variables, or output features for the system under consideration.
4. Designing simple P, PD, PID controllers for the subsystems. Optimal controllers can also be designed.

Uncertainty In this problem : Fuzzy Logic - Temperature, Humidity and Speed.

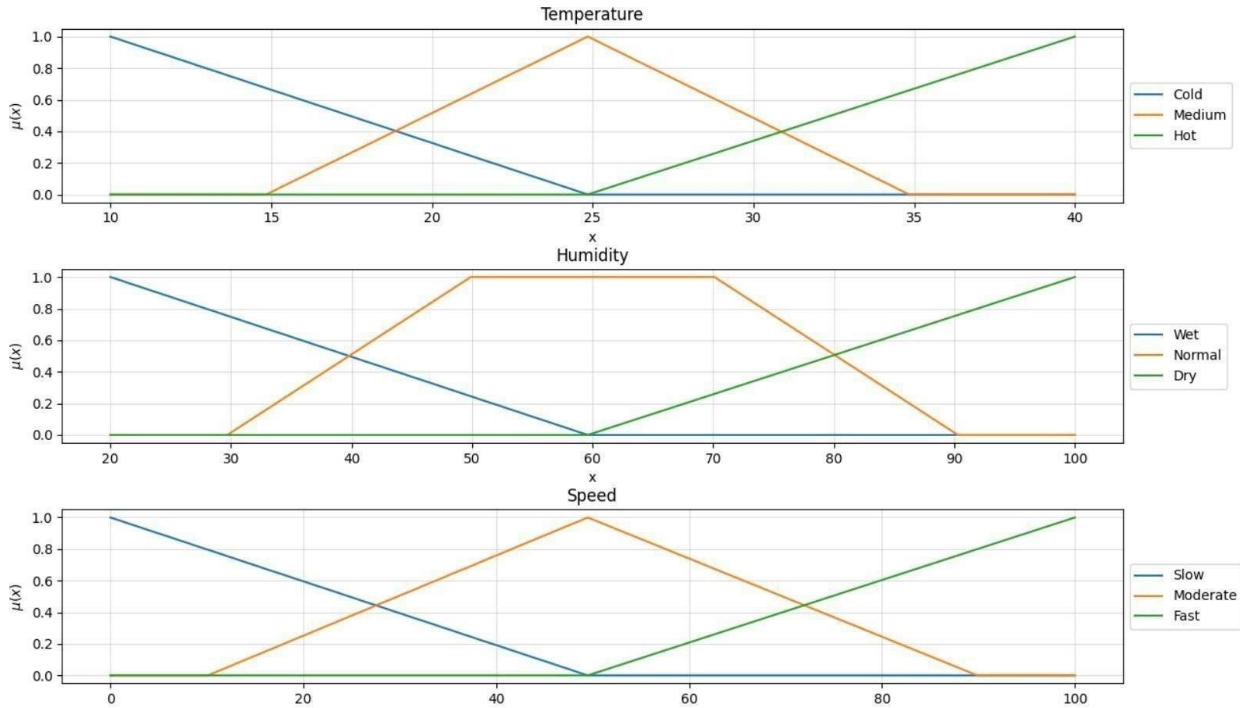
CODE :

```
from fuzzy_system.fuzzy_variable_output import
FuzzyOutputVariable from fuzzy_system.fuzzy_variable_input import
FuzzyInputVariable # from fuzzy_system.fuzzy_variable import
FuzzyVariable from fuzzy_system.fuzzy_system import FuzzySystem temp
= FuzzyInputVariable('Temperature', 10, 40, 100)
temp.add_triangular('Cold', 10, 10, 25) temp.add_triangular('Medium',
15, 25, 35) temp.add_triangular('Hot', 25, 40, 40) humidity
= FuzzyInputVariable('Humidity', 20, 100, 100)
humidity.add_triangular('Wet', 20, 20, 60) humidity.add_trapezoidal('Normal',
30, 50, 70, 90) humidity.add_triangular('Dry', 60, 100, 100) motor_speed
= FuzzyOutputVariable('Speed', 0, 100, 100) motor_speed.add_triangular('Slow',
0, 0, 50) motor_speed.add_triangular('Moderate', 10, 50, 90)
motor_speed.add_triangular('Fast', 50, 100, 100)

system = FuzzySystem() system.add_input_variable(temp)
system.add_input_variable(humidity)
system.add_output_variable(motor_speed)

system.add_rule(
                { 'Temperature':'Cold',
                        'Humidity':'Wet' }, {
                'Speed':'Slow'})

system.add_rule(
                { 'Temperature':'Cold',

                        ''Humidity':'Normal' },
                { 'Speed':'Slow'})

system.add_rule(
```

```
                { 'Temperature':'Medium',
                        'Humidity':'Wet' },
                { 'Speed':'Slow'})

system.add_rule(
                { 'Temperature':'Medium',
                        'Humidity':'Normal' },
                { 'Speed':'Moderate'})

system.add_rule(
                { 'Temperature':'Cold',
                        'Humidity':'Dry' },
                { 'Speed':'Moderate'})

system.add_rule(
                { 'Temperature':'Hot',
                        'Humidity':'Wet' },
                { 'Speed':'Moderate'})

system.add_rule(
                { 'Temperature':'Hot',
                        'Humidity':'Normal' },
                { 'Speed':'Fast'})

system.add_rule(
                { 'Temperature':'Hot',
                        'Humidity':'Dry' },
                { 'Speed':'Fast'}) system.add_rule(

                { 'Temperature':'Medium',
                        'Humidity':'Dry' },
                { 'Speed':'Fast'})
```

```
output = system.evaluate_output({
                                    'Temperature':18,
                                    'Humidity':60 })

print(output) system.plot_system()
```

OUTPUT :



Result: We have successfully implemented fuzzy uncertainty problem using matplotlib and output is received.

EXPERIMENT 8: Implementation of learning algorithms for an application

8A: Linear regression

Aim: To write a program to implement linear regression on student score dataset

Algorithm:

The main function to calculate values of coefficients

1. Initialize the parameters.
2. Predict the value of a dependent variable by given an independent variable.
3. Calculate the error in prediction for all data points.
4. Calculate partial derivatives w.r.t a0 and a1.
5. Calculate the cost for each number and add them.
6. Update the values of a0 and a1.

Dataset:

In [46]:

```
df.head()
```

Out[46]:

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
0	female	group B	bachelor's degree	standard	none	72	72	74
1	female	group C	some college	standard	completed	69	90	88
2	female	group B	master's degree	standard	none	90	95	93
3	male	group A	associate's degree	free/reduced	none	47	57	44
4	male	group C	some college	standard	none	76	78	75

Code:

```
import pandas as pd
pd.set_option('display.max_columns', None)
import numpy as np

df = pd.read_csv('../input/students-performance-in-exams/StudentsPerformance.csv')
df.shape
df.columns
df.info()
df.describe()
df.head()
df.isnull().sum()
df.rename(columns = {'race/ethnicity':'race'}, inplace = True)
df.rename(columns = {'parental level of education':'parent_education'}, inplace = True)
df.rename(columns = {'test preparation course':'prep_course'}, inplace = True)
df.rename(columns = {'math score':'math_score'}, inplace = True)
df.rename(columns = {'reading score':'reading_score'}, inplace = True)
df.rename(columns = {'writing score':'writing_score'}, inplace = True)
df['total_score'] = df['math_score'] + df['reading_score'] + df['writing_score']
df.columns
```

```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize = (6,8))
sns.set(style = 'darkgrid', font = 'sans-serif', font_scale = 1.25, palette = 'Set2')
ax = sns.countplot(
          x = 'gender',
          data = df,
          edgecolor = 'black')
ax.set_title('Distribution of Student Genders', fontsize = 15)
ax.set(xlabel = 'Gender', ylabel = 'Frequency')

plt.figure(figsize = (9,6))
sns.set(style = 'darkgrid', font = 'sans-serif', font_scale = 1.25, palette = 'deep')
ax = sns.countplot(
          x = 'race',
          data = df,
          edgecolor = 'black')
ax.set_title('Distribution of Student Race/Ethnicity', fontsize = 15)
ax.set(xlabel = 'Race/Ethnicity', ylabel = 'Frequency')

plt.figure(figsize = (13,6))
sns.set(style = 'darkgrid', font = 'sans-serif', font_scale = 1.25, palette = 'deep')
ax = sns.countplot(
          x = 'parent_education',
          data = df,
          edgecolor = 'black')
ax.set_title('Distribution of Parent Education Level', fontsize = 20)
ax.set(xlabel = 'Parental Education Level', ylabel = 'Frequency')

plt.figure(figsize = (6,8))
sns.set(style = 'darkgrid', font = 'sans-serif', font_scale = 1.25, palette = 'deep')
ax = sns.countplot(
          x = 'lunch',
          data = df,
          edgecolor = 'black')
ax.set_title('Distribution of Lunch Options', fontsize = 15)
ax.set(xlabel = 'Lunch Option', ylabel = 'Frequency')

plt.figure(figsize = (6,8))
sns.set(style = 'darkgrid', font = 'sans-serif', font_scale = 1.25, palette = 'deep')
ax = sns.countplot(
          x = 'prep_course',
          data = df,
          edgecolor = 'black')
ax.set_title('Distribution of Prep Course', fontsize = 15)
ax.set(xlabel = 'Prep Course', ylabel = 'Frequency')

sns.set(style = 'darkgrid', font = 'sans-serif', font_scale = 1.25)
plt.figure(figsize = (10,6))
plt.hist(df['math_score'], bins = 20, color = 'cornflowerblue')
plt.xlabel('Math Score', fontsize = 13)
plt.ylabel('Frequency', fontsize = 13)
plt.title('Distribution of Math Scores', fontsize = 13)
plt.show()

sns.set(style = 'darkgrid', font = 'sans-serif', font_scale = 1.25)
plt.figure(figsize = (10,6))
plt.hist(df['reading_score'], bins = 20, color = 'lightcoral')
plt.xlabel('Reading Score', fontsize = 13)
plt.ylabel('Frequency', fontsize = 13)
plt.title('Distribution of Reading Scores', fontsize = 13)
plt.show()

sns.set(style = 'darkgrid', font = 'sans-serif', font_scale = 1.25)
plt.figure(figsize = (10,6))
plt.hist(df['writing_score'], bins = 20, color = 'goldenrod')
plt.xlabel('Writing Score', fontsize = 13)
plt.ylabel('Frequency', fontsize = 13)
plt.title('Distribution of Writing Score', fontsize = 13)
plt.show()

sns.set(style = 'darkgrid', font = 'sans-serif', font_scale = 1.25)
plt.figure(figsize = (10,6))
plt.hist(df['total_score'], bins = 20, color = 'darkorchid')
plt.xlabel('Total Score', fontsize = 13)
plt.ylabel('Frequency', fontsize = 13)
plt.title('Distribution of Total Score', fontsize = 13)
plt.show()

df1 = df[['gender', 'race', 'parent_education', 'prep_course', 'lunch']]
X = pd.get_dummies(df1, columns = ['gender', 'race', 'parent_education', 'prep_course', 'lunch'], dtype = int)
y = df['total_score']
```

```
from sklearn.linear_model import LinearRegression
import statsmodels.api as sm

X_constant = sm.add_constant(X)
lin_reg = sm.OLS(y, X_constant).fit()
lin_reg.summary()

import statsmodels.stats.api as sms
sns.set_style('darkgrid')
sns.mpl.rcParams['figure.figsize'] = (15.0, 9.0)

def linearity_test(model, y):
          fitted_vals = model.predict()
          resids = model.resid

          fig, ax = plt.subplots(1,2)

          sns.regplot(x = fitted_vals, y = y, lowess = True, ax = ax[0], line_kws = {'color': 'red'})
          ax[0].set_title('Observed vs. Predicted Values', fontsize = 16)
          ax[0].set(xlabel = 'Predicted', ylabel = 'Observed')

          sns.regplot(x = fitted_vals, y = resids, lowess = True, ax = ax[1], line_kws = {'color' : 'red'})
          ax[1].set_title('Residuals vs. Predicted Values', fontsize = 16)
          ax[1].set(xlabel = 'Predicted', ylabel = 'Residuals')

linearity_test(lin_reg, y)
lin_reg.resid.mean()
```

Output:

```
In [65]:  lin_reg.resid.mean()

Out[65]:  1.2644818525586742e-13
```

Good, very small.

Result: Linear regression was trained and tested on students' scores dataset.

8B: Support Vector Machine (SVM)

Aim: To write a program to implement support vector machine on breast cancer detection dataset

Algorithm:

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called support vectors, and hence the algorithm is termed as Support Vector Machine.

Dataset:

In [3]:

```
bc = pd.read_csv('../input/data.csv')
bc.head(1)
```

Out[3]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	co
0	842302	M	17.99	10.38	122.8	1001.0	0.1184	0.2776	0.3

1 rows × 33 columns

Code:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn import preprocessing
import pandas as pd
import random
import itertools
import seaborn as sns

sns.set(style = 'darkgrid')

def plot_confusion_matrix(cm, classes,
                              normalize=False,
                              title='Confusion matrix',
                              cmap=plt.cm.Blues):

          plt.imshow(cm, interpolation='nearest', cmap=cmap)
          plt.title(title)
          plt.colorbar()
          tick_marks = np.arange(len(classes))
          plt.xticks(tick_marks, classes, rotation=45)
          plt.yticks(tick_marks, classes)

          if normalize:
          cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
          print("Normalized confusion matrix")
          else:
          print('Confusion matrix, without normalization')

          print(cm)

          thresh = cm.max() / 2.
          for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
          plt.text(j, i, cm[i, j],
                    horizontalalignment="center",
                    color="white" if cm[i, j] > thresh else "black")

          plt.tight_layout()
          plt.ylabel('True label')
          plt.xlabel('Predicted label')

bc = pd.read_csv('../input/data.csv')
bc.head(1)

bcs = pd.DataFrame(preprocessing.scale(bc.iloc[:,2:32]))
bcs.columns = list(bc.iloc[:,2:32].columns)
bcs['diagnosis'] = bc['diagnosis']
```

```
from pandas.plotting import scatter_matrix
p = sns.PairGrid(bcs.iloc[:,20:32], hue = 'diagnosis', palette = 'Reds')
p.map_upper(plt.scatter, s = 20, edgecolor = 'w')
p.map_diag(plt.hist)
p.map_lower(sns.kdeplot, cmap = 'GnBu_d')
p.add_legend()

p.figsize = (30,30)

mbc = pd.melt(bcs, "diagnosis", var_name="measurement")
fig, ax = plt.subplots(figsize=(10,5))
p = sns.violinplot(ax = ax, x="measurement", y="value", hue="diagnosis", split = True, data=mbc, inner = 'quartile', palette = 'Set2');
p.set_xticklabels(rotation = 90, labels = list(bcs.columns));
sns.swarmplot(x = 'diagnosis', y = 'concave points_worst',palette = 'Set2', data = bcs);

sns.jointplot(x = bc['concave points_worst'], y = bc['area_mean'], stat_func=None, color="#4CB391", edgecolor = 'w', size = 6);

X = bcs.iloc[:,0:30]

y = bcs['diagnosis']
class_names = list(y.unique())

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)

svc = SVC(kernel = 'linear',C=.1, gamma=10, probability = True)
svc.fit(X,y)
y_pred = svc.fit(X_train, y_train).predict(X_test)
t = pd.DataFrame(svc.predict_proba(X_test))
svc.score(X_train,y_train), svc.score(X_test, y_test)

mtrx = confusion_matrix(y_test,y_pred)
np.set_printoptions(precision = 2)

plt.figure()
plot_confusion_matrix(mtrx,classes=class_names,title='Confusion matrix, without normalization')

plt.figure()
plot_confusion_matrix(mtrx, classes=class_names, normalize = True, title='Normalized confusion matrix')

plt.show()
```

Output:

```
[20]:
svc = SVC(kernel = 'linear',C=.1, gamma=10, probability = True)
svc.fit(X,y)
y_pred = svc.fit(X_train, y_train).predict(X_test)
t = pd.DataFrame(svc.predict_proba(X_test))
svc.score(X_train,y_train), svc.score(X_test, y_test)

[20… (0.984251968503937, 0.9787234042553191)
```

Result: Support Vector Machine was trained and tested on breast cancer detection.

8C: K-means clustering

Aim: To write a program to implement k-means clustering on customer demographic dataset

Algorithm:

The working of the K-Means algorithm is explained in the below steps:

Step-1: Select the number K to decide the number of clusters.

Step-2: Select random K points or centroids. (It can be other from the input dataset).

Step-3: Assign each data point to their closest centroid, which will form the predefined K clusters.

Step-4: Calculate the variance and place a new centroid of each cluster.

Step-5: Repeat the third steps, which means reassign each datapoint to the new closest centroid of each cluster.

Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.

Step-7: The model is ready.

Dataset:

In [3]:

```
df.head()
```

Out[3]:

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

Code:

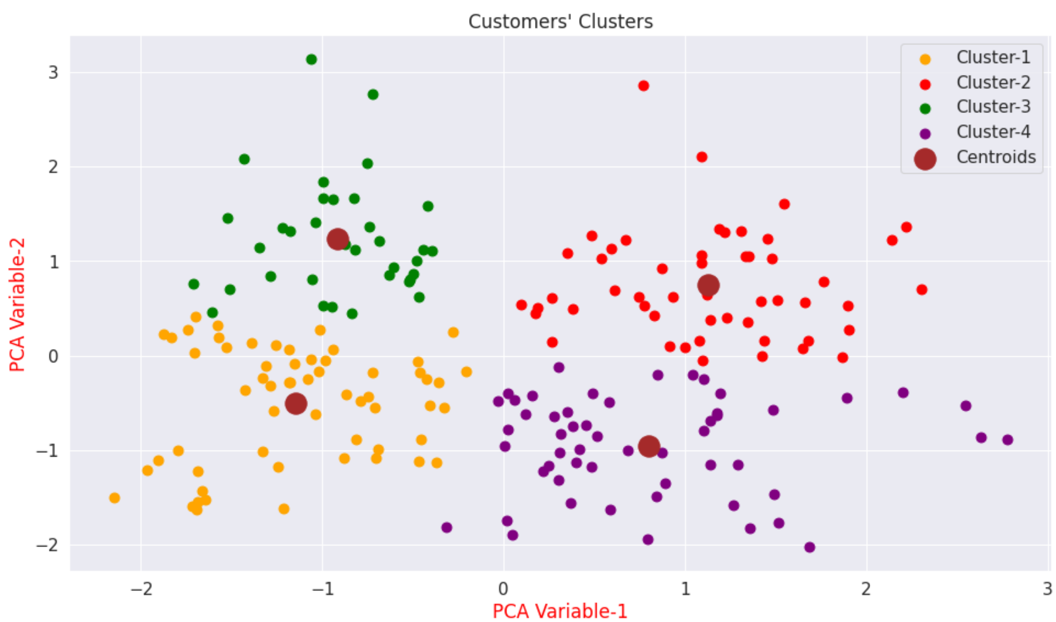
```
df.head()

df.drop('CustomerID', axis=1, inplace = True)
df.head()

df.shape
df.info()

df.isnull().sum()

df.describe()
cor = df.corr()
sns.set(font_scale=1.4)
plt.figure(figsize=(9,8))
sns.heatmap(cor, annot=True, cmap='plasma')
plt.tight_layout()
plt.show()

plt.figure(figsize=(16,12),facecolor='#9DF08E')

# Spending Score
```

```
plt.subplot(3,3,1)
plt.title('Spending Score\n', color='#FF000B')
sns.distplot(df['Spending Score (1-100)'], color='orange')

# Age
plt.subplot(3,3,2)
plt.title('Age\n', color='#FF000B')
sns.distplot(df['Age'], color='#577AFF')

# Annual Income
plt.subplot(3,3,3)
plt.title('Annual Income\n', color='#FF000B')
sns.distplot(df['Annual Income (k$)'], color='black')

plt.suptitle(' Distribution Plots\n', color='#0000C1', size = 30)
plt.tight_layout()

# Before-After Label Encoder

from sklearn.preprocessing import LabelEncoder

print('\033[0;32m' + 'Before Label Encoder\n' + '\033[0m' + '\033[0;32m', df['Gender'])

le = LabelEncoder()
df['Gender'] = le.fit_transform(df.iloc[:,0])

print('\033[0;31m' + '\n\nAfter Label Encoder\n' + '\033[0m' + '\033[0;31m', df['Gender'])
spending_score_male = 0
spending_score_female = 0

for i in range(len(df)):
          if df['Gender'][i] == 1:
          spending_score_male = spending_score_male + df['Spending Score (1-100)'][i]
          if df['Gender'][i] == 0:
          spending_score_female = spending_score_female + df['Spending Score (1-100)'][i]


print('\033[1m' + '\033[93m' + f'Males Spending Score  : {spending_score_male}')
print('\033[1m' + '\033[93m' + f'Females Spending Score: {spending_score_female}')

plt.figure(figsize=(16,16),facecolor='#54C6C0')
plt.subplot(3,3,1)
plots = sns.barplot(x=['Female','Male'], y=df['Gender'].value_counts(), data=df)

for bar in plots.patches:
          plots.annotate(format(bar.get_height(), '.0f'),
                    (bar.get_x() + bar.get_width() / 2,
                    bar.get_height()), ha='center', va='center',
                    size=13, xytext=(0, 8),
                    textcoords='offset points',color='red')

plt.xlabel("Gender", size=14)
plt.ylabel("Number", size=14)
plt.yticks(np.arange(0,116,10),size='14')
plt.grid(False)
plt.title("Number of Genders\n", color="red", size='22')

# Gender & Total Spending Score

list_genders_spending_score = [int(spending_score_female),int(spending_score_male)]
series_genders_spending_score = pd.Series(data = list_genders_spending_score)


plt.subplot(3,3,2)
plots = sns.barplot(x=['Female','Male'], y=series_genders_spending_score, palette=['yellow','purple'])

for bar in plots.patches:
          plots.annotate(format(bar.get_height(), '.0f'),
                    (bar.get_x() + bar.get_width() / 2,
                    bar.get_height()), ha='center', va='center',
                    size=13, xytext=(0, 8),
                    textcoords='offset points',color='red')

plt.xlabel("Gender", size=14)
plt.ylabel("Total Spending Score", size=14)
plt.yticks(np.arange(0,6001,1000),size='14')
plt.grid(False)
plt.title("Gender & Total Spending Score\n", color="red", size='22')


# Gender & Mean Spending Score

list_genders_spending_score_mean =
[int(spending_score_female/df['Gender'].value_counts()[0]),int(spending_score_male/df['Gender'].value_counts()[1])]
series_genders_spending_score_mean = pd.Series(data = list_genders_spending_score_mean)
```

```
plt.subplot(3,3,3)
plots = sns.barplot(x=['Female','Male'], y=series_genders_spending_score_mean, palette='hsv')

for bar in plots.patches:
          plots.annotate(format(bar.get_height(), '.0f'),
                    (bar.get_x() + bar.get_width() / 2,
                    bar.get_height()), ha='center', va='center',
                    size=13, xytext=(0, 8),
                    textcoords='offset points',color='red')

plt.xlabel("Gender", size=14)
plt.ylabel("Mean Spending Score", size=14)
plt.yticks(np.arange(0,71,10),size='14')
plt.grid(False)
plt.title("Gender & Mean Spending Score\n", color="red", size='22')
plt.tight_layout()
plt.show()
plt.figure(figsize=(12,8))
sns.scatterplot(x = df['Age'], y = df['Spending Score (1-100)'])
plt.title('Age - Spending Score', size = 23, color='red')
plt.figure(figsize=(12,8))
sns.scatterplot(x = df['Annual Income (k$)'], y = df['Spending Score (1-100)'], palette = "red")
plt.title('Annual Income - Spending Score', size = 23, color='red')
x = df.iloc[:,0:].values
print("\033[1;31m"  + f'X data before PCA:\n {x[0:5]}')
# standardization before PCA
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X = sc.fit_transform(x)

# PCA
from sklearn.decomposition import PCA
pca = PCA(n_components = 2)
X_2D = pca.fit_transform(X)
print("\033[0;32m" + f'\nX data after PCA:\n {X_2D[0:5,:]}')

# finding optimum number of clusters
from sklearn.cluster import KMeans
wcss_list = []

for i in range(1,11):
          kmeans_test = KMeans(n_clusters = i, init ='k-means++', random_state=88)
          kmeans_test.fit(X_2D)
          wcss_list.append(kmeans_test.inertia_)

plt.figure(figsize=(9,6))
plt.plot(range(1, 11), wcss_list)
plt.title('The Elbow Method', color='red',fontsize='23')
plt.xlabel('Number of clusters')
plt.xticks(np.arange(1,11))
plt.ylabel('WCSS')
plt.show()
kmeans = KMeans(n_clusters = 4, init ='k-means++', random_state=88)
y_kmeans = kmeans.fit_predict(X_2D)

plt.figure(1 , figsize = (16 ,9))
plt.scatter(X_2D[y_kmeans == 0, 0], X_2D[y_kmeans == 0, 1], s = 80, c = 'orange', label = 'Cluster-1')
plt.scatter(X_2D[y_kmeans == 1, 0], X_2D[y_kmeans == 1, 1], s = 80, c = 'red', label = 'Cluster-2')
plt.scatter(X_2D[y_kmeans == 2, 0], X_2D[y_kmeans == 2, 1], s = 80, c = 'green', label = 'Cluster-3')
plt.scatter(X_2D[y_kmeans == 3, 0], X_2D[y_kmeans == 3, 1], s = 80, c = 'purple', label = 'Cluster-4')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 375, c = 'brown', label = 'Centroids')
plt.title("Customers' Clusters")
plt.xlabel('PCA Variable-1', color='red')
plt.ylabel('PCA Variable-2', color='red')
plt.legend()
plt.show()
x = df[['Age','Annual Income (k$)','Spending Score (1-100)']].values
x_df = df[['Age','Annual Income (k$)','Spending Score (1-100)']] # this line for 3d scatter plot

wcss_list = []

for i in range(1,11):
          kmeans_test = KMeans(n_clusters = i, init ='k-means++', random_state=88)
          kmeans_test.fit(x)
          wcss_list.append(kmeans_test.inertia_)

plt.figure(figsize=(9,6))
plt.plot(range(1, 11), wcss_list)
plt.title('The Elbow Method', color='red',fontsize='23')
plt.xlabel('Number of clusters')
plt.xticks(np.arange(1,11))
plt.ylabel('WCSS')
plt.show()
```

```
# KMeans
kmeans = KMeans(n_clusters = 2, init ='k-means++', random_state=88)
y_kmeans = kmeans.fit_predict(x)
# clusters visualization
plt.figure(1 , figsize = (16 ,9))
plt.scatter(x[y_kmeans == 0, 0], x[y_kmeans == 0, 1], s = 80, c = '#13DB8C', label = 'Cluster-1')
plt.scatter(x[y_kmeans == 1, 0], x[y_kmeans == 1, 1], s = 80, c = '#72BAFF', label = 'Cluster-2')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 350, c = 'brown', label = 'Centroids')
plt.title("Customers' Clusters")
plt.xlabel('Age', color='red')
plt.ylabel('Annual Income (k$)', color='red')
plt.legend()
plt.show()

x = df[['Annual Income (k$)','Spending Score (1-100)']].values
# finding optimum number of clusters
wcss_list = []
for i in range(1,11):
          kmeans_test = KMeans(n_clusters = i, init ='k-means++', random_state=88)
          kmeans_test.fit(x)
          wcss_list.append(kmeans_test.inertia_)

plt.figure(figsize=(9,6))
plt.plot(range(1, 11), wcss_list)
plt.title('The Elbow Method', color='red',fontsize='23')
plt.xlabel('Number of clusters')
plt.xticks(np.arange(1,11))
plt.ylabel('WCSS')
plt.show()
kmeans = KMeans(n_clusters = 5, init ='k-means++', random_state=88)
y_kmeans = kmeans.fit_predict(x)

# clusters visualization
plt.figure(1 , figsize = (16 ,9))
plt.scatter(x[y_kmeans == 0, 0], x[y_kmeans == 0, 1], s = 80, c = 'orange', label = 'Cluster-1')
plt.scatter(x[y_kmeans == 1, 0], x[y_kmeans == 1, 1], s = 80, c = 'red', label = 'Cluster-2')
plt.scatter(x[y_kmeans == 2, 0], x[y_kmeans == 2, 1], s = 80, c = 'purple', label = 'Cluster-3')
plt.scatter(x[y_kmeans == 3, 0], x[y_kmeans == 3, 1], s = 80, c = 'lime', label = 'Cluster-4')
plt.scatter(x[y_kmeans == 4, 0], x[y_kmeans == 4, 1], s = 80, c = 'blue', label = 'Cluster-5')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 375, c = 'brown', label = 'Centroids')
plt.title("Customers' Clusters")
plt.xlabel('Annual Income (k$)', color='red')
plt.ylabel('Spending Score', color='red')
plt.legend()
plt.show()
```

Output:



Result: K-means clustering was trained and tested on customer demographic data

8D: Apriori

Aim: To write a program to implement apriori on sales dataset

Algorithm:

Step 1. Computing the support for each individual item
Step 2. Deciding on the support threshold
Step 3. Selecting the frequent items
Step 4. Finding the support of the frequent itemsets
Step 5. Repeat for larger sets
Step 6. Generate Association Rules and compute confidence
Step 7. Compute lift

Dataset:

[2]:
```
# Loading the Data
data = pd.read_csv('../input/online-retail/Online_Retail.csv')
data.head()
```

[2]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom

Code:

```
import numpy as np
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules

# Loading the Data
data = pd.read_csv('../input/online-retail/Online_Retail.csv')
data.head()

# Exploring the columns of the data
data.columns

# Exploring the different regions of transactions
data.Country.unique()

# Stripping extra spaces in the description
data['Description'] = data['Description'].str.strip()

# Dropping the rows without any invoice number
data.dropna(axis = 0, subset =['InvoiceNo'], inplace = True)
data['InvoiceNo'] = data['InvoiceNo'].astype('str')

# Dropping all transactions which were done on credit
data = data[~data['InvoiceNo'].str.contains('C')]

# Transactions done in France
basket_fra = (data[data['Country'] =="France"]
           .groupby(['InvoiceNo', 'Description'])['Quantity']
           .sum().unstack().reset_index().fillna(0)
           .set_index('InvoiceNo'))

# Transactions done in the Brazil
basket_bra = (data[data['Country'] =="Brazil"]
           .groupby(['InvoiceNo', 'Description'])['Quantity']
           .sum().unstack().reset_index().fillna(0)
           .set_index('InvoiceNo'))

# Transactions done in Portugal
basket_por = (data[data['Country'] =="Portugal"]
           .groupby(['InvoiceNo', 'Description'])['Quantity']
           .sum().unstack().reset_index().fillna(0)
           .set_index('InvoiceNo'))

# Transactions done in Portugal
basket_swe = (data[data['Country'] =="Sweden"]
           .groupby(['InvoiceNo', 'Description'])['Quantity']
           .sum().unstack().reset_index().fillna(0)
           .set_index('InvoiceNo'))

# Defining the hot encoding function to make the data suitable
# for the concerned libraries
def hot_encode(x):
    if(x<= 0):
           return 0
    if(x>= 1):
           return 1

# Encoding the datasets
basket_encoded = basket_fra.applymap(hot_encode)
basket_fra = basket_encoded

basket_encoded = basket_bra.applymap(hot_encode)
```

```
basket_bra = basket_encoded

basket_encoded = basket_por.applymap(hot_encode)
basket_por = basket_encoded

basket_encoded = basket_swe.applymap(hot_encode)
basket_swe = basket_encoded



#France
# Building the model
frq_items = apriori(basket_fra, min_support = 0.05, use_colnames = True)
# Collecting the inferred rules in a dataframe
rules = association_rules(frq_items, metric ="lift", min_threshold = 1)
rules = rules.sort_values(['confidence', 'lift'], ascending =[False, False])
print(rules.head())

#Brazil
frq_items = apriori(basket_bra, min_support = 0.05, use_colnames = True)
rules = association_rules(frq_items, metric ="lift", min_threshold = 1)
rules = rules.sort_values(['confidence', 'lift'], ascending =[False, False])
print(rules.head())

#Portugal
frq_items = apriori(basket_por, min_support = 0.05, use_colnames = True)
rules = association_rules(frq_items, metric ="lift", min_threshold = 1)
rules = rules.sort_values(['confidence', 'lift'], ascending =[False, False])
print(rules.head())

#Sweden
frq_items = apriori(basket_swe, min_support = 0.05, use_colnames = True)
rules = association_rules(frq_items, metric ="lift", min_threshold = 1)
rules = rules.sort_values(['confidence', 'lift'], ascending =[False, False])
print(rules.head())
```

Output:

Rules for items of Portugal

```
#Portugal
frq_items = apriori(basket_por, min_support = 0.05, use_colnames = True)
rules = association_rules(frq_items, metric ="lift", min_threshold = 1)
rules = rules.sort_values(['confidence', 'lift'], ascending =[False, False])
print(rules.head())
```

```
                          antecedents                          consequents  \
1170    (SET 12 COLOUR PENCILS SPACEBOY)  (SET 12 COLOUR PENCILS DOLLY GIRL)
1171  (SET 12 COLOUR PENCILS DOLLY GIRL)    (SET 12 COLOUR PENCILS SPACEBOY)
1172  (SET 12 COLOUR PENCILS DOLLY GIRL)  (SET OF 4 KNICK KNACK TINS LONDON)
1173  (SET OF 4 KNICK KNACK TINS LONDON)  (SET 12 COLOUR PENCILS DOLLY GIRL)
1174 (SET OF 4 KNICK KNACK TINS POPPIES)  (SET 12 COLOUR PENCILS DOLLY GIRL)

     antecedent support  consequent support   support  confidence       lift  \
1170           0.051724            0.051724  0.051724         1.0  19.333333
1171           0.051724            0.051724  0.051724         1.0  19.333333
1172           0.051724            0.051724  0.051724         1.0  19.333333
1173           0.051724            0.051724  0.051724         1.0  19.333333
1174           0.051724            0.051724  0.051724         1.0  19.333333

     leverage  conviction
1170 0.049049         inf
1171 0.049049         inf
1172 0.049049         inf
1173 0.049049         inf
1174 0.049049         inf
```

+ Code + Markdown

Rules for items of Sweden.

[14]:
```
#Sweden
frq_items = apriori(basket_swe, min_support = 0.05, use_colnames = True)
rules = association_rules(frq_items, metric ="lift", min_threshold = 1)
rules = rules.sort_values(['confidence', 'lift'], ascending =[False, False])
print(rules.head())
```

```
                      antecedents                      consequents  \
0     (12 PENCILS SMALL TUBE SKULL)      (PACK OF 72 SKULL CAKE CASES)
1     (PACK OF 72 SKULL CAKE CASES)      (12 PENCILS SMALL TUBE SKULL)
4           (36 DOILIES DOLLY GIRL)     (ASSORTED BOTTLE TOP  MAGNETS)
5    (ASSORTED BOTTLE TOP  MAGNETS)            (36 DOILIES DOLLY GIRL)
180  (CHILDRENS CUTLERY DOLLY GIRL)  (CHILDRENS CUTLERY CIRCUS PARADE)

     antecedent support  consequent support   support  confidence  lift  \
0              0.055556            0.055556  0.055556         1.0  18.0
1              0.055556            0.055556  0.055556         1.0  18.0
4              0.055556            0.055556  0.055556         1.0  18.0
5              0.055556            0.055556  0.055556         1.0  18.0
180            0.055556            0.055556  0.055556         1.0  18.0

     leverage  conviction
0    0.052469         inf
1    0.052469         inf
4    0.052469         inf
5    0.052469         inf
180  0.052469         inf
```

Result: Apriori algorithm was successfully trained and tested on a sales dataset.

EXPERIMENT 9: Implementation of NLP programs

Aryan Jalla
RA1911003010729

AIM:

To write a program to perform sentiment analysis on given statements.cd

PROCEDURE:

1. Load the CSV dataset into Pandas DataFrame
2. Remove the unnecessary columns from the DataFrame
3. Splitting the dataset into train and test set
4. Removing neutral sentiments
5. Filter the *stopwords* from the text entries
6. Define *word features* and their extraction from text
7. Train Naive Bayes classifier for classification of text into positive or negative
8. Test the trained classifier on our test-set

DATASET:

[14...

	text	sentiment
0	RT @NancyLeeGrahn: How did everyone feel about...	Neutral
1	RT @ScottWalker: Didn't catch the full #GOPdeb...	Positive
2	RT @TJMShow: No mention of Tamir Rice and the ...	Neutral
3	RT @RobGeorge: That Carly Fiorina is trending ...	Positive
4	RT @DanScavino: #GOPDebate w/ @realDonaldTrump...	Positive

CODE:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O
(e.g. pd.read_csv)
from sklearn.model_selection import train_test_split #
function for splitting data to train and test sets

import nltk
from nltk.corpus import stopwords
from nltk.classify import SklearnClassifier

from wordcloud import WordCloud,STOPWORDS
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')

from subprocess import check_output


data = pd.read_csv('../input/Sentiment.csv')
# Keeping only the necessary columns
data = data[['text','sentiment']]


# First of all, splitting the dataset into a training
and a testing set. The test set is the 10% of the
original dataset. For this particular analysis I
dropped the neutral tweets, as my goal was to only
differentiate positive and negative tweets.


# Splitting the dataset into train and test set
train, test = train_test_split(data,test_size = 0.1)
```

```
# Removing neutral sentiments
train = train[train.sentiment != "Neutral"]


# As a next step I separated the Positive and Negative
tweets of the training set in order to easily visualize
their contained words.  After that I cleaned the text
from hashtags, mentions  and links. Now they were ready
for a WordCloud visualization which shows only the most
emphatic words of the Positive and Negative tweets.

train_pos = train[ train['sentiment'] == 'Positive']
train_pos = train_pos['text']
train_neg = train[ train['sentiment'] == 'Negative']
train_neg = train_neg['text']

def wordcloud_draw(data, color = 'black'):
    words = ' '.join(data)
    cleaned_word = " ".join([word for word in
words.split()
                            if 'http' not in word
                                and not
word.startswith('@')
                                and not
word.startswith('#')
                                and word != 'RT'
                            ])
    wordcloud = WordCloud(stopwords=STOPWORDS,
                      background_color=color,
                      width=2500,
                      height=2000
                  ).generate(cleaned_word)
```

```
    plt.figure(1,figsize=(13, 13))
    plt.imshow(wordcloud)
    plt.axis('off')
    plt.show()

print("Positive words")
wordcloud_draw(train_pos,'white')
print("Negative words")
wordcloud_draw(train_neg)


# Interesting to notice the following words and
expressions in the positive word set:
#  **truth**, **strong**, **legitimate**,
**together**, **love**, **job**
#
# In my interpretation, people tend to believe that
their ideal candidate is truthful, legitimate, above
good and bad.


# At the same time, negative tweets contains words
like:
# **influence**, **news**, **elevator music**,
**disappointing**, **softball**, **makeup**, **cherry
picking**, **trying**
#
# In my understanding people missed the decisively
action and considered the scolded candidates too soft
and cherry picking.
```

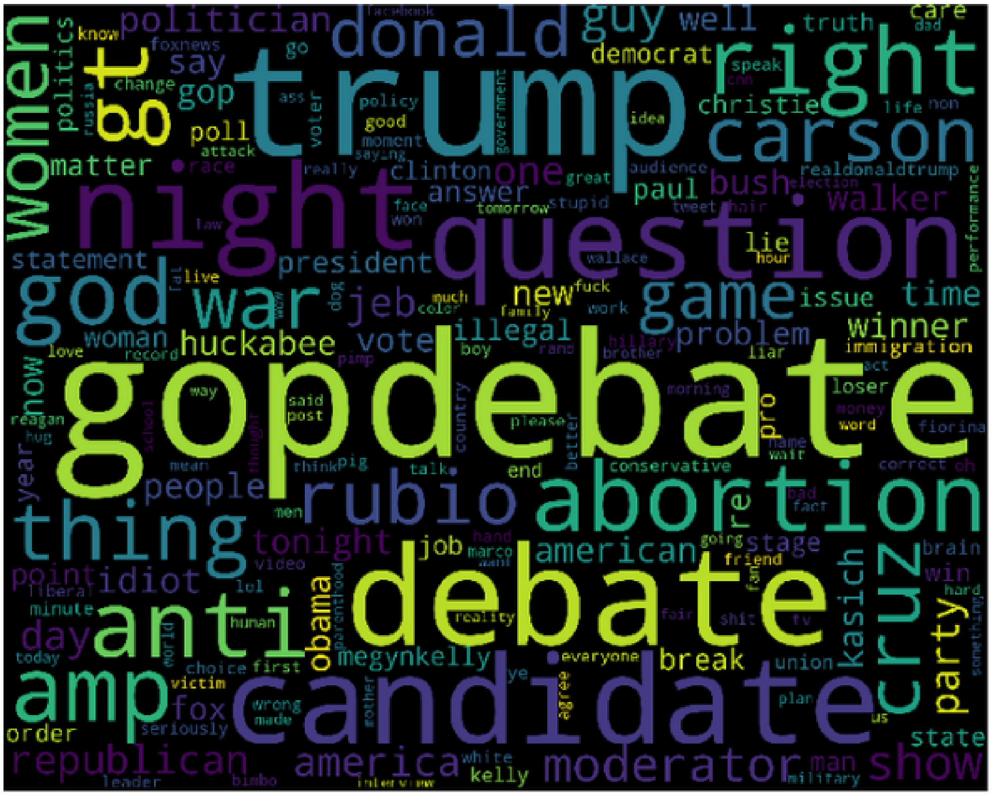
```
# After the visualization, I removed the hashtags,
mentions, links and stopwords from the training set.
#
# **Stop Word:** Stop Words are words which do not
contain important significance to be used in Search
Queries. Usually these words are filtered out from
search queries because they return vast amounts of
unnecessary information. ( the, for, this etc. )

tweets = []
stopwords_set = set(stopwords.words("english"))

for index, row in train.iterrows():
    words_filtered = [e.lower() for e in
row.text.split() if len(e) >= 3]
    words_cleaned = [word for word in words_filtered
    if 'http' not in word
    and not word.startswith('@')
    and not word.startswith('#')
    and word != 'RT']
    words_without_stopwords = [word for word in
words_cleaned if not word in stopwords_set]
    tweets.append((words_without_stopwords,
row.sentiment))

test_pos = test[ test['sentiment'] == 'Positive']
test_pos = test_pos['text']
test_neg = test[ test['sentiment'] == 'Negative']
test_neg = test_neg['text']
```

```
# As a next step I extracted the so called features
with nltk lib, first by measuring a frequent
distribution and by selecting the resulting keys.

# Extracting word features
def get_words_in_tweets(tweets):
    all = []
    for (words, sentiment) in tweets:
    all.extend(words)
    return all

def get_word_features(wordlist):
    wordlist = nltk.FreqDist(wordlist)
    features = wordlist.keys()
    return features
w_features =
get_word_features(get_words_in_tweets(tweets))

def extract_features(document):
    document_words = set(document)
    features = {}
    for word in w_features:
    features['contains(%s)' % word] = (word in
document_words)
    return features


# Hereby I plotted the most frequently distributed
words. The most words are centered around debate
nights.

wordcloud_draw(w_features)
```

```
# Using the nltk NaiveBayes Classifier I classified the
extracted tweet word features.

# Training the Naive Bayes classifier
training_set =
nltk.classify.apply_features(extract_features,tweets)
classifier =
nltk.NaiveBayesClassifier.train(training_set)


# Finally, with my selected metrics, I tried to measure
how the classifier algorithm scored.


neg_cnt = 0
pos_cnt = 0
for obj in test_neg:
     res =
classifier.classify(extract_features(obj.split()))
     if(res == 'Negative'):
     neg_cnt = neg_cnt + 1
for obj in test_pos:
     res =
classifier.classify(extract_features(obj.split()))
     if(res == 'Positive'):
     pos_cnt = pos_cnt + 1

print('[Negative]: %s/%s '  % (len(test_neg),neg_cnt))

print('[Positive]: %s/%s '  % (len(test_pos),pos_cnt))
```

OUTPUT:

```
wordcloud_draw(w_features)
```




```
neg_cnt = 0
pos_cnt = 0
for obj in test_neg:
    res =  classifier.classify(extract_features(obj.split()))
    if(res == 'Negative'):
        neg_cnt = neg_cnt + 1
for obj in test_pos:
    res =  classifier.classify(extract_features(obj.split()))
    if(res == 'Positive'):
        pos_cnt = pos_cnt + 1

print('[Negative]: %s/%s '  % (len(test_neg),neg_cnt))
print('[Positive]: %s/%s '  % (len(test_pos),pos_cnt))
```

```
[Negative]: 851/814
[Positive]: 225/67
```

RESULTS:

Sentiment analysis was performed successfully on the given statements

Aryan Jalla

RA1911003010729

Experiment-10

Implementation of Natural Language Problem – Text to Speech

Aim:

To implement natural language problem programs- Text to Speech.

Algorithm:

1. Import, install, download all the required modules / packages/ libraries required to perform natural language processing activities.
2. Perform tokenization and display the output.
3. Convert the given input into bi-grams, tri-grams and n- grams as required.
4. Perform stemming for the given input.
5. Perform part-of-speech tagging and display the output for the given input.
6. Implement Named entity recognition on the given input.
7. Perform text-to-speech with the help of gTTS module.

Code:

```
!pip install gTTS import nltk

import nltk.corpus


#Tokenization

from nltk.tokenize import word_tokenize

chess = "Samay Raina is the best chess streamer in the world" nltk.download('punkt')

word_tokenize(chess)


#sentence tokenizer

from nltk.tokenize import sent_tokenize

chess2 = "Samay Raina is the best chess streamer in the world. Sagar Sh ah is the best
chess coach in the world"
```

sent_tokenize(chess2)

#Checking the number of tokens len(word_tokenize(chess))

#bigrams and n-grams

astronaut = "Can anybody hear me or am I talking to myself? My mind is running empty in the search for someone else"

astronaut_token=(word_tokenize(astronaut)) list(nltk.bigrams(astronaut_token)) list(nltk.trigrams(astronaut_token)) list(nltk.ngrams(astronaut_token,5))

#Stemming

from nltk.stem import PorterStemmer my_stem = PorterStemmer() my_stem.stem("eating") my_stem.stem("going") my_stem.stem("shopping")

#pos-tagging

tom ="Tom Hanks is the best actor in the world" tom_token = word_tokenize(tom) nltk.download('averaged_perceptron_tagger') nltk.pos_tag(tom_token)

#Named entity recognition from nltk import ne_chunk

president = "Barack Obama was the 44th President of America" president_token = word_tokenize(president)

president_pos = nltk.pos_tag(president_token)

nltk.download('maxent_ne_chunker') nltk.download('words') print(ne_chunk(president_pos))

from gtts import gTTS

from IPython.display import Audio

tts = gTTS('Hello Atul, How are you') tts.save('1.wav')

sound_file = '1.wav' Audio(sound_file, autoplay=True)

Output:

a) **Tokenization:**

```
from nltk.tokenize import word_tokenize
chess = "Samay Raina is the best chess streamer in the world"
nltk.download('punkt')
word_tokenize(chess)
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
['Samay',
 'Raina',
 'is',
 'the',
 'best',
 'chess',
 'streamer',
 'in',
 'the',
 'world']
```

b) **Sentence tokenizer:**

```
[ ] from nltk.tokenize import sent_tokenize
    chess2 = "Samay Raina is the best chess streamer in the world. Sagar Shah is the best chess coach in the world"
    sent_tokenize(chess2)
```

```
['Samay Raina is the best chess streamer in the world.',
 'Sagar Shah is the best chess coach in the world']
```

c) **Number of tokens** (for chess = "Samay Raina is the best chess streamed in the world")

```
[ ] len(word_tokenize(chess))
```

```
10
```

d) **Bigrams:**

```
astronaut = "Can anybody hear me or am I talking to myself? My mind is running empty in the search for someone else"

astronaut_token=(word_tokenize(astronaut))
list(nltk.bigrams(astronaut_token))
list(nltk.trigrams(astronaut_token))
list(nltk.ngrams(astronaut_token,5))
```

```
[('Can', 'anybody', 'hear', 'me', 'or'),
 ('anybody', 'hear', 'me', 'or', 'am'),
 ('hear', 'me', 'or', 'am', 'I'),
 ('me', 'or', 'am', 'I', 'talking'),
 ('or', 'am', 'I', 'talking', 'to'),
 ('am', 'I', 'talking', 'to', 'myself'),
 ('I', 'talking', 'to', 'myself', '?'),
 ('talking', 'to', 'myself', '?', 'My'),
 ('to', 'myself', '?', 'My', 'mind'),
 ('myself', '?', 'My', 'mind', 'is'),
 ('?', 'My', 'mind', 'is', 'running'),
 ('My', 'mind', 'is', 'running', 'empty'),
 ('mind', 'is', 'running', 'empty', 'in'),
 ('is', 'running', 'empty', 'in', 'the'),
 ('running', 'empty', 'in', 'the', 'search'),
 ('empty', 'in', 'the', 'search', 'for'),
 ('in', 'the', 'search', 'for', 'someone'),
 ('the', 'search', 'for', 'someone', 'else')]
```

e) Trigrams:

```
astronaut = "Can anybody hear me or am I talking to myself? My mind is running empty in the search for someone else"

astronaut_token=(word_tokenize(astronaut))
list(nltk.bigrams(astronaut_token))
list(nltk.trigrams(astronaut_token))
list(nltk.ngrams(astronaut_token,5))
```

```
[('Can', 'anybody', 'hear', 'me', 'or'),
 ('anybody', 'hear', 'me', 'or', 'am'),
 ('hear', 'me', 'or', 'am', 'I'),
 ('me', 'or', 'am', 'I', 'talking'),
 ('or', 'am', 'I', 'talking', 'to'),
 ('am', 'I', 'talking', 'to', 'myself'),
 ('I', 'talking', 'to', 'myself', '?'),
 ('talking', 'to', 'myself', '?', 'My'),
 ('to', 'myself', '?', 'My', 'mind'),
 ('myself', '?', 'My', 'mind', 'is'),
 ('?', 'My', 'mind', 'is', 'running'),
 ('My', 'mind', 'is', 'running', 'empty'),
 ('mind', 'is', 'running', 'empty', 'in'),
 ('is', 'running', 'empty', 'in', 'the'),
 ('running', 'empty', 'in', 'the', 'search'),
 ('empty', 'in', 'the', 'search', 'for'),
 ('in', 'the', 'search', 'for', 'someone'),
 ('the', 'search', 'for', 'someone', 'else')]
```

f) N-grams:


```
astronaut = "Can anybody hear me or am I talking to myself? My mind is running empty in the search for someone else"

astronaut_token=(word_tokenize(astronaut))
list(nltk.bigrams(astronaut_token))
list(nltk.trigrams(astronaut_token))
list(nltk.ngrams(astronaut_token,5))
```

```
[('Can', 'anybody', 'hear', 'me', 'or'),
 ('anybody', 'hear', 'me', 'or', 'am'),
 ('hear', 'me', 'or', 'am', 'I'),
 ('me', 'or', 'am', 'I', 'talking'),
 ('or', 'am', 'I', 'talking', 'to'),
 ('am', 'I', 'talking', 'to', 'myself'),
 ('I', 'talking', 'to', 'myself', '?'),
 ('talking', 'to', 'myself', '?', 'My'),
 ('to', 'myself', '?', 'My', 'mind'),
 ('myself', '?', 'My', 'mind', 'is'),
 ('?', 'My', 'mind', 'is', 'running'),
 ('My', 'mind', 'is', 'running', 'empty'),
 ('mind', 'is', 'running', 'empty', 'in'),
 ('is', 'running', 'empty', 'in', 'the'),
 ('running', 'empty', 'in', 'the', 'search'),
 ('empty', 'in', 'the', 'search', 'for'),
 ('in', 'the', 'search', 'for', 'someone'),
 ('the', 'search', 'for', 'someone', 'else')]
```


g) Stemming:

```
from nltk.stem import PorterStemmer
my_stem = PorterStemmer()
my_stem.stem("eating")
my_stem.stem("going")
my_stem.stem("shopping")
```

```
'shop'
```

h) Pos-tagging:

```
[ ] tom ="Tom Hanks is the best actor in the world"
    tom_token = word_tokenize(tom)
    nltk.download('averaged_perceptron_tagger')
    nltk.pos_tag(tom_token)
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
[('Tom', 'NNP'),
 ('Hanks', 'NNP'),
 ('is', 'VBZ'),
 ('the', 'DT'),
 ('best', 'JJS'),
 ('actor', 'NN'),
 ('in', 'IN'),
 ('the', 'DT'),
 ('world', 'NN')]
```

i) **Named entity recognition:**

```
from nltk import ne_chunk
president = "Barack Obama was the 44th President of America"
president_token = word_tokenize(president)
president_pos = nltk.pos_tag(president_token)

nltk.download('maxent_ne_chunker')
nltk.download('words')
print(ne_chunk(president_pos))

from gtts import gTTS
from IPython.display import Audio
tts = gTTS('Hello Atul, How are you')
tts.save('1.wav')
sound_file = '1.wav'
Audio(sound_file, autoplay=True)
```

```
[nltk_data] Downloading package maxent_ne_chunker to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping chunkers/maxent_ne_chunker.zip.
[nltk_data] Downloading package words to /root/nltk_data...
[nltk_data]   Unzipping corpora/words.zip.
(S
  (PERSON Barack/NNP)
  (PERSON Obama/NNP)
  was/VBD
  the/DT
  44th/JJ
  President/NNP
  of/IN
  (GPE America/NNP))
```

Automatic document saving has been pending for 12 minutes. Reloading may fix the problem. Save and reload the page.

Result:

Thus, google text to speech has been performed along with other language processing successfully.