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: Akash Kumar Singh
Reg No: RA1911003010750

SRM 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

SRM INSTITUTE OF SCIENCE & TECHNOLOGY

DEPARTMENT OF NETWORKING & COMMUNICATIONS

18CSC305J-ARTIFICIAL INTELLIGENCE

SEMESTER – 6 BATCH-2

REGISTRATION NUMBER	**RA1911003010750**
NAME	**AKASH KUMAR SINGH**

B. Tech- CSE, Third Year (Section: C2)

Year 2021-2022 / Even Semester

INDEX

Ex No	DATE	Title	Page No	Marks
1	12/01/2022	**Toy Problem:** The Goat Wolf and Cabbage Riddle		

EXPERIMENT -1

TOY PROBLEM

THE GOAT WOLF AND CABBAGE RIDDLE

Aim: To implement a toy problem in Python using Google Colab.

Problem Title: The Goat Wolf And Cabbage Riddle

Problem Statement: There is a farmer who wishes to cross a river but he is not alone. He also has a goat, a wolf, and a cabbage along with him. Your task is to get everything to the other side.

Restrictions:

1. only you can handle the boat
2. when you're in the boat, there is only space for one more item
3. you can't leave the goat alone with the wolf, nor with the cabbage (or something will be eaten)

Algorithm:

Implementation of a depth-first search for a solution to the goat-wolf-cabbage puzzle.
Depth-first search: Search used by the Prolog interpreter. Simple to implement, but may get stuck in an infinite branch and fail to find a goal state in another branch. May not find the shortest route to goal.

To extend a path into an answer-path do

if the path has reached a goal state

then return it as an answer-path

else if the current path can be extended to a new state

then save the current state and continue to extend the path from the new state

else

backtrack to extend the path from the previous state to a different child

To extend the path into a new state do

find another state which can be reached from the last state in the path; the state found is the new state

Depth-First Search in Prolog

```
depth_first_search(AnsPath) :-
    initial(Init), depth_first([Init], AnsPath)
    depth_first([S|Path], [S]) :- final(S), !.
```

```
depth_first([S|Path], [S|AnsPath]) :-
        extend([S|Path], S1),
        depth_first([S1, S|Path], AnsPath).
extend([S|Path], S1) :-
        next_state(S, S1), not(member(S1, [S|Path]))
```

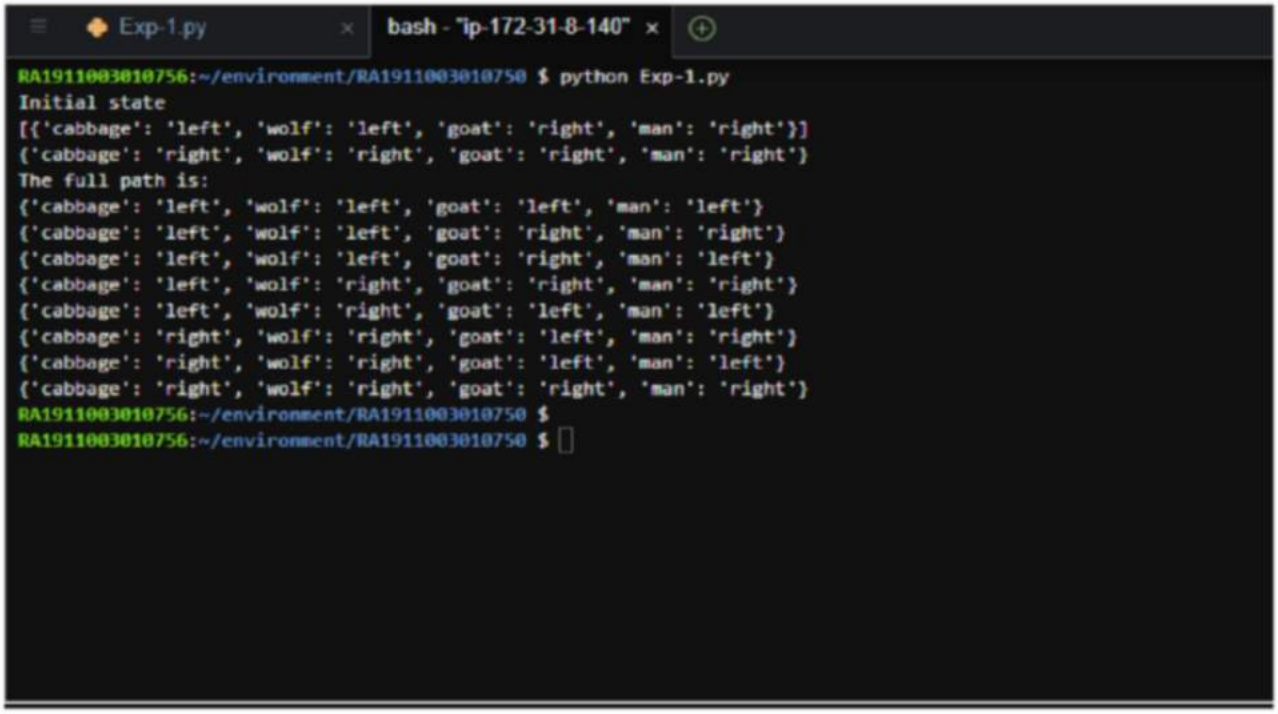
Tools: Google Colab

Code:

```
entity = ['goat', 'wolf', 'cabbage']
path = []

# Defines who can eat whom
def eats(x, y):
    if x == 'goat' and y == 'cabbage':
        return True
    elif x == 'wolf' and y == 'goat':
        return True
    else:
        return False

# Defines if a pair of entities is safe to be left alone on one side
# of the river.
def safe_pair(x, y):
    if eats(x, y) or eats(y, x):
        return False
    else:
        return True


def state_of(who, state):
    try:
        return state[who]
    except KeyError:
        state[who] = False
        return False


def safe_state(state):
    if state_of('man', state) == state_of('goat', state):
        return True
    elif state_of('goat', state) == state_of('wolf', state):
        return False
    elif state_of('goat', state) == state_of('cabbage', state):
        return False
    else:
        return True


def move(who, state):
    if state[who] == 'left':
        state[who] = 'right'
    else:
        state[who] = 'left'
    return state


def goal_reach(state):
```

```
    if not state:
        return False
    return (state_of('man', state)=='right' and
            state_of('goat', state)=='right' and
            state_of('wolf', state)=='right' and
            state_of('cabbage',state)=='right')

def check_add_child(child, list_states):
    if safe_state(child):
      list_states.append(child)
    return list_states



def expand_states(state):
    children = []
    child = state.copy()
    # the man can also move alone
    move('man', child)
    check_add_child(child, children)
    for ent in entity:
        # Move one object on the same side as the man
        if state_of(ent, state) == state_of('man', state):
            child = state.copy()
            move('man', child)
            move(ent, child)
            check_add_child(child, children)
        #else:
      #print "unsafe state", child
    return children

# Searches for a solution from the initial state
def search_sol(state):
    path.append(state)
    next = state.copy()
    while next and not goal_reach(next):
        nl = expand_states(next)
        next = {}
        for child in nl:
            if not (child in path):
                next = child
                path.append(next)
                break
    return next


initial_state = {}
initial_state['man'] = 'left'
for e in entity:
    initial_state[e] = 'left'


print("Expanding initial state")
print(expand_states(initial_state))

print ("Searching for a solution from the initial state:")
print (search_sol(initial_state))

# Evaluate the variable path to see the solution backwards.
print ("The full path is:")
for s in path:
    print (s)
```

Output:



Result: Successfully implemented the toy problem.

SRM INSTITUTE OF SCIENCE & TECHNOLOGY

DEPARTMENT OF NETWORKING COMMUNICATIONS

18CSC305J-ARTIFICIAL INTELLIGENCE

SEMESTER – 6 BATCH-2

REGISTRATION NUMBER	**RA1911003010750**
NAME	**AKASH KUMAR SINGH**

B. Tech- CSE, Third Year (Section: C2)

Year 2021-2022 / Even Semester

INDEX

Ex No	DATE	Title	Page No	Marks
1	**07/02/2022**	Agent Program: (Kruskal's Minimum Spanning Tree)		

EXPERIMENT -2

Agent Program

Aim: To implement an agent program in python using Google Colab.

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.

Algorithm:

In Kruskal's algorithm, we start from edges with the lowest weight and keep adding the edges until the goal is reached. The steps to implement Kruskal's algorithm are listed as follows -

- First, sort all the edges from low weight to high.
- Now, take the edge with the lowest weight and add it to the spanning tree. If the edge to be added creates a cycle, then reject the edge.
- Continue to add the edges until we reach all vertices, and a minimum spanning tree is created.

The algorithm is a Greedy Algorithm. The Greedy Choice is to pick the smallest weight edge that does not cause a cycle in the MST constructed so far.

Step 1: Create a forest F in such a way that every vertex of the graph is a sep arate tree.
Step 2: Create a set E that contains all the edges of the graph.
Step 3: Repeat Steps 4 and 5 **while** E is NOT EMPTY and F is not spanning
Step 4: Remove an edge from E with minimum weight
Step 5: IF the edge obtained in Step 4 connects two different trees, then add it to the forest F
(**for** combining two trees into one tree).
ELSE

Discard the edge
Step 6: END

Tools: Google Colab

Time Complexity:

The time complexity of Kruskal's algorithm is O(ElogE) or O(VlogV), where E is the no. of edges, and V is the no. of vertices.

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):
    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
    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(1, 2, 6)
g.addEdge(2, 3, 2)
g.addEdge(1, 3, 4)
g.addEdge(1, 0, 8)
g.addEdge(0, 3, 14)


g.KruskalMST()
```

Output:

```
      print("%d -- %d == %d" % (u, v, weight))
    print("Minimum Spanning Tree" , minimumCost)

# Driver code
g = Graph(4)
g.addEdge(1, 2, 6)
g.addEdge(2, 3, 2)
g.addEdge(1, 3, 4)
g.addEdge(1, 0, 8)
g.addEdge(0, 3, 14)


g.KruskalMST()
```

```
Edges in the constructed MST
2 -- 3 == 2
1 -- 3 == 4
1 -- 0 == 8
Minimum Spanning Tree 14
```

Result: Successfully implemented the agent program.

SRM INSTITUTE OF SCIENCE & TECHNOLOGY

DEPARTMENT OF NETWORKING & COMMUNICATIONS

18CSC305J-ARTIFICIAL INTELLIGENCE

SEMESTER – 6

BATCH-2

REGISTRATION NUMBER	**RA1911003010750**
NAME	**Akash Kumar Singh**

B.Tech - CSE / CC, Third Year (Section: C2)

Year 2021-2022 / Even Semester

INDEX

Ex No	DATE	Title	Page No	Marks
1	14/02/22	**Implementation of constraint satisfaction problems**		

EXPERIMENT - 3

(Crypt Arithmetic Puzzle)

NAME: AKASH KUMAR SINGH
REG NO: RA1911003010750

Problem Statement

- *assign each letter a digit from 0 to 9 so that the arithmetic works out correctly*
- *the rules are that all occurrences of a letter must be assigned the same digit*
- *no digit can be assigned to more than one letter*
- *example*

```
  SEND
+ MORE
  --------
 MONEY
  -------
```

Algorithm

Base Function

- Take 3 strings as input
- Make string of unique letters with them
- Make a dictionary with keys as unique values from strings combined initialized at -1
- Make an array that determines whether a number (0 to 9) has been assigned or not
- Pass it to recursive function

Recursive Function

- If current_index is equal to length of unique string
  - convert s1, s2 and s3 to their respective nums
  - If num1 + num2 equals to num3
  - Print result

- Else
    - Assign values left from array of number to a current index char
    - Call recursive function again with updated dictionary
    - Re assign values of current index char to 0 and map it to -1 (Backtracking Step)

Optimization Technique: The algorithm above actually has a lot in common with the permutations algorithm, it pretty much just creates all arrangements of the mapping from characters to digits and tries each until one works or all have been successfully tried. For a large puzzle, this could take a while. A smarter algorithm could take into account the structure of the puzzle and avoid going down dead-end paths. For example, if we assign the characters starting from the one's place and moving to the left, at each stage, we can verify the correctness of what we have so far before we continue onwards. This definitely complicates the code but leads to a tremendous improvement in efficiency, making it much more feasible to solve large puzzles.

- Start by examining the rightmost digit of the topmost row, with a carry of 0
- If we are beyond the leftmost digit of the puzzle, return true if no carry, false otherwise
- If we are currently trying to assign a char in one of the addends
  If char already assigned, just recur on the row beneath this one, adding value into the sumIf not assigned, then
    - for (every possible choice among the digits not in use)
      make that choice and then on row beneath this one, if successful, return trueif !successful, unmake assignment and try another digit
    - return false if no assignment worked to trigger backtracking
- Else if trying to assign a char in the sum
- If char assigned & matches correct,
  recur on next column to the left with carry, if success return true,
- If char assigned & doesn't match, return false
- If char unassigned & correct digit already used, return false
- If char unassigned & correct digit unused,
  assign it and recur on next column to left with carry, if success return true
- return false to trigger backtracking.

Tool : AWS cloud9 and Python 3.9.0

SRM INSTITUTE OF SCIENCE & TECHNOLOGY

DEPARTMENT OF NETWORKING & COMMUNICATIONS

18CSC305J-ARTIFICIAL INTELLIGENCE

SEMESTER – 6

BATCH-2

REGISTRATION NUMBER	**RA1911003010750**
NAME	**Akash Kumar Singh**

B.Tech - CSE / CC, Third Year (Section: C2)

Year 2021-2022 / Even Semester

Ex No	DATE	Title	Page No	Marks
1	21/02/22	Uniformed Search		

INDEX

EXPERIMENT – 4

(i) Write a Program to find shortest path for a given unweighted graph / tree by applying BFS method.

PROBLEM STATEMENT: Given an unweighted undirected graph, we have to find the shortest path from the given source to the given destination using the Breadth-First-Search

Breadth-First Search: Breadth-First Search (BFS) is an algorithm used for traversing graphs or trees. Traversing means visiting each node of the graph. Breadth-First Search is a recursive algorithm to search all the vertices of a graph or a tree. BFS in python can be implemented by using data structures like a dictionary and lists. Breadth-First Search in tree and graph is almost the same. The only difference is that the graph may contain cycles, so we may traverse to the same node again.

Algorithm:

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.

Tool: AWS cloud9 and Python 3.9.0

Python Code

```
def add_edge(adj, src, dest):

        adj[src].append(dest);
        adj[dest].append(src);
def BFS(adj, src, dest, v, pred, dist):

        queue = []
        visited = [False for i in range(v)];
        for i in range(v):

                dist[i] = 1000000
                pred[i] = -1;
        visited[src] = True;
        dist[src] = 0;
        queue.append(src);
        while (len(queue) != 0):
                u = queue[0];
                queue.pop(0);
                for i in range(len(adj[u])):

                        if (visited[adj[u][i]] == False):
                                visited[adj[u][i]] = True;
                                dist[adj[u][i]] = dist[u] + 1;
                                pred[adj[u][i]] = u;
                                queue.append(adj[u][i]);

                                if (adj[u][i] == dest):
                                        return True;

        return False;

def printShortestDistance(adj, s, dest, v):

        pred=[0 for i in range(v)]
        dist=[0 for i in range(v)];
```

```
    if (BFS(adj, s, dest, v, pred, dist) == False):
        print("Given source and destination are not connected")

    path = []
    crawl = dest;
    crawl = dest;
    path.append(crawl);

    while (pred[crawl] != -1):
        path.append(pred[crawl]);
        crawl = pred[crawl];


    print("Shortest path length is : " + str(dist[dest]), end = '')
    print("\nPath is : : ")

    for i in range(len(path)-1, -1, -1):
        print(path[i], end=' ')

if __name__=='__main__':
    v = 8;
    adj = [[] for i in range(v)];
    add_edge(adj, 0, 1);
    add_edge(adj, 0, 3);
    add_edge(adj, 1, 2);
    add_edge(adj, 3, 4);
    add_edge(adj, 3, 7);
    add_edge(adj, 4, 5);
    add_edge(adj, 4, 6);
    add_edge(adj, 4, 7);
    add_edge(adj, 5, 6);
    add_edge(adj, 6, 7);
    add_edge(adj, 2, 6);

    source = 0
    dest = 6;
    printShortestDistance(adj, source, dest, v);
```

Output

Exp-4(BFS).py × Exp-4(DFS).py ×

```
66      print("\nPath is : : ")
67
68      for i in range(len(path)-1, -1, -1):
69          print(path[i], end=' ')
70
71  if __name__=='__main__':
72
73      v = 8;
74
75
76      adj = [[] for i in range(v)];
77
78      add_edge(adj, 0, 1);              #     0-------1
79      add_edge(adj, 0, 3);              #     |       ||
80      add_edge(adj, 1, 2);              #     |       |
81      add_edge(adj, 3, 4);              #     3       2
82      add_edge(adj, 3, 7);              #    / \      |
83      add_edge(adj, 4, 5);              #   /   \     |
84      add_edge(adj, 4, 6);              #  4-----7    |
85      add_edge(adj, 4, 7);              #  | \   |    |
86      add_edge(adj, 5, 6);        #     |  \  |    |
87      add_edge(adj, 6, 7);        #     |   \ |    /
88      add_edge(adj, 2, 6);        #     |    \|   /
89      source = 0                  #     5-----6----
90      dest = 6;
91      printShortestDistance(adj, source, dest, v);
92
```

RA1911003010750/Exp-4 × RA1911003010750/Exp-4 ×

Run C RA1911003010750/Exp-4\{ Runner: Python 3

```
Shortest path length is : 3
Path is : :
0 1 2 6

Process exited with code: 0
```

(ii) Write a Program to find shortest path for a given unweighted graph / tree by applying DFS method.

PROBLEM STATEMENT: Given an unweighted undirected graph, we have to find the shortest path from the given source to the given destination using the Depth-First-Search

The Depth-First Search: The first algorithm I will be discussing is Depth-First search which as the name hints at, explores possible vertices (from a supplied root) down each branch before backtracking. This property allows the algorithm to be implemented succinctly in both iterative and recursive forms. Below is a listing of the actions performed upon each visit to a node.

- Mark the current vertex as being visited.
- Explore each adjacent vertex that is not included in the visited set.

Algorithm:

1. We will start by putting any one of the graph's vertices 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.

Python Code

```
import sys

class ShortestPath:
  def __init__(self, start, end):
    self.start = start
    self.end = end
    self.shortLength = sys.maxsize

  def findPath(self):
    self.dfs(self.start)
    self.trace_route()

  def dfs(self, vertex):
    global length
    length +=1
    if length > self.shortLength:
      return
    if vertex == self.end:
      self.shortLength = length
```

```
      return
    visited[vertex] = 1
    #iterate through all unvisited neighbors vertices
    for i in range(N):
      if adjacencyM[vertex][i] == 1 and visited[i] == 0:
        nbr = i
        prev[nbr] = vertex
        self.dfs(nbr)
    length -=1
  def trace_route(self):
    vertex = self.end
    route = []
    while vertex != -1:
      route.append(vertices_names[vertex])
      vertex = prev[vertex]
    route.reverse()
    print(route)

if __name__ == '__main__':
  #vertices or nodes
  vertices_names = ['A', 'B', 'C', 'D', 'E'];
  #number of vertices
  N = len(vertices_names)

  #Adjacency Matrix
  adjacencyM = [[0, 1, 0, 0, 0],
          [1, 0, 1, 0, 1],
          [0, 1, 0, 1, 1],
          [0, 0, 1 ,0, 1],
          [0, 1, 0, 1, 0]];

  #List mapping of vertices to mark them visited
  visited = [0 for x in range(N)]
  #List to stores preceding vertices
  prev = [-1 for x in range(N)]

  #Driver code
  sp = ShortestPath(0, 4)
  length = 0
  sp.findPath()
```

Output

Result: Successfully solved and found the shortest path algorithm using BFS and DFS algorithm.

SRM INSTITUTE OF SCIENCE & TECHNOLOGY

DEPARTMENT OF NETWORKING & COMMUNICATIONS

18CSC305J-ARTIFICIAL INTELLIGENCE

SEMESTER –

6 BATCH-2

REGISTRATION NUMBER	**RA1911003010750**
NAME	**Akash Kumar Singh**

B.Tech- CSE / CC, Third Year (Section: C2)

Year 2021-2022 / Even Semester

INDEX

Ex No	DATE	Title	Page No	Marks
1	07/03/22	**Developing Best first search and A* Algorithm for real world problems**		

Tool: VS Code and Python 3.9.0

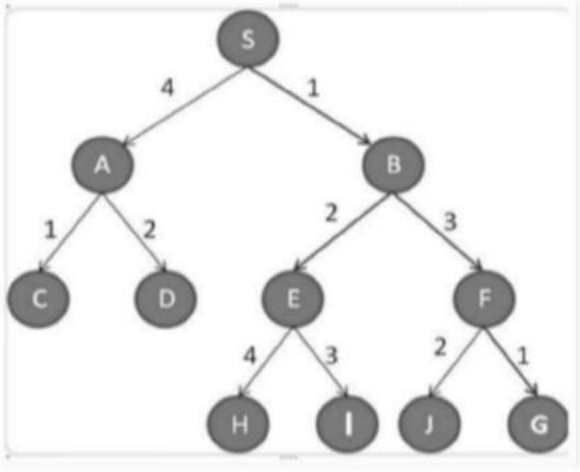
Best First Search (Informed Search)

In BFS and DFS, when we are at a node, we can consider any of the adjacent as next node. So both BFS and DFS blindly explore paths without considering any cost function. The idea of Best First Search is to use an evaluation function to decide which adjacent is most promising and then explore. Best First Search falls under the category of Heuristic Search or Informed Search.
We use a priority queue to store costs of nodes. So the implementation is a variation of BFS, we just need to change Queue to PriorityQueue.

Algorithm :

```
1) Create an empty
    PriorityQueuePriorityQueue
    pq;
  2) Insert "start" in pq.
    pq.insert(start)
  3) Until PriorityQueue is empty
      u = PriorityQueue.DeleteMin
      If u is the goal
        Exit
      Else
        Foreach neighbor v of u
          If v "Unvisited"
            Mark v "Visited"
            pq.insert(v)
        Mark u "Examined"
End procedure
```

GRAPH for Best First Search:-



NODE	Heuristic Value

A	11
B	5
C	9
D	9
E	4
F	2
G	3
H	7
I	0
J	3
S	15

CODE:

Best First Search:

```
from collections import defaultdict from
queue import PriorityQueue v=11

graph=defaultdict(list) h =
{
    'A':11,
    'B':5,
    'C':9,
    'D':9,
    'E':4,
    'F':2,
    'G':3,
    'H':4,
    'I':0,
    'J':3,
    'S':15,
  }
  answer=[]
def best_first_search(start,end):
  visited=[] visited.append(start)
```

```
    q=PriorityQueue()
    q.put((h[start],start))


    while q.empty()==False:
        a=q.get()
        answer.append(a[1]) if
        a[1]==end:

            break
          else:


            for e in graph[a[1]]:

            if e[0] not in visited:
                visited.append(e[0])
                q.put((h[e[0]],e[0]))


def add_vertex(x,y,c):
    graph[x].append((y,c))

    add_vertex('S','A',4)

    add_vertex('S','B',1)

    add_vertex('A','C',1)

    add_vertex('A','D',2)

    add_vertex('B','E',2)

    add_vertex('B','F',3)

    add_vertex('E','H',4)

    add_vertex('E','I',3)

    add_vertex('F','J',2)

    add_vertex('F','G',1)

    best_first_search('S','I') print(answer)
```

OUTPUT

BFS

A* Algorithm

A heuristic algorithm sacrifices optimality, with precision and accuracy for speed, to solveproblems faster and more efficiently.
All graphs have different nodes or points which the algorithm has to take, to reach the finalnode. The paths between these nodes all have a numerical value, which is considered as the weight of the path. The total of all paths transverse gives you the cost of that route.
Initially, the Algorithm calculates the cost to all its immediate neighboring nodes,n, and chooses the one incurring the least cost. This process repeats until no new nodes can be chosen and all paths have been traversed. Then, you should consider the best path amongthem. If f(n) represents the final cost, then it can be denoted as :
f(n) = g(n) + h(n), where :
g(n) = cost of traversing from one node to another. This will vary from node to nodeh(n) = heuristic approximation of the node's value. This is not a real value but an approximation cost

Algorithm

- Make an open list containing starting node
    - If it reaches the destination node :
    - Make a closed empty list
    - If it does not reach the destination node, then consider a node with the lowest f-score in the open list

We are finished

- Else :

Put the current node in the list and check its neighbors

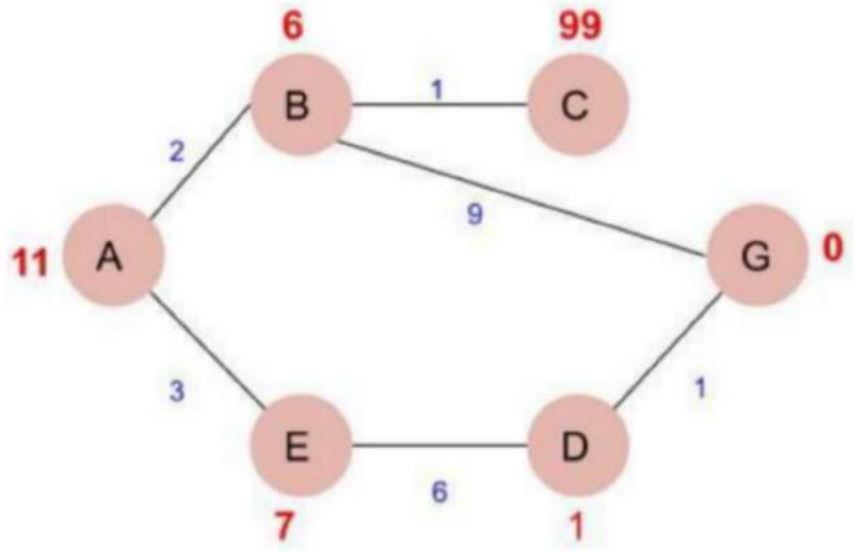
- For each neighbor of the current node :
    - If the neighbor has a lower g value than the current node and is in the closed list:

Replace neighbor with this new node as the neighbor's parent

- Else If (current g is lower and neighbor is in the open list):

Replace neighbor with the lower g value and change the neighbor's parent to the current node.

Else If the neighbor is not in both lists:Add it to the open list and set its g



Code:-

```
from collections import deque
class Graph:
def__init___(self, adjacency_list):
   self.adjacency_list = adjacency_list
def get_neighbors(self, v): return
   self.adjacency_list[v]
def h(self, n):
       H = {
       'A': 11,
       'B': 6,
       'C': 99,
       'D': 1,
       'E': 7,
       'G': 0,
       }
       return H[n]
def a_star_algorithm(self, start_node, stop_node):
   open_list = set([start_node])
       closed_list = set([])


       g = {}
       g[start_node] = 0
       parents = {}
       parents[start_node] = start_node

   while len(open_list) > 0: n =
       None
          for v in open_list:
          if n == None or g[v] + self.h(v) < g[n] + self.h(n): n =
             v;
          if n == None:
```

```
            print('Path does not exist!')
            return None
        if n == stop_node:
            reconst_path = [] while
            parents[n] != n:
                reconst_path.append(n) n
                = parents[n]
            reconst_path.append(start_node)
            reconst_path.reverse()
            print('Path found: {}'.format(reconst_path))
            return reconst_path
        for (m, weight) in self.get_neighbors(n):
        if m not in open_list and m not in closed_list:
            open_list.add(m)
                parents[m] = n
                g[m] = g[n] + weight
            else:
            if g[m] > g[n] + weight:
                g[m] = g[n] + weight
                parents[m] = n

                if m in closed_list:
                    closed_list.remove(m)
                    open_list.add(m)
        open_list.remove(n)
        closed_list.add(n)
    print('Path does not exist!')
    return None
adjacency_list = { 'A':
[('B', 2), ('E', 3)],
'B': [('C', 1),('G', 9)],
'C': None, 'E':
[('D', 6)],
'D': [('G', 1)],
}

graph1 = Graph(adjacency_list)
graph1.a_star_algorithm('A', 'G')
```

OUTPUT

A*

```
Path: ['S: 0', 'A: 2', 'C: 9', 'G: 17']


Process exited with code: 0
```

Result: A* and Best first search algorithms were implemented successfully.

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
DEPARTMENT OF NETWORKING & COMMUNICATIONS
18CSC305J-ARTIFICIAL INTELLIGENCE

SEMESTER – 6 BATCH-2

REGISTRATION NUMBER	**RA1911003010750**
NAME	**Akash Kumar Singh**

B.Tech- CSE / CC, Third Year (Section: C2)
Year 2021-2022 / Even Semester

INDEX

Ex No	DATE	Title	Page No	Marks
1	21/03/22	Unification and resolution for real world problems		

Tool: VS Code and Python 3.9.0

Aim:

Implementation of unification and resolution for real world problems.

Unification

Code:

```
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
            == '(':
            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):
    # Step 1:
        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__': #
  Data 1
#f1 = 'p(b(A), X, f(g(Z)))'

#f2 = 'p(Z, f(Y), f(Y))'

# Data 2
f1 = 'Q(a, g(x, a), f(y))'
```

```
f2 = 'Q(a, g(f(b), a), x)'


# Data 3

#f1 = 'Q(a, g(x, a, d), f(y))'

#f2 = 'Q(a, g(f(b), a), x)'


result = unify(f1, f2) if
not result:
  print('Unification failed!')
else:
  print('Unification successfully!')
  print(result)
```

Output:

```
Unification successfully!
['f(b)/x', 'f(y)/x']
```

Source Code:

```
import copy
import time


class Parameter: variable_count
  = 1


def __init__(self, name=None):
  if name:
    self.type = "Constant"
    self.name = name

  else:
```

```
        self.type = "Variable"

        self.name = "v" + str(Parameter.variable_count)

        Parameter.variable_count += 1


    def isConstant(self):

        return self.type == "Constant"

        def unify(self, type_, name):
        self.type = type_ self.name= name


    def __eq__(self, other):

        return self.name == other.name def

    str_(self):


        return self.name


class Predicate:

    def __init__(self, name, params):

        self.name = name self.params
        = params
    def __eq__(self, other):

        return self.name == other.name and all(a == b for a, b in zip(self.params, other.params))

    def __str__(self):

        return self.name + "(" + ",".join(str(x) for x in self.params) + ")"

    def getNegatedPredicate(self):

        return Predicate(negatePredicate(self.name), self.params)




class Sentence:

    sentence_count = 0
```

```
def __init__(self, string):

    self.sentence_index = Sentence.sentence_count
    Sentence.sentence_count += 1
    self.predicates = []
    self.variable_map = {}
    local = {}


    for predicate in string.split("|"):

        name = predicate[:predicate.find("(")]
        params = []


            for param in predicate[predicate.find("(") + 1:
                predicate.find(")")].split(","): if param[0].islower():
                    if param not in local: # Variable
                        local[param] = Parameter()
                    self.variable_map[local[param].name] = local[param]
                new_param = local[param]
            else:
                new_param = Parameter(param)
                self.variable_map[param] = new_param
            params.append(new_param)

        self.predicates.append(Predicate(name, params))

def getPredicates(self):

    return [predicate.name for predicate in self.predicates]

def findPredicates(self, name):

    return [predicate for predicate in self.predicates if predicate.name == name]

    def removePredicate(self, predicate):
        self.predicates.remove(predicate)
    for key, val in self.variable_map.items():
            if not val:
                self.variable_map.pop(key)

def containsVariable(self):

    return any(not param.isConstant() for param in self.variable_map.values())

def __eq__(self, other):

    if len(self.predicates) == 1 and self.predicates[0] == other:

        return True
    return False
```

```
  def __str__(self):

    return "".join([str(predicate) for predicate in self.predicates])



class KB:

  def __init__(self, inputSentences):
    self.inputSentences = [x.replace(" ", "") for x in inputSentences]
    self.sentences = []
    self.sentence_map = {}

  def prepareKB(self):

    self.convertSentencesToCNF()

    for sentence_string in self.inputSentences:

      sentence = Sentence(sentence_string) for
      predicate in sentence.getPredicates():

        self.sentence_map[predicate] = self.sentence_map.get(predicate, []) + [sentence]

  def convertSentencesToCNF(self):

    for sentenceIdx in range(len(self.inputSentences)):

      if "=>" in self.inputSentences[sentenceIdx]: # Do negation of the Premise and add them as
literal

        self.inputSentences[sentenceIdx] = negateAntecedent(self.inputSentences[sentenceIdx])

  def askQueries(self, queryList):
    results = []

    for query in queryList:

      negatedQuery = Sentence(negatePredicate(query.replace(" ", "")))
      negatedPredicate = negatedQuery.predicates[0] prev_sentence_map
      = copy.deepcopy(self.sentence_map)
      self.sentence_map[negatedPredicate.name] =
      self.sentence_map.get(negatedPredicate.name, []) +
            [negatedQuery] self.timeLimit = time.time() + 40

      try:

        result = self.resolve([negatedPredicate], [False]*(len(self.inputSentences) + 1))
      except:
        result = False
      self.sentence_map = prev_sentence_map
        if result:
          results.append("TRUE")
```

```
        else:
            results.append("FALSE")

    return results


  def resolve(self, queryStack, visited, depth=0):

    if time.time() > self.timeLimit:
        raise Exception if
    queryStack:

        query = queryStack.pop(-1)

        negatedQuery = query.getNegatedPredicate()
        queryPredicateName = negatedQuery.name
            if queryPredicateName not in self.sentence_map:
                return False
        else:

            queryPredicate = negatedQuery

                for kb_sentence in self.sentence_map[queryPredicateName]:
                    if not visited[kb_sentence.sentence_index]:

                    for kbPredicate in kb_sentence.findPredicates(queryPredicateName):

                        canUnify, substitution = performUnification(copy.deepcopy(queryPredicate),
copy.deepcopy(kbPredicate))


                        if canUnify:

                            newSentence = copy.deepcopy(kb_sentence)
                            newSentence.removePredicate(kbPredicate)
                            newQueryStack = copy.deepcopy(queryStack)


                            if substitution:

                                for old, new in substitution.items():

                                    if old in newSentence.variable_map:

                                        parameter = newSentence.variable_map[old]
                                        newSentence.variable_map.pop(old)
                                        parameter.unify("Variable" if new[0].islower() else "Constant", new)
                                        newSentence.variable_map[new] = parameter


"Constant", new)
```

```
for predicate in newQueryStack:
    for index, param in enumerate(predicate.params): if param.name in
      substitution:
      new = substitution[param.name] predicate.params[index].unify("Variable" if new[0].islower()
      else
                    for predicate in newSentence.predicates:
                      newQueryStack.append(predicate)

                  new_visited = copy.deepcopy(visited)
                    if kb_sentence.containsVariable() and len(kb_sentence.predicates) > 1:
                      new_visited[kb_sentence.sentence_index] = True


                    if self.resolve(newQueryStack, new_visited, depth + 1):
                      return True
        return False
    return True
  def performUnification(queryPredicate, kbPredicate):
    substitution = {}
    if queryPredicate == kbPredicate:
      return True, {}

  else:

      for query, kb in zip(queryPredicate.params, kbPredicate.params):
        if query == kb:
        continue
      if kb.isConstant():

        if not query.isConstant():

            if query.name not in substitution:
              substitution[query.name] = kb.name
            elif substitution[query.name] != kb.name:
              return False, {}
          query.unify("Constant", kb.name)
        else:
          return False, {}
      else:
        if not query.isConstant():
            if kb.name not in substitution:
              substitution[kb.name] = query.name
            elif substitution[kb.name] != query.name:
              return False, {}
          kb.unify("Variable", query.name)
        else:
            if kb.name not in substitution:
              substitution[kb.name] = query.name
            elif substitution[kb.name] != query.name:
              return False, {}
```

```
    return True, substitution

def negatePredicate(predicate):

    return predicate[1:] if predicate[0] == "~" else "~" + predicate

def negateAntecedent(sentence):

    antecedent = sentence[:sentence.find("=>")]
    premise = []


    for predicate in antecedent.split("&"):

        premise.append(negatePredicate(predicate))


    premise.append(sentence[sentence.find("=>") + 2:])
    return "|".join(premise)

def getInput(filename):

        with open(filename, "r") as file:
            noOfQueries = int(file.readline().strip())
        inputQueries = [file.readline().strip() for _ in range(noOfQueries)]
        noOfSentences = int(file.readline().strip())
        inputSentences = [file.readline().strip() for _ in range(noOfSentences)]
        return inputQueries, inputSentences

def printOutput(filename, results):

    print(results)

        with open(filename, "w") as file:
            for line in results:
            file.write(line)
            file.write("\n")

    file.close()

if __name__ == '__main__':

    inputQueries_, inputSentences_ = getInput("RA1911003010307/Exp-7/resolution_input.txt")
    knowledgeBase = KB(inputSentences_)
    knowledgeBase.prepareKB()
    results_ = knowledgeBase.askQueries(inputQueries_)
    printOutput("RA1911003010307/Exp-7/resolution_output.txt", results_)
```

Input:

```
2
~Alive(Alice)
Alive(Bob)
2
Sick(x) => Alive(x)
~Sick(x) => Alive(x)
```

Output:

```
['FALSE', 'FALSE']
```

resolution_output.txt

```
FALSE
FALSE
```

SRM INSTITUTE OF SCIENCE & TECHNOLOGY

DEPARTMENT OF NETWORKING & COMMUNICATIONS

18CSC305J-ARTIFICIAL INTELLIGENCE

SEMESTER – 6 BATCH-2

REGISTRATION NUMBER	**RA1911003010750**
NAME	**Akash Kumar Singh**

B.Tech- CSE / CC, Third Year (Section: C2)
Year 2021-2022 / Even Semester

INDEX

Ex No	DATE	Title	Page No	Marks
1	21/03/22	**Uncertain methods (Fuzzy logic/ Dempster Shafer Theory)**		

Tool: VS Code and Python 3.9.0

EXPERIMENT - 7

Aim:

Implementation of uncertain methods for an application (Fuzzy logic/ Dempster Shafer Theory)

Code:

```
#include <iostream>
#include <cmath>
#include <cstring>


const double cdMinimumPrice =0; const
double cdMaximumPrice =70;
using namespace std; class
CFuzzyFunction
{

protected :

   double dLeft, dRight;
   char cType;
   char* sName;


public:

   CFuzzyFunction(){};

   virtual ~CFuzzyFunction(){ delete [] sName; sName=NULL;}



   virtual void
   setInterval(double l,
         double r)

   {dLeft=l; dRight=r;}

         virtual void setMiddle(
   double dL=0,
         double dR=0)=0;
   virtual void
   setType(char c)
   { cType=c;}
```

```
  virtual void setName(const
  char* s){
   sName = new char[strlen(s)+1];
   strcpy(sName,s);

  }


  bool isDotInInterval(double t){

        if((t>=dLeft)&&(t<=dRight)) return true; else return false;

  }

  char getType(void)const{ return cType;}
  Void getName() const{

        cout<<sName<<endl;

  }

  virtual double getValue(double t)=0;

};

class CTriangle : public CFuzzyFunction{

private:

  double dMiddle;

      public:

        void

  setMiddle(double dL, double dR){
        dMiddle=dL;

  }

  double getValue(double t)
  {
        if(t<=dLeft)
              return 0; else
        if(t<dMiddle)
              return (t-dLeft)/(dMiddle-dLeft);
        else if(t==dMiddle)
              return 1.0; else
        if(t<dRight)
              else
  }
};
```

```
return (dRight-t)/(dRight-dMiddle); return 0;


class CTrapezoid : public CFuzzyFunction{
private:

   double dLeftMiddle, dRightMiddle;

Public:
void

   setMiddle(double dL, double dR){

         dLeftMiddle=dL; dRightMiddle=dR;

   }

   double getValue(double
   t)
   {
         if(t<=dLeft)
         return 0;

         else if(t<dLeftMiddle)

               return (t-dLeft)/(dLeftMiddle-dLeft);
         else if(t<=dRightMiddle)
               return 1.0; else
         if(t<dRight)

               return (dRight-t)/(dRight-dRightMiddle);

   }

};

return 0;

int main(void)
{
   CFuzzyFunction *FuzzySet[3];
   FuzzySet[0] = new CTrapezoid;
   FuzzySet[1] = new CTriangle;
   FuzzySet[2] = new CTrapezoid;
   FuzzySet[0]->setInterval(-5,30);
   FuzzySet[0]->setMiddle(0,20);
   FuzzySet[0]->setType('r');
   FuzzySet[0]->setName("low_price");
   FuzzySet[1]->setInterval(25,45);
   FuzzySet[1]->setMiddle(35,35);
   FuzzySet[1]->setType('t');
   FuzzySet[1]->setName("good_price");
```

```
  FuzzySet[2]->setInterval(40,75);
  FuzzySet[2]->setMiddle(50,70);
  FuzzySet[2]->setType('r');
  FuzzySet[2]->setName("to_expensive");
  double dValue;
  do{

   cout<<"\nImput the value->"; cin>>dValue;

   if(dValue<cdMinimumPrice) continue;
   if(dValue>cdMaximumPrice) continue;
  for(int i=0; i<3; i++){

        cout<<"\nThe dot="<<dValue<<endl;
        if(FuzzySet[i]->isDotInInterval(dValue))
              cout<<"In the interval";
        Else
            cout<<"Not in the interval";

        cout<<endl;

        cout<<"The name of function is"<<endl;
        FuzzySet[i]->getName();
        cout<<"and the membership is=";
        cout<<FuzzySet[i]->getValue(dValue);

   }

   cout<<endl;

  }

  while(true);

  return EXIT_SUCCESS;
}
```

Output:

```
Running /home/ubuntu/environment/RA1911003010307/Exp-6/fuzztlogic.cpp
Running /home/ubuntu/environment/RA1911003010307/Exp-6/fuzztlogic.cpp

Imput the value->10

The dot=10
In the interval
The name of function is
low_price
and the membership is=1
The dot=10
Not in the interval
The name of function is
good_price
and the membership is=0
The dot=10
Not in the interval
The name of function is
to_expensive
and the membership is=0

Imput the value->
```

EXPERIMENT-8
Implementation of learning algorithms for an application

NAME: AKASH KUMAR SINGH
REG NO: RA1911003010750

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:

```
#!/usr/bin/env python
# coding: utf-8

# # Student Test Scores Analysis - Linear Regression

# The following analysis is performed on a generated
data structure of hypothetical student test scores and
the student's various characteristics.
#
# The analysis includes an exploratory data analysis,
regression modeling, and assumption checks.
#
# Much of the modeling code was sourced from the
following article:
https://towardsdatascience.com/verifying-the-
assumptions-of-linear-regression-in-python-and-r-
f4cd2907d4c0

# **Load Libraries**

# In[33]:


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


# # Environment Set Up

# **Load Data**

# In[2]:
```

```
df = pd.read_csv('../input/students-performance-in-
exams/StudentsPerformance.csv')


# **Data Overview**

# In[3]:


df.shape


# In[4]:


df.columns


# In[5]:


df.info()


# In[6]:


df.describe()


# In[7]:


df.head()
```

```
# # Data Cleaning

# **Check for nulls**

# In[8]:


df.isnull().sum()


# Awesome, no nulls

# **Rename Columns**

# In[9]:


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)


# **Create a total_score column**

# In[10]:
```

```
df['total_score'] = df['math_score'] +
df['reading_score'] + df['writing_score']


# Check the column now

# In[11]:


df.columns


# # Data Exploration - Visualization

# **Load Libaries**

# In[12]:


import seaborn as sns
import matplotlib.pyplot as plt


# **Variable Distributions**

# In[13]:


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')


# In[14]:


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')


# In[15]:


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')
```

```
# In[16]:


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')


# In[17]:


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')


# In[18]:


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()

# In[19]:

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()

# In[20]:


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()

# # Modeling - Linear Regression

# **Setting up**

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']

# **Model**

# In[23]:

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()

# # Checking Assumptions

# In[24]:
```

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

# **Assumption Check: Linearity**

# Y (response variable) is assumed to be a linear
function of the features. We will inspect plots of
observed vs predicted values AND residuals vs predicted
values. We hope to find a linear plot for the former
and a horizontal line for the latter.

# In[25]:


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)
```

```
# Looks good! Linearity assumption is satisfied.

# **Assumption Check: Mean of residuals is zero**

# In[26]:

lin_reg.resid.mean()

# Good, very small.
```

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	0.1

1 rows × 33 columns

Code:

```
#!/usr/bin/env python
# coding: utf-8

# ## Breast Cancer Predictions using SVM
#
# **NOTE:** *This is a submission for subject
"ARTIFICIAL INTELLIGENCE Lab - Ex. 8b : SVM". I have
used an [exisiting copy of a
notebook](https://www.kaggle.com/code/teampark/breast-
cancer-prediction-using-svm-98-5) for conducting
experiment.*

# In[ ]:


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')


# In[ ]:


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

```
    """
    This function prints and plots the confusion
matrix.
    Normalization can be applied by setting
`normalize=True`.
    """
    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')


# In[ ]:
```

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


# Scale the data to chart it and allow better
predictive power

# In[ ]:


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


# Check for correlations between variables and
diagnosis

# In[ ]:


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)


# Let's see how each variable breaks down by diagnosis
```

```
# In[ ]:


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);

# Let's build a SVM to predict malignant or benign
tumors

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)

# Model scores very well.  Very slight over-fitting.

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='C
onfusion 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:

```
#!/usr/bin/env python
# coding: utf-8

#
#
#

# **Let's see how data looks like.**

# In[ ]:


df.head()


# **I see an ID column here, I'll drop it right away
because the ID is just a unique number that identifies
each row, not a feature therefore ID column is
meaningless in this analysis.**

# In[ ]:
```

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


# In[ ]:


df.shape


# **Our dataset consists of 200 rows and 4 columns.**

# In[ ]:


df.info()


# **There is no null data, it's good ✓**
#
# **Dtype of Gender --> Object, therefore I am going to convert it to numeric.**

# In[ ]:


df.isnull().sum()


# **Also in this way we can see that there is no null data.**

# In[ ]:
```

```
df.describe()


# **We saw multiple values of data with the .describe(
) method. What caught my attention here was the height
of the standard deviations. Because in the column whose
average is 50, about half of it, that is 25 standard
deviations, there is also the same situation in the
annual income. The Age column also has a standard
deviation of about 33% compared to the mean.**

# **Let's look at the correlations.**

# In[ ]:


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()


# **It is clear from this correlation table that older
customers have less income and therefore spend less
money.**

# In[ ]:


# -Distribution Plots-

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()


# **The distributions are generally similar to the
normal distribution, with only a few standard
deviations. The 'more normal' distribution among the
distributions is the 'Spending Score'. That's good
because it's our target column.**

# In[ ]:


# 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'])


# **As you can see, we converted the 'Gender' column to
numeric using the 'Label Encoder'.**
#
# **Male --> 1 , Female -->0**

# In[ ]:


# Let's look at the current state of our Data Frame.
df.head()


# **All the columns have become numeric, good.**

# In[ ]:


# Let's calculate how much to shop for which 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}')


# In[ ]:


# Let's try to understand the relationship between
gender and spending score.

# Number of genders

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()


# **What do we understand from these 3 graphs?**
#
# **There is no significant difference in the mean spending scores of males and females. Since the mean spending scores are very close to each other, the difference between the total spending scores is the difference between the number of male and female customers, but this difference is not
```

serious. Considering all this, it would be meaningless to choose a gender-based target audience. ✓**

```
# In[ ]:


# Let's look at the relationship between Age and
Spending score

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')


# ![supermarket mall my
picture.PNG](attachment:0685e60d-32f6-4a66-9608-
5319e6b5fac7.PNG)
#
# **People between the ages of 20-40 have made more
purchases, considering the inference we just made about
women, we can make our target audience more specific.**

# In[ ]:


# Let's look at the relationship between Annual Income
and Spending Score

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')
```

```
# ![supermarker mall my
figure2.PNG](attachment:2f02dba8-788c-4540-a217-
f616c745e98e.PNG)
#
# **One of the two regions shown can be selected as the
target audience. Even though the number of people whose
annual income is between (40-60)k$ is higher (we
understand this from the number of data points), the
number of that audience is higher but the spending
score is low, so if we make shopping attractive for
them by choosing the target audience from the two
regions above, we will see more profit can be made.**
#

# <a id = "3"></a><br> <p style = "font-size : 45px;
color :#190033 ; font-family : 'Comic Sans MS'; text-
align : center; background-color : #66FFB2; border-
radius: 5px 5px;"><strong>CLUSTERINGS</strong></p>

# <a id = "4"></a><br> <p style = "font-size : 30px;
color :#0009B7 ; font-family : 'Comic Sans MS'; text-
align : center; background-color : #F7CF2D; border-
radius: 5px 5px;">Clustering (4 Variables)</p>

# # PCA
# **Why PCA?**
# <a id = "5"></a><br>
# **Since we will be doing clustering with 4 variables,
I will reduce the size, because after the clustering, I
may have trouble in 4D visualization while visualizing.
I will do the dimension reduction with PCA, PCA works
like this: for example we have a dataset with 4
columns, we want to make it as many columns as we want,
I will reduce it to 2 columns.**
```

```
# In[ ]:


# x assignment
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,:]}')


# **As you can see, X data, which we defined as 4
dimensional (red part), has now been reduced to 2
dimensions (green part) thanks to PCA.**

# In[ ]:


# 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()


# **'4' is optimum number of clusters. Because the most
break in the chart is at that point. This is how we
will select the next optimal n_clusters.**

# In[ ]:


# KMeans
kmeans = KMeans(n_clusters = 4, init ='k-means++',
random_state=88)
y_kmeans = kmeans.fit_predict(X_2D)


# In[ ]:


# clusters visualization
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()


# <a id = "6"></a><br> <p style = "font-size : 30px;
color :#0009B7 ; font-family : 'Comic Sans MS'; text-
align : center; background-color : #F7CF2D; border-
radius: 5px 5px;">Clustering (Age & Annual Income &
Spending Score)

# In[ ]:


# x assignment
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


# In[ ]:


# 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()


# **'6' is optimum number of clusters**

# In[ ]:


# KMeans
kmeans = KMeans(n_clusters = 6, init ='k-means++',
random_state=88)
clusters = kmeans.fit_predict(x_df)
x_df['label'] = clusters


# In[ ]:


# # clusters visualization
fig = px.scatter_3d(data_frame=x_df, x='Age', y='Annual
Income (k$)', z='Spending Score (1-100)',color =
'label', size = 'label')
fig.show()
```

```
# **3D visualization was used as there were 3
variables.**

# <a id = "7"></a><br> <p style = "font-size : 30px;
color :#0009B7 ; font-family : 'Comic Sans MS'; text-
align : center; background-color : #F7CF2D; border-
radius: 5px 5px;">Clustering (Age & Annual Income)</p>

# In[ ]:


# x assignment
x = df[['Age','Annual Income (k$)']].values


# In[ ]:


# 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()


# **'2' is optimum number of clusters.**

# In[ ]:


# KMeans
kmeans = KMeans(n_clusters = 2, init ='k-means++',
random_state=88)
y_kmeans = kmeans.fit_predict(x)


# In[ ]:


# 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()


# <a id = "8"></a><br> <p style = "font-size : 30px;
color :#0009B7 ; font-family : 'Comic Sans MS'; text-
align : center; background-color : #F7CF2D; border-
```

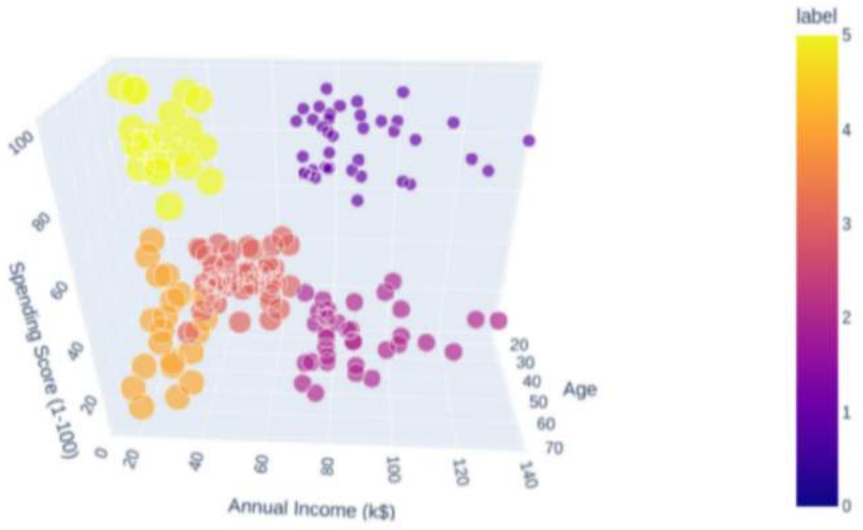
```
radius: 5px 5px;">Clustering (Annual Income & Spending
Score)</p>

# In[ ]:


# x assignment
x = df[['Annual Income (k$)','Spending Score (1-
100)']].values


# In[ ]:


# 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()


# **'5' is optimum number of clusters.**

# In[ ]:
```

```
kmeans = KMeans(n_clusters = 5, init ='k-means++',
random_state=88)
y_kmeans = kmeans.fit_predict(x)


# In[ ]:


# 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()


# <a id = "9"></a><br> <p style = "font-size : 30px;
color :#0009B7 ; font-family : 'Comic Sans MS'; text-
align : center; background-color : #F7CF2D; border-
radius: 5px 5px;">Clustering (Age & Spending Score)</p>
```

```
# In[ ]:


# x assignment
x = df[['Age','Spending Score (1-100)']].values


# In[ ]:


# 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()


# **'4' is optimum number of clusters.**

# In[ ]:


# KMeans
```

```
kmeans = KMeans(n_clusters = 4, init ='k-means++',
random_state=88)
y_kmeans = kmeans.fit_predict(x)


# In[ ]:


# clusters visualization
plt.figure(1 , figsize = (16 ,9))
plt.scatter(x[y_kmeans == 0, 0], x[y_kmeans == 0, 1], s
= 80, c = '#00FF00', label = 'Cluster-1')
plt.scatter(x[y_kmeans == 1, 0], x[y_kmeans == 1, 1], s
= 80, c = '#00FFFF', label = 'Cluster-2')
plt.scatter(x[y_kmeans == 2, 0], x[y_kmeans == 2, 1], s
= 80, c = '#FF00FF', label = 'Cluster-3')
plt.scatter(x[y_kmeans == 3, 0], x[y_kmeans == 3, 1], s
= 80, c = '#FF4500', 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('Age', 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:

```
#!/usr/bin/env python
# coding: utf-8

# In[ ]:


#this is an open-source association rule .py sample for
the dataset Online_Retail
```

```
#for more information on association rules follow
https://bobrupakroy.medium.com/association-rule-
learning-9e3d9caf1f28

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


# In[ ]:


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


# In[ ]:


# Exploring the columns of the data
data.columns


# In[ ]:


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


# In[ ]:
```

```
# 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')]


# In[ ]:


# 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'))


# In[ ]:


# 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



# In[ ]:


# 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


# In[ ]:
```

```
#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())


# In[ ]:


#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())


# In[ ]:


#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())


# In[ ]:


#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 sales dataset.

NAME: AKASH KUMAR SINGH

ROLL NO: RA1911003010750

18CSC305J- ARTIFICIAL INTELLEGENCE

Experiment-9

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.

Experiment-10
Applying deep learning methods to solve an application

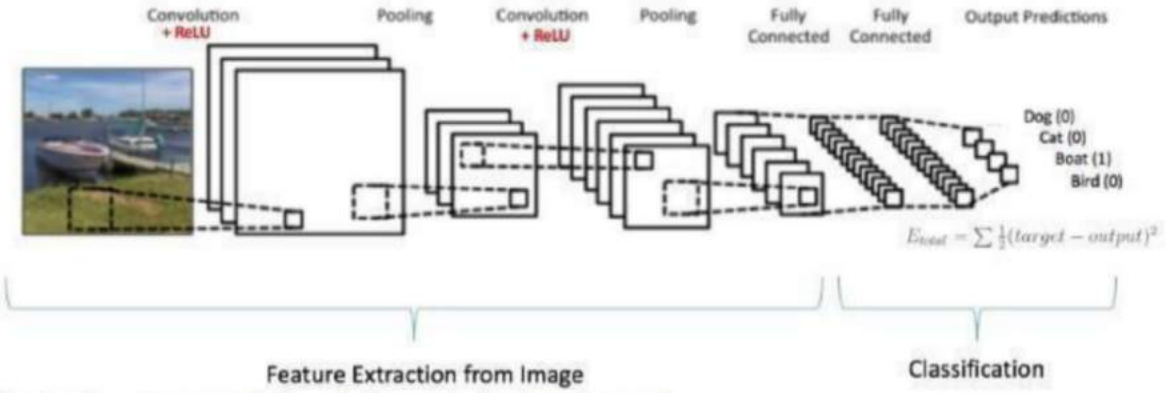
NAME: AKASH KUMAR SINGH
REG NO: RA1911003010750

AIM:

To implement deep learning methods to solve an application

ALGORITHM:

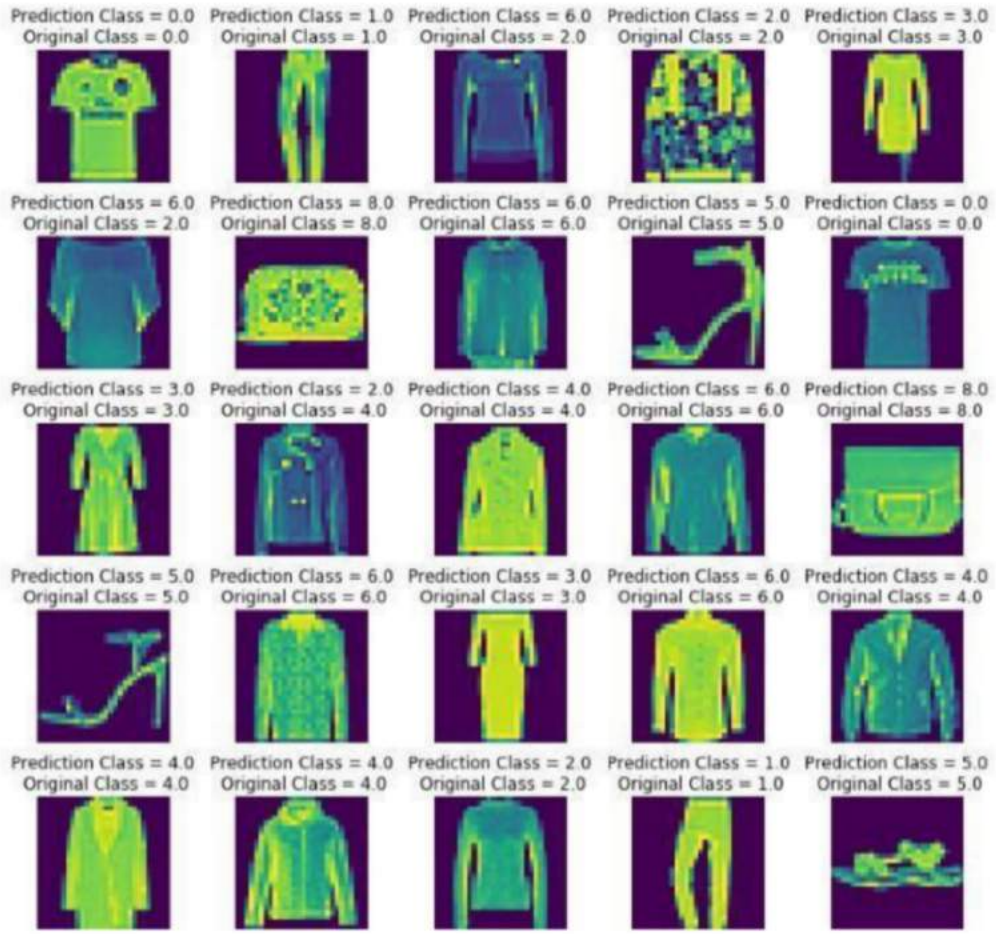
Every image is an cumulative arrangement of dots (a pixel) arranged in a special order. If you change the order or color of a pixel, the image would change as well.



Three basic components to define a basic convolutional neural network.

OUTPUT:

```
fig, axes = plt.subplots(L, W, figsize = (12,12))
axes = axes.ravel()

for i in np.arange(0, L * W):
    axes[i].imshow(x_test[i].reshape(28,28))
    axes[i].set_title(f"Prediction Class = {predicted_classes[i]:0.1f}\n Original Class = {y_tes
t[i]:0.1f}")
    axes[i].axis('off')

plt.subplots_adjust(wspace=0.5)
```



RESULT: Implementation of deep learning methods successfully completed using python.