Date: **Ex No:** **5.01.2022**	**Title of the Lab** **Toy Program Implementation (Vacuum Cleaner Problem)**	**Name: MAINAK CHAUDHURI** **Registration Number: RA1911027010039** **Section: N1** **Lab Batch: 2** **Day Order: 3**

AIM: **To implement a toy program for vacuum cleaner working**

Description of the Concept or Problem given:

Here, states 1 and 2 are our initial states and state 7 and state 8 are our final states (goal states). This means that, initially, both the rooms are full of dirt and the vacuum cleaner can reside in any room. And to reach the final goal state, both the rooms should be clean and the vacuum cleaner again can reside in any of the two rooms.

The vacuum cleaner can perform the following functions: move left, move right, move forward, move backward and to suck dust. But as there are only two rooms in our problem, the vacuum cleaner performs only the following functions here: move left, move right and suck.

Manual Solution:

1. Select a room out of Room A and Room B

2. If there is dirt in Room A, suck the dirt

3. If there is no dirt, move to the next room

4. If there is dirt in Room B, suck the dirt.

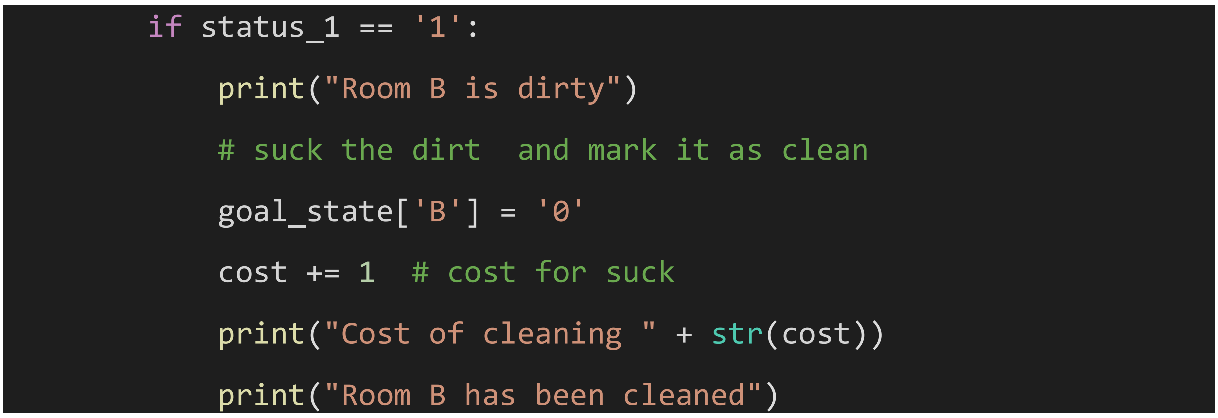
5. If both rooms are clean, stop the process

6. If the other room is dirty, then move the vacuum cleaner to the room and clean the dirt

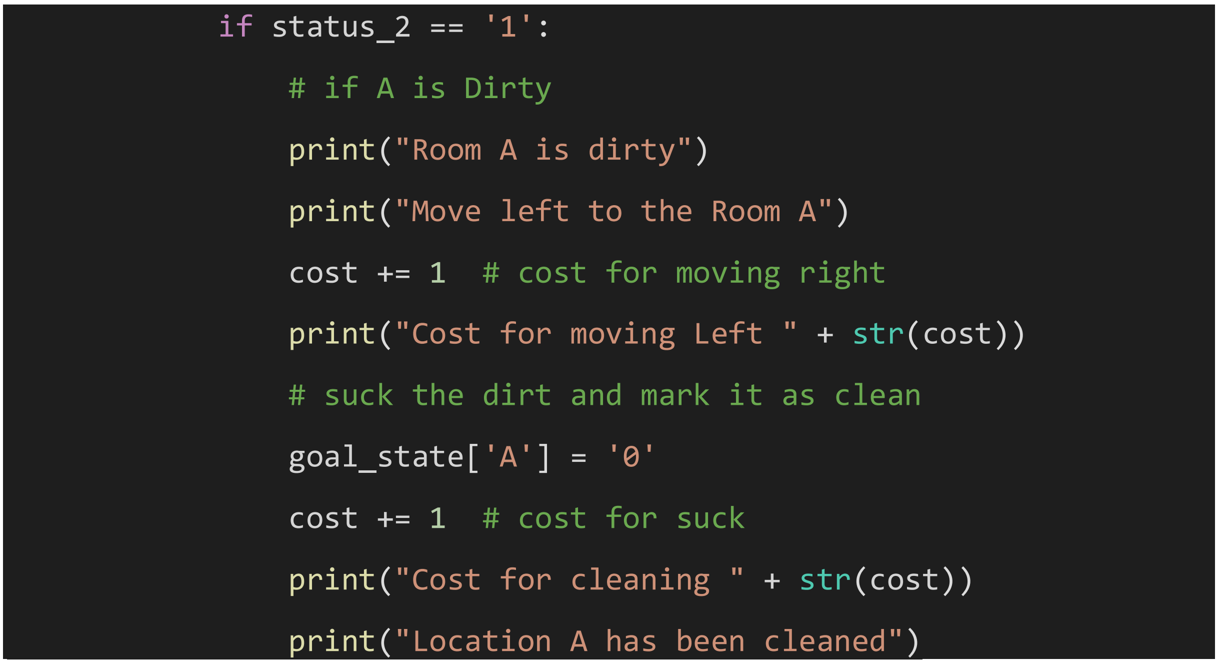
7. Check if both rooms are clean before stopping.

Program Implementation [Coding]

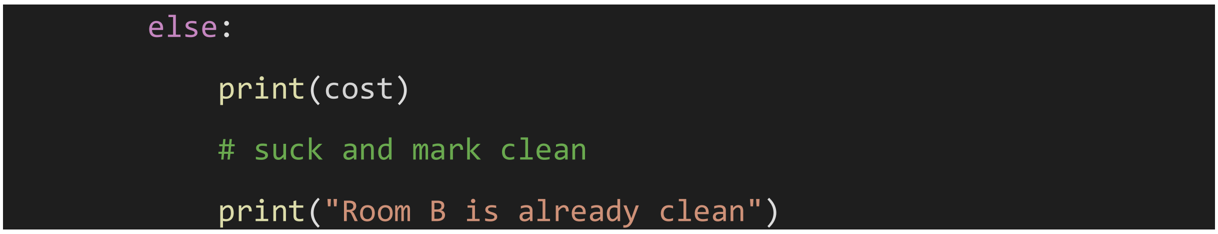
```python
# Room Cleaning Problem using AI


def room_cleaner():
    goal_state = {'A':'0','B':'0'}  # The final state of the rooms after cleaning.
    cost = 0                         # The cost of an operation with the vacuum cleaner


    loc_input = input("Enter the vacuum cleaner\'s location ")
    status_1 = input("Enter the status of the present room ")
    status_2 = input("Enter the status of the next room ")


    if loc_input == 'A':
        # Room A is dirty
        print("Vacuum is placed in Room A")
        if status_1 == '1':
            print("Room A is dirty")
            # clean the dirt from room
            goal_state['A'] = '0'
            cost += 1
            print("Cost of cleaning Room A " + str(cost))
            print("Room A has been cleaned")


            if status_2 == '1':


                print("Room B is dirty")
                print("Move right to the Room B")
                cost +1     #cost for moving right
                print("Cost for moving right " + str(cost))
                # clean the room
                goal_state['B'] = '0'
                cost += 1                       #cost for suck
                print("Cost of cleaning " + str(cost))
                print("Room B has been cleaned")
            else:
                print("No action was performed " + str(cost))
                # cleaned the room
                print("Room B is already clean")

        if status_1 == '0':
            print("Room A is already clean ")
            if status_2 == '1':# if B is Dirty
                print("Room B is dirty")
                print("Moving right to the Room B")
                cost += 1  #cost for moving right
                print("Cost of moving right " + str(cost))
                # suck the dirt and mark it as clean
                goal_state['B'] = '0'
                cost += 1   #cost for suck
                print("Cost for cleaning" + str(cost))
                print("Room B has been cleaned")
            else:
                print("No action was performed " + str(cost))
                print(cost)
                # cleaned the room
                print("Room B is already clean.")

    else:
        print("Vacuum is placed in Room B")
        # Location B is Dirty.
```



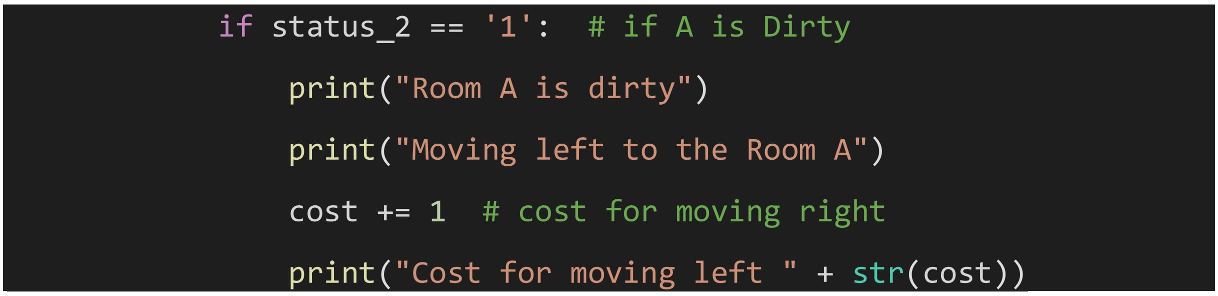
```
        if status_1 == '1':
            print("Room B is dirty")
            # suck the dirt  and mark it as clean
            goal_state['B'] = '0'
            cost += 1  # cost for suck
            print("Cost of cleaning " + str(cost))
            print("Room B has been cleaned")
```



```
            if status_2 == '1':
                # if A is Dirty
                print("Room A is dirty")
                print("Move left to the Room A")
                cost += 1  # cost for moving right
                print("Cost for moving Left " + str(cost))
                # suck the dirt and mark it as clean
                goal_state['A'] = '0'
                cost += 1  # cost for suck
                print("Cost for cleaning " + str(cost))
                print("Location A has been cleaned")
```



```
        else:
            print(cost)
            # suck and mark clean
            print("Room B is already clean")
```

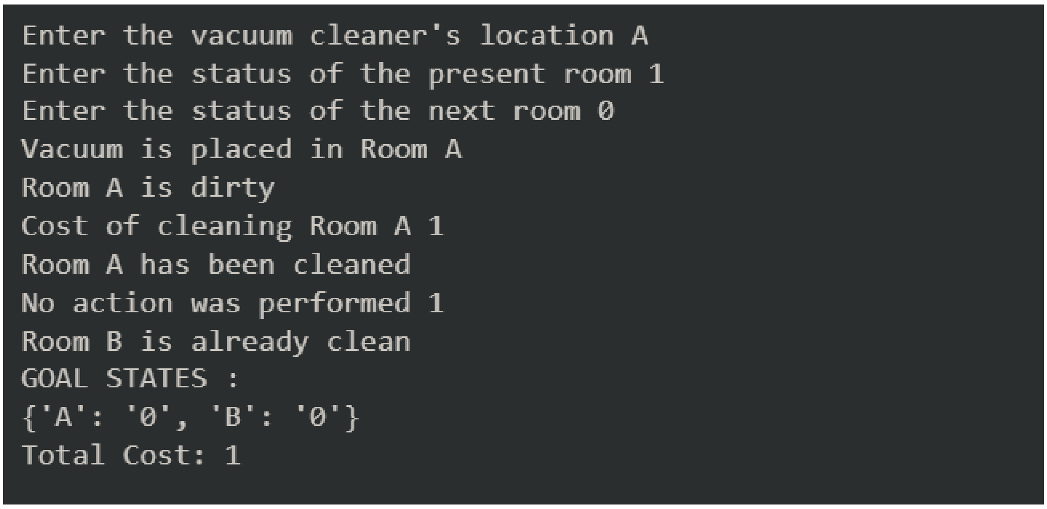


```
            if status_2 == '1':  # if A is Dirty
                print("Room A is dirty")
                print("Moving left to the Room A")
                cost += 1  # cost for moving right
                print("Cost for moving left " + str(cost))
                # suck the dirt and mark it as clean
                goal_state['A'] = '0'
                cost += 1  # cost for suck
                print("Cost for cleaning " + str(cost))
                print("Location A has been cleaned")
            else:
                print("No action is performed " + str(cost))
                # suck and mark clean
                print("Room A is already clean")
```

```
    # done cleaning
    print("GOAL STATES : ")
    print(goal_state)
    print("Total Cost: " + str(cost))
```

```

room_cleaner()
```

Screenshots of the Outputs



```
Enter the vacuum cleaner's location A
Enter the status of the present room 1
Enter the status of the next room 0
Vacuum is placed in Room A
Room A is dirty
Cost of cleaning Room A 1
Room A has been cleaned
No action was performed 1
Room B is already clean
GOAL STATES :
{'A': '0', 'B': '0'}
Total Cost: 1
```

Mainak Chaudhuri

Signature of the Student

MAINAK CHAUDHURI

Date: Ex No: 12.01.2022	Title of the Lab Agent and Real World Problems	Name: MAINAK CHAUDHURI Registration Number: RA1911027010039 Section: N1 Lab Batch: 2 Day Order: 3

AIM: Wumpus World Problem,

Description of the Concept or Problem given:

The Wumpus world is a cave which has 4/4 rooms connected with passageways. So there are total 16 rooms which are connected with each other. We have a knowledge-based agent who will go forward in this world. The cave has a room with a beast which is called Wumpus, who eats anyone who enters the room. The Wumpus can be shot by the agent, but the agent has a single arrow. In the Wumpus world, there are some Pits rooms which are bottomless, and if agent falls in Pits, then he will be stuck there forever. The exciting thing with this cave is that in one room there is a possibility of finding a heap of gold. So the agent goal is to find the gold and climb out the cave without fallen into Pits or eaten by Wumpus. The agent will get a reward if he comes out with gold, and he will get a penalty if eaten by Wumpus or falls in the pit.

Manual Solution

Agent's First step:

Initially, the agent is in the first room or on the square [1,1], and we already know that this room is safe for the agent, so to represent on the below diagram (a) that room is safe we will add symbol OK. Symbol A is used to represent agent, symbol B for the breeze, G for Glitter or gold, V for the visited room, P for pits, W for Wumpus. At Room [1,1] agent does not feel any breeze or any Stench which means the adjacent squares are also OK.

Agent's second Step:

Now agent needs to move forward, so it will either move to [1, 2], or [2,1]. Let's suppose agent moves to the room [2, 1], at this room agent perceives some breeze which means Pit is around this room. The pit can be in [3, 1], or [2,2], so we will add symbol P? to say that, is this Pit room? Now agent will stop and think and will not make any harmful move. The agent will go back to the [1, 1] room. The room [1,1], and [2,1] are visited by the agent, so we will use symbol V to represent the visited squares.

Agent's third step:

At the third step, now agent will move to the room [1,2] which is OK. In the room [1,2] agent perceives a stench which means there must be a Wumpus nearby. But Wumpus cannot be in the room [1,1] as by rules of the game, and also not in [2,2] (Agent had not detected any stench when he was at [2,1]). Therefore agent infers that Wumpus is in the room [1,3], and in current state, there is no breeze which means in [2,2] there is no Pit and no Wumpus. So it is safe, and we will mark it OK, and the agent moves further in [2,2].

Agent's fourth step:

At room [2,2], here no stench and no breezes present so let's suppose agent decides to move to [2,3]. At room [2,3] agent perceives glitter, so it should grab the gold and climb out of the cave.

Program Implementation [Coding]

```
#Download and extract all necessary files
!rm -rf /content/*
!wget https://github.com/aimacode/aima-python/archive/master.zip 2>/dev/null
!unzip -q master.zip
!mv aima-python-master/* /content
!wget https://github.com/aimacode/aima-data/archive/f6cbea61ad0c21c6b7be826d17af5a8d3a7c2c86.zip 2>/dev/null
!unzip -q f6cbea61ad0c21c6b7be826d17af5a8d3a7c2c86.zip
!rm -rf aima-data
!mv aima-data-f6cbea61ad0c21c6b7be826d17af5a8d3a7c2c86 aima-data
#Install Libraries
!pip install ipythonblocks 2>/dev/null

from agents import *

class BlindDog(Agent):
    def eat(self, thing):
        print("Dog: Ate food at {}.".format(self.location))

    def drink(self, thing):
        print("Dog: Drank water at {}.".format( self.location))

dog = BlindDog()
class Food(Thing):
    pass

class Water(Thing):
    pass

class Park(Environment):
    def percept(self, agent):
        '''return a list of things that are in our agent's location'''
        things = self.list_things_at(agent.location)
        return things

    def execute_action(self, agent, action):
        '''changes the state of the environment based on what the agent does.'''
        if action == "move down":
            print('{} decided to {} at location: {}'.format(str(agent)[1:-1], action, agent.location))
            agent.movedown()
        elif action == "eat":
            items = self.list_things_at(agent.location, tclass=Food)
            if len(items) != 0:
                if agent.eat(items[0]): #Have the dog eat the first item
                    print('{} ate {} at location: {}'
                          .format(str(agent)[1:-1], str(items[0])[1:-1], agent.location))
                    self.delete_thing(items[0]) #Delete it from the Park after.
        elif action == "drink":
            items = self.list_things_at(agent.location, tclass=Water)
            if len(items) != 0:
                if agent.drink(items[0]): #Have the dog drink the first item
                    print('{} drank {} at location: {}'
                          .format(str(agent)[1:-1], str(items[0])[1:-1], agent.location))
                    self.delete_thing(items[0]) #Delete it from the Park after.

    def is_done(self):
        '''By default, we're done when we can't find a live agent,
        but to prevent killing our cute dog, we will stop before itself - when there is no more food or water'''
        no_edibles = not any(isinstance(thing, Food) or isinstance(thing, Water) for thing in self.things)
        dead_agents = not any(agent.is_alive() for agent in self.agents)
        return dead_agents or no_edibles
class BlindDog(Agent):
    location = 1

    def movedown(self):
        self.location += 1

    def eat(self, thing):
        '''returns True upon success or False otherwise'''
        if isinstance(thing, Food):
            return True
        return False

    def drink(self, thing):
        ''' returns True upon success or False otherwise'''
        if isinstance(thing, Water):
            return True
        return False

def program(percepts):
    '''Returns an action based on it's percepts'''
    for p in percepts:
        if isinstance(p, Food):
            return 'eat'
        elif isinstance(p, Water):
            return 'drink'
    return 'move down'
    park = Park()
    dog = BlindDog(program)
    dogfood = Food()
    water = Water()
    park.add_thing(dog, 1)
    park.add_thing(dogfood, 5)
    park.add_thing(water, 7)

    park.run(5)
    park.add_thing(water, 15)
    park.run(10)
    class Park2D(XYEnvironment):
        def percept(self, agent):
            '''return a list of things that are in our agent's location'''
            things = self.list_things_at(agent.location)
            return things

        def execute_action(self, agent, action):
            '''changes the state of the environment based on what the agent does.'''
            if action == "move down":
                print('{} decided to {} at location: {}'.format(str(agent)[1:-1], action, agent.location))
                agent.movedown()
            elif action == "eat":
                items = self.list_things_at(agent.location, tclass=Food)
                if len(items) != 0:
                    if agent.eat(items[0]): #Have the dog eat the first item
                        print('{} ate {} at location: {}'
                              .format(str(agent)[1:-1], str(items[0])[1:-1], agent.location))
                        self.delete_thing(items[0]) #Delete it from the Park after.
            elif action == "drink":
                items = self.list_things_at(agent.location, tclass=Water)
                if len(items) != 0:
                    if agent.drink(items[0]): #Have the dog drink the first item
                        print('{} drank {} at location: {}'
                              .format(str(agent)[1:-1], str(items[0])[1:-1], agent.location))
                        self.delete_thing(items[0]) #Delete it from the Park after.


        def is_done(self):
            '''By default, we're done when we can't find a live agent,
            but to prevent killing our cute dog, we will stop before itself - when there is no more food or water'''
            no_edibles = not any(isinstance(thing, Food) or isinstance(thing, Water) for thing in self.things)
            dead_agents = not any(agent.is_alive() for agent in self.agents)
            return dead_agents or no_edibles


    class BlindDog(Agent):
        location = [0,1] # change location to a 2d value
        direction = Direction("down") # variable to store the direction our dog is facing
        def movedown(self):
            self.location[1] += 1

        def eat(self, thing):
            '''returns True upon success or False otherwise'''
            if isinstance(thing, Food):
                return True
            return False

        def drink(self, thing):
            ''' returns True upon success or False otherwise'''
            if isinstance(thing, Water):
                return True
            return False

    def program(percepts):
        '''Returns an action based on it's percepts'''
        for p in percepts:
            if isinstance(p, Food):
                return 'eat'
            elif isinstance(p, Water):
                return 'drink'
        return 'move down'
    park = Park2D(5,20) # park width is set to 5, and height to 20
    dog = BlindDog(program)
    dogfood = Food()
    water = Water()
    park.add_thing(dog, [0,1])
    park.add_thing(dogfood, [0,5])
    park.add_thing(water, [0,7])
    morewater = Water()
    park.add_thing(morewater, [0,15])
    park.run(20)
    from random import choice

    turn = False # global variable to remember to turn if our dog hits the boundary
    class EnergeticBlindDog(Agent):
        location = [0,1]
        direction = Direction("down")

        def moveforward(self, success=True):
            '''moveforward possible only if success (ie valid destination location)'''
            global turn
            if not success:
                turn = True # if edge has been reached, remember to turn
                return
            if self.direction.direction == Direction.R:
                self.location[0] += 1
            elif self.direction.direction == Direction.L:
                self.location[0] -= 1
            elif self.direction.direction == Direction.D:
                self.location[1] += 1
            elif self.direction.direction == Direction.U:
                self.location[1] -= 1

        def turn(self, d):
            self.direction = self.direction + d

        def eat(self, thing):
            '''returns True upon success or False otherwise'''
            if isinstance(thing, Food):
                return True
            return False

        def drink(self, thing):
            ''' returns True upon success or False otherwise'''
            if isinstance(thing, Water):
                return True
            return False

    def program(percepts):
        '''Returns an action based on it's percepts'''
        global turn
        for p in percepts: # first eat or drink - you're a dog!
            if isinstance(p, Food):
                return 'eat'
            elif isinstance(p, Water):
                return 'drink'
        if turn: # then recall if you were at an edge and had to turn
            turn = False
            choice = random.choice((1,2));
        else:
            choice = random.choice((1,2,3,4)) # 1-right, 2-left, others-forward
        if choice == 1:
            return 'turnright'
        elif choice == 2:
            return 'turnleft'
        else:
            return 'moveforward'
```

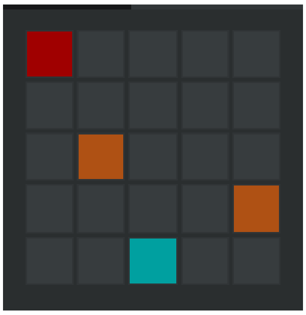
```
class Park2D(XYEnvironment):
def percept(self, agent):
    '''return a list of things that are in our agent's location'''
    things = self.list_things_at(agent.location)
    return things

def execute_action(self, agent, action):
    '''changes the state of the environment based on what the agent does.'''
    if action == 'turnright':
        print('{} decided to {} at location: {}'.format(str(agent)[1:-1], action, agent.location))
        agent.turn(Direction.R)
    elif action == 'turnleft':
        print('{} decided to {} at location: {}'.format(str(agent)[1:-1], action, agent.location))
        agent.turn(Direction.L)
    elif action == 'moveforward':
        loc = copy.deepcopy(agent.location) # find out the target location
        if agent.direction.direction == Direction.R:
            loc[0] += 1
        elif agent.direction.direction == Direction.L:
            loc[0] -= 1
        elif agent.direction.direction == Direction.D:
            loc[1] += 1
        elif agent.direction.direction == Direction.U:
            loc[1] -= 1
        if self.is_inbounds(loc):# move only if the target is a valid location
            print('{} decided to move {}wards at location: {}'.format(str(agent)[1:-1], agent.direction.direction, agent.location))
            agent.moveforward()
        else:
```

```
                    print('{} decided to move {}wards at location: {}, but couldn\'t'.format(str(agent)[1:-1], agent.direction.direction, agent.location))
                    agent.moveforward(False)
            elif action == "eat":
                items = self.list_things_at(agent.location, tclass=Food)
                if len(items) != 0:
                    if agent.eat(items[0]):
                        print('{} ate {} at location: {}'
                              .format(str(agent)[1:-1], str(items[0])[1:-1], agent.location))
                        self.delete_thing(items[0])
            elif action == "drink":
                items = self.list_things_at(agent.location, tclass=Water)
                if len(items) != 0:
                    if agent.drink(items[0]):
                        print('{} drank {} at location: {}'
                              .format(str(agent)[1:-1], str(items[0])[1:-1], agent.location))
                        self.delete_thing(items[0])

        def is_done(self):
            '''By default, we're done when we can't find a live agent,
            but to prevent killing our cute dog, we will stop before itself - when there is no more food or water'''
            no_edibles = not any(isinstance(thing, Food) or isinstance(thing, Water) for thing in self.things)
            dead_agents = not any(agent.is_alive() for agent in self.agents)
            return dead_agents or no_edibles
    park = Park2D(3,3)
    dog = EnergeticBlindDog(program)
    dogfood = Food()
    water = Water()
    park.add_thing(dog, [0,0])
```

```
park.add_thing(dogfood, [1,2])
park.add_thing(water, [2,1])
morewater = Water()
park.add_thing(morewater, [0,2])
print("dog started at [0,0], facing down. Let's see if he found any food or water!")
park.run(20)
class GraphicPark(GraphicEnvironment):
    def percept(self, agent):
        '''return a list of things that are in our agent's location'''
        things = self.list_things_at(agent.location)
        return things

    def execute_action(self, agent, action):
        '''changes the state of the environment based on what the agent does.'''
        if action == 'turnright':
            print('{} decided to {} at location: {}'.format(str(agent)[1:-1], action, agent.location))
            agent.turn(Direction.R)
        elif action == 'turnleft':
            print('{} decided to {} at location: {}'.format(str(agent)[1:-1], action, agent.location))
            agent.turn(Direction.L)
        elif action == 'moveforward':
            loc = copy.deepcopy(agent.location) # find out the target location
            if agent.direction.direction == Direction.R:
                loc[0] += 1
            elif agent.direction.direction == Direction.L:
                loc[0] -= 1
            elif agent.direction.direction == Direction.D:
                loc[1] += 1
            elif agent.direction.direction == Direction.U:
                loc[1] -= 1
            if self.is_inbounds(loc):# move only if the target is a valid location
                print('{} decided to move {}wards at location: {}'.format(str(agent)[1:-1], agent.direction.direction, agent.location))
                agent.moveforward()
            else:
                print('{} decided to move {}wards at location: {}, but couldn\'t'.format(str(agent)[1:-1], agent.direction.direction, agent.location))
                agent.moveforward(False)
        elif action == "eat":
            items = self.list_things_at(agent.location, tclass=Food)
            if len(items) != 0:
                if agent.eat(items[0]):
                    print('{} ate {} at location: {}'
                          .format(str(agent)[1:-1], str(items[0])[1:-1], agent.location))
                    self.delete_thing(items[0])
        elif action == "drink":
            items = self.list_things_at(agent.location, tclass=Water)
            if len(items) != 0:
                if agent.drink(items[0]):
                    print('{} drank {} at location: {}'
                          .format(str(agent)[1:-1], str(items[0])[1:-1], agent.location))
                    self.delete_thing(items[0])


    def is_done(self):
        '''By default, we're done when we can't find a live agent,
        but to prevent killing our cute dog, we will stop before itself - when there is no more food or water'''
        no_edibles = not any(isinstance(thing, Food) or isinstance(thing, Water) for thing in self.things)
        dead_agents = not any(agent.is_alive() for agent in self.agents)
        return dead_agents or no_edibles
```

```
from ipythonblocks import BlockGrid


park = GraphicPark(5,5, color={'EnergeticBlindDog': (200,0,0), 'Water': (0, 200, 200),
'Food': (230, 115, 40)})
dog = EnergeticBlindDog(program)
dogfood = Food()
water = Water()
park.add_thing(dog, [0,0])
park.add_thing(dogfood, [1,2])
park.add_thing(water, [0,1])
morewater = Water()
morefood = Food()
park.add_thing(morewater, [2,4])
park.add_thing(morefood, [4,3])
print("dog started at [0,0], facing down. Let's see if he found any food or water!")
park.run(20)


from ipythonblocks import BlockGrid
from agents import *


color = {"Breeze": (225, 225, 225),
        "Pit": (0,0,0),
        "Gold": (253, 208, 23),
        "Glitter": (253, 208, 23),
        "Wumpus": (43, 27, 23),
        "Stench": (128, 128, 128),
        "Explorer": (0, 0, 255),
        "Wall": (44, 53, 57)
        }


def program(percepts):
    '''Returns an action based on it's percepts'''
    print(percepts)
    return input()
w = WumpusEnvironment(program, 7, 7)
grid = BlockGrid(w.width, w.height, fill=(123, 234, 123))

def draw_grid(world):
    global grid
    grid[:] = (123, 234, 123)
    for x in range(0, len(world)):
        for y in range(0, len(world[x])):
            if len(world[x][y]):
                grid[y, x] = color[world[x][y][-1].__class__.__name__]

def step():
    global grid, w
    draw_grid(w.get_world())
    grid.show()
    w.step()
```

Screenshots of the Outputs:



Mainak Chaudhuri

Signature of the Student

MAINAK CHAUHDURI

Date: **Ex No:** 3.1	**Title of the Lab** Cryptarithmetic	**Name:** Mainak Chaudhuri **Registration Number:** RA1911027010039 **Section:** N1 **Lab Batch:** 1 **Day Order:** 3

AIM:

To implement the Cryptarithmetic Problem (CROSS + ROADS = DANGER) problem in python.

Description of the Concept or Problem given:

Cryptarithmetic Problem is a type of constraint satisfaction problem where the game is about digits and its unique replacementeither with alphabets or other symbols. In cryptarithmetic problem, the digits (0-9) get substituted by some possible alphabets or symbols.

The task in cryptarithmetic problem is to substitute each digit withan alphabet to get the result arithmetically correct.

Manual Solution:

We can perform all the arithmetic operations on a givencryptarithmetic problem.

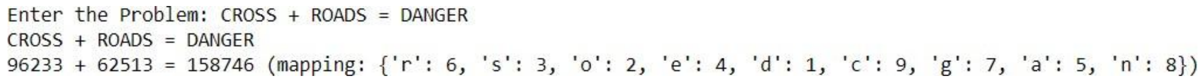
The rules or constraints on a cryptarithmetic problem are asfollows:

- There should be a unique digit to be replaced with a uniquealphabet.
- The result should satisfy the predefined arithmetic rules, i.e.,2+2 =4, nothing else.
- Digits should be from 0-9 only.
- There should be only one carry forward, while performing theaddition operation on a problem.
- The problem can be solved from both sides, i.e., lefthand side(L.H.S), or righthand side (R.H.S).

Program Implementation [Coding]

```
import itertools

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

def solve2(equation):
    left, right = equation.lower().replace(' ', '').split('=')
    left = left.split('+')
    letters = set(right)
    for word in left:
        for letter in word:
            letters.add(letter)
    letters = list(letters)

    digits = range(10)
    for perm in itertools.permutations(digits, len(letters)):
        sol = dict(zip(letters, perm))

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

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

Screenshots of the Outputs:



```
Enter the Problem: CROSS + ROADS = DANGER
CROSS + ROADS = DANGER
96233 + 62513 = 158746 (mapping: {'r': 6, 's': 3, 'o': 2, 'e': 4, 'd': 1, 'c': 9, 'g': 7, 'a': 5, 'n': 8})
```

Signature of the Student

[MAINAK CHAUDHURI]

Date: **Ex No:** 3.1	**Title of the Lab** Graph colouring	**Name:** Mainak Chaudhuri **Registration Number:** RA1911027010039 **Section:** N1 **Lab Batch:** 1 **Day Order:** 3

AIM:

To implement the graph colouring problem in python.

Description of the Concept or Problem given:

Graph colouring is the procedure of assignment of colours to each vertex of a graph G such that no adjacent vertices get same colour.The objective is to minimise the number of colours while colouring agraph. The smallest number of colours required to colour a graph Gis called its chromatic number of that graph.

Manual Solution:

The steps required to colour a graph G with n number of verticesare as follows −

Arrange the vertices of the graph in some order.

Choose the first vertex and colour it with the first colour. Choose the next vertex and colour it with the lowest numbered colour that has not been coloured on any vertices adjacent to it.

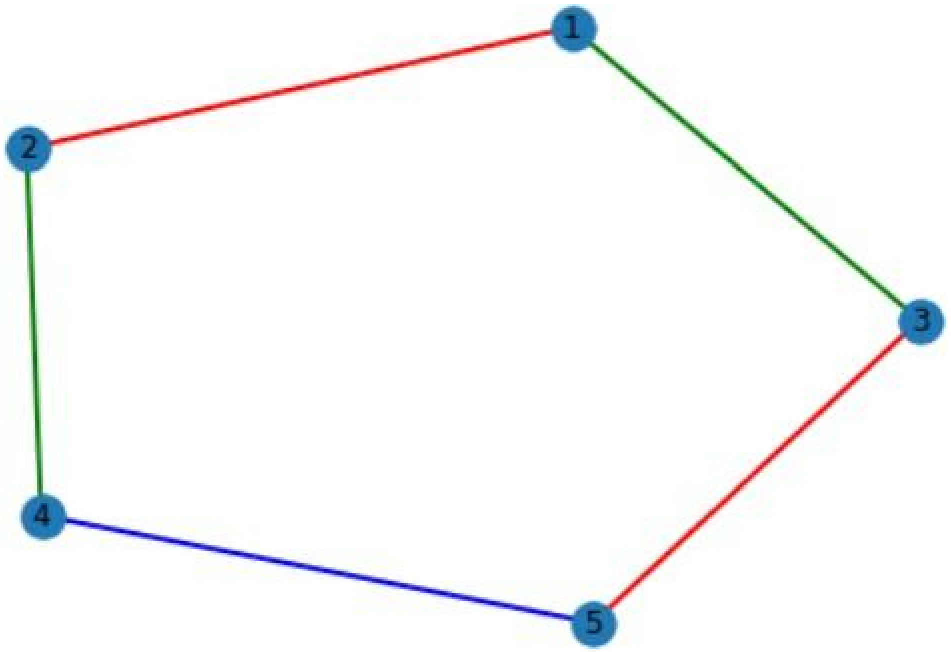
If all the adjacent vertices are coloured with this colour, assign a new colour to it. Repeat this step until all the vertices are coloured.

Program Implementation [Coding]

```
import matplotlib.pyplot as plt
import networkx as nx
from matplotlib.patches import Polygon
import numpy as np
G = nx.Graph()

colors = {0:"red", 1:"green", 2:"blue", 3:"yellow"}
G.add_nodes_from([1,2,3,4,5])
G.add_edges_from([(1,2), (1,3), (2,4), (3,5), (4,5)])
nodes = list(G.nodes)
edges = list(G.edges)
color_lists = []
color_of_edge = []
some_colors = ['red','green','blue','yellow']
for i in range(len(nodes) + 1):
    color_lists.append([])
    color_of_edge.append(-1)
```

```
def getSmallestColor(ls1,ls2):
    i = 1
    while(i in ls1 or i in ls2):
        i = i + 1
    return i

#iterate over edges
i = 0
for ed in edges:
    newColor = getSmallestColor(color_lists[ed[0]],color_lists[ed[1]])
    color_lists[ed[0]].append(newColor)
    color_lists[ed[1]].append(newColor)
    color_of_edge[i] = newColor
    i = i + 1

# Makin graph again G = nx.Graph()
for i in range(len(edges)):
    G.add_edge(edges[i][0],edges[i][1],color=some_colors[color_of_edge[i]-1])
colors = nx.get_edge_attributes(G,'color').values()
nx.draw(G, edge_color=colors, with_labels=True, width=2) plt.show()
```

Screenshots of the Outputs:



Signature of the Student

[MAINAK CHAUDHURI]

Ex. 4	**SOCIAL NETWORKING RECOMMENDATION USING BFS ALGORITHM**	**MAINAK CHAUDHURI RA1911027010039 CSE - BD, N1**

AIM : Finding friends/connections using heuristic breadth first search algorithm.

Breadth First Search, being a level order traversal, would be quite efficient over Depth First Search for many business based insights, like the following:

1. Finding all the friends of all the people in the network
2. Finding all the mutual friends for a node in the network
3. Finding the shortest path between two people in the network
4. Finding the nth level friends for a person in the network

We can find the path between two people by running a BFS algorithm, starting the traversal from one person in level order until we reach the other person or in a much optimised way we can run a bi-directional BFS from both the nodes until our search meet at some point and hence we conclude the path , whereas DFS being a depth wise traversal may run through many unnecessary sub-trees, unknowing of the fact that the friend could be on the first level itself.
Moreover, in order to find friends at nth level, using BFS this could be done in much less time, as this traversal keeps account of all the nodes in each level.

Input Format
The first line of the input denotes the total number of users (n) in the social media network.
space.

Output Format
The output consists of space separated friend-ids which are present at level k from the source friend-id s. If no friend present at level k then print 0

CODE:

```
#social network bfs algorithm
from collections import deque
graph = {}
queries = []
def accept_values():
    #accept number of vertices and edges
    vertex_edge = [int(i) for  i in input().strip().split(" ")]
    #accept the edges for the undirected graph
    for i in range(vertex_edge[1]):
        edge = [int(i) for i in input().strip().split(" ")]
        try:
            graph[edge[0]].append(edge[1])
        except:
            graph[edge[0]] = [edge[1]]
        try:
            graph[edge[1]].append(edge[0])
        except:
            graph[edge[1]] = [edge[0]]
    #accepting the number of queries and the queries themselves
    number_of_queries = int(input())
    for i in range(number_of_queries):
        queries.append([int(i) for i in input().strip().split(" ")])
    #applying bfs on the (source, required distance) queries
    for query in queries:
        bfs(query)

def bfs(query):
    counter = 0
    q = deque()
    q.append(query[0])
    visited = {query[0] : True}
    distance = {query[0] : 0}
    while q:
        popped = q.popleft()
        for neighbour in graph[popped]:
            if neighbour not in visited:
                visited[neighbour] = True
                #stop looking further because we have all the nodes at the required distance,
i think
                if distance[popped] + 1 > query[1]:
                    for key in distance:
                        if distance[key] == query[1]:
                          counter +=1
                    print(counter)
                    return
                #keep going until we reach the required query distance
                else:
                    distance[neighbour] = distance[popped] + 1
                    q.append(neighbour)
    #prints 0 if there are no such nodes
    print(counter)
#calling function to accept values as per required format
accept_values()
```

OUTPUT:

```
0 61 21 32 43 64 5-1 -11 2

4 6
```

EXPERIMENT 5
18CSC305J

Reg No. : RA1911027010039
Name: Mainak Chaudhuri

AIM: To implement Best First Algorithm and A* Algorithm using python.

BEST FIRST SEARCH

Description:
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.

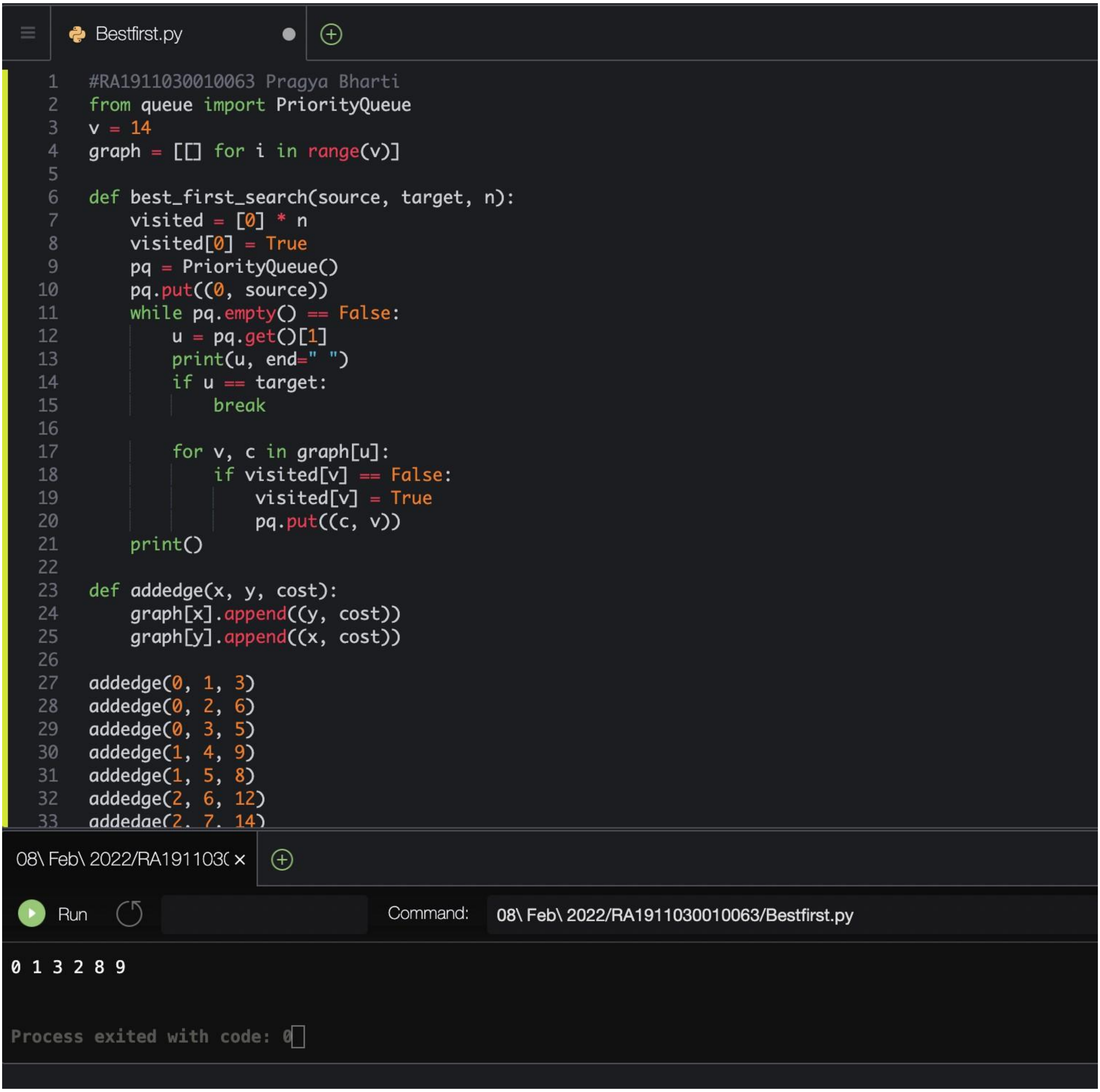
Algorithm:

- Define a list, OPEN, consisting solely of a single node, the start node, *s*.
- IF the list is empty, return failure.
- Remove from the list the node *n* with the best score (the node where *f* is the minimum), and move it to a list, CLOSED.
- Expand node *n*.
- IF any successor to *n* is the goal node, return success and the solution (by tracing the path from the goal node to *s*).
- FOR each successor node: 1.apply the evaluation function, *f*, to the node. 2. IF the node has not been in either list, add it to OPEN.
- looping structure by sending the algorithm back to the second step.

Code:

```
from queue import PriorityQueue
v = 14
graph = [[] for i in range(v)]

def best_first_search(source, target, n):
    visited = [0] * n
    visited[0] = True
    pq = PriorityQueue()
    pq.put((0, source))
    while pq.empty() == False:
        u = pq.get()[1]
        print(u, end=" ")
        if u == target:
            break

        for v, c in graph[u]:
            if visited[v] == False:
                visited[v] = True
                pq.put((c, v))
    print()

def addedge(x, y, cost):
    graph[x].append((y,   cost))
    graph[y].append((x, cost))

addedge(0, 1, 3)
addedge(0, 2, 6)
addedge(0, 3, 5)
addedge(1, 4, 9)
addedge(1, 5, 8)
addedge(2, 6, 12)
addedge(2, 7, 14)
addedge(3, 8, 7)
addedge(8, 9, 5)
addedge(8, 10, 6)
addedge(9, 11, 1)
addedge(9, 12, 10)
addedge(9, 13, 2)

source = 0
target = 9 best_first_search(source,
target, v)
```

Output:



```python
#RA1911030010063 Pragya Bharti
from queue import PriorityQueue
v = 14
graph = [[] for i in range(v)]

def best_first_search(source, target, n):
    visited = [0] * n
    visited[0] = True
    pq = PriorityQueue()
    pq.put((0, source))
    while pq.empty() == False:
        u = pq.get()[1]
        print(u, end=" ")
        if u == target:
            break

        for v, c in graph[u]:
            if visited[v] == False:
                visited[v] = True
                pq.put((c, v))
    print()

def addedge(x, y, cost):
    graph[x].append((y, cost))
    graph[y].append((x, cost))

addedge(0, 1, 3)
addedge(0, 2, 6)
addedge(0, 3, 5)
addedge(1, 4, 9)
addedge(1, 5, 8)
addedge(2, 6, 12)
addedge(2, 7, 14)
```

Command: 08\ Feb\ 2022/RA1911030010063/Bestfirst.py

```
0 1 3 2 8 9


Process exited with code: 0
```

A* Best First Search

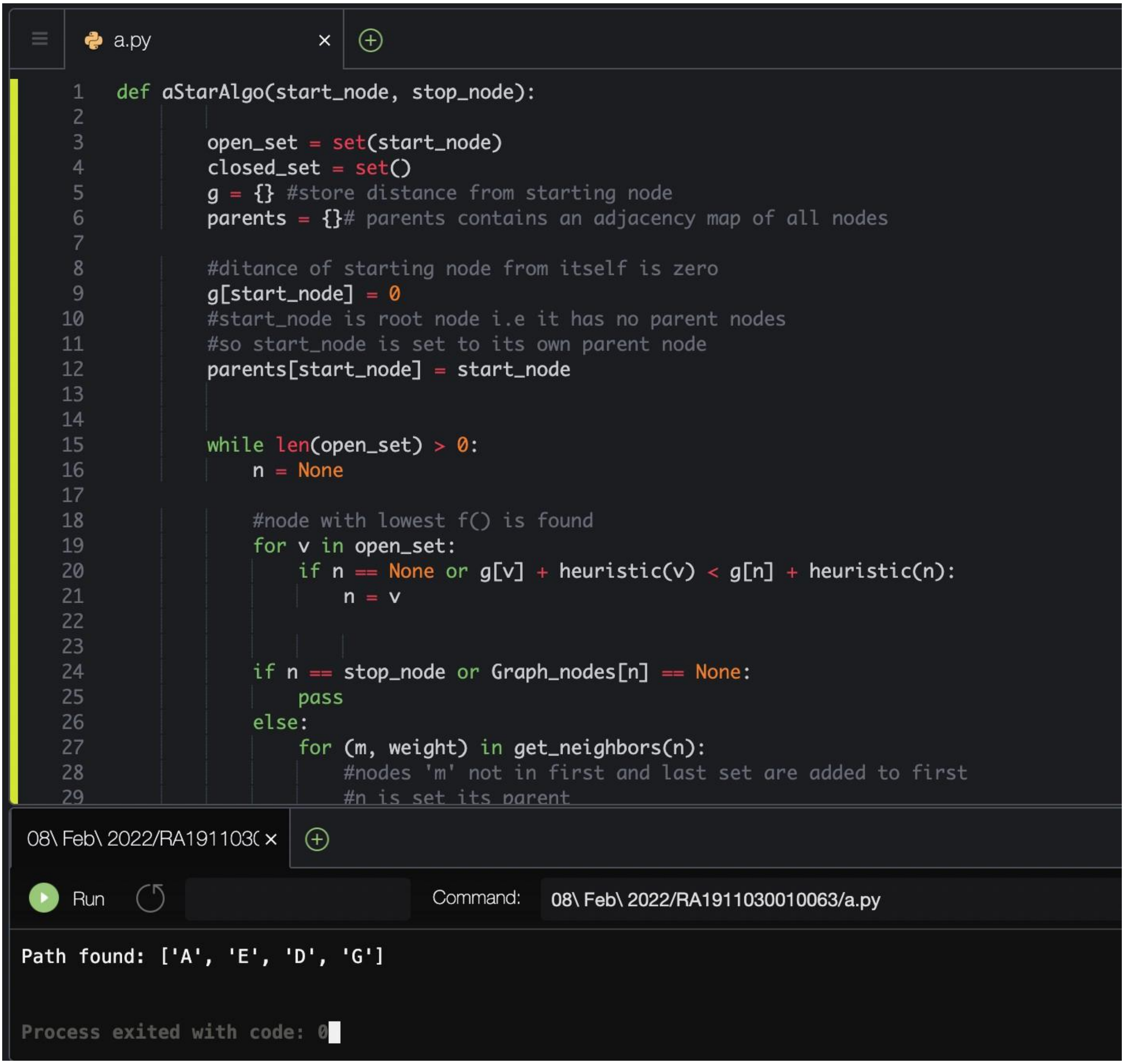
Description:

A* is an informed search algorithm, or a best-first search, meaning that it is formulated in terms of weighted graphs: starting from a specific starting node of a graph, it aims to find a path to the given goal node having the smallest cost (least distance travelled, shortest time, etc.). It does this by maintaining a tree of paths originating at the start node and extending those paths one edge at a time until its termination criterion is satisfied.

Code:

```
def aStarAlgo(start_node, stop_node):

        open_set = set(start_node)
        closed_set = set()
        g = {} #store distance from starting node
        parents = {}# parents contains an adjacency map of all
nodes

        #ditance of starting node from itself is zero
        g[start_node] = 0
        #start_node is root node i.e it has no parent nodes
        #so start_node is set to its own parent node
        parents[start_node] = start_node


        while len(open_set) > 0:
            n = None

            #node with lowest f() is found
            for v in open_set:
                if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):
                    n = v
            if n == stop_node or Graph_nodes[n] == None:
                pass
            else:
                for (m, weight) in get_neighbors(n):
                    #nodes 'm' not in first and last set are added  to
first
                    #n is set its parent
                    if m not in open_set and m not in closed_set:
                        open_set.add(m)
                        parents[m] = n
                        g[m] = g[n] + weight


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

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

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

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

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

                path.append(start_node)

                path.reverse()

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


            # remove n from the open_list, and add it to closed_list
            # because all of his neighbors were inspected
            open_set.remove(n)
            closed_set.add(n)

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

#define fuction to return neighbor and its distance
#from the passed node
def get_neighbors(v):
    if v in Graph_nodes:
        return Graph_nodes[v]
    else:
        return None
#for simplicity we ll consider heuristic distances given
#and this function returns heuristic distance for all nodes
def heuristic(n):
        H_dist =
            { 'A':
            11,
            'B': 6,
            'C': 99,
            'D': 1,
            'E': 7,
            'G': 0,

        }

        return H_dist[n]

#Describe your graph here
Graph_nodes = {
    'A': [('B', 2), ('E', 3)],
    'B': [('C', 1),('G', 9)],
    'C': None,
    'E': [('D', 6)],
    'D': [('G', 1)],

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

Output:



```
def aStarAlgo(start_node, stop_node):

        open_set = set(start_node)
        closed_set = set()
        g = {} #store distance from starting node
        parents = {}# parents contains an adjacency map of all nodes

        #ditance of starting node from itself is zero
        g[start_node] = 0
        #start_node is root node i.e it has no parent nodes
        #so start_node is set to its own parent node
        parents[start_node] = start_node


        while len(open_set) > 0:
            n = None

            #node with lowest f() is found
            for v in open_set:
                if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):
                    n = v


            if n == stop_node or Graph_nodes[n] == None:
                pass
            else:
                for (m, weight) in get_neighbors(n):
                    #nodes 'm' not in first and last set are added to first
                    #n is set its parent
```

```
Path found: ['A', 'E', 'D', 'G']


Process exited with code: 0
```

Result: Best first and A* algorithm were successfully executed in python.

Date:30/03/2022 Ex No: 6	Implementation of fuzzy logic	Name: Mainak Chaudhuri Registration Number: RA1911027010039 Section: N1 Lab Batch: 1 Day Order: 2

AIM:

To implement Fuzzy logic.

Description of the Concept or Problem given:

Implementation of fuzzy logic for a specific application

Manual Solution

Our Input would be of crisp input. Through fuzzification we load the fuzzy input into rule evaluation phase. Then we get out the fuzzy output. We then proceed to defuzzification andgive out the crisp output.

Program Implementation [Coding]

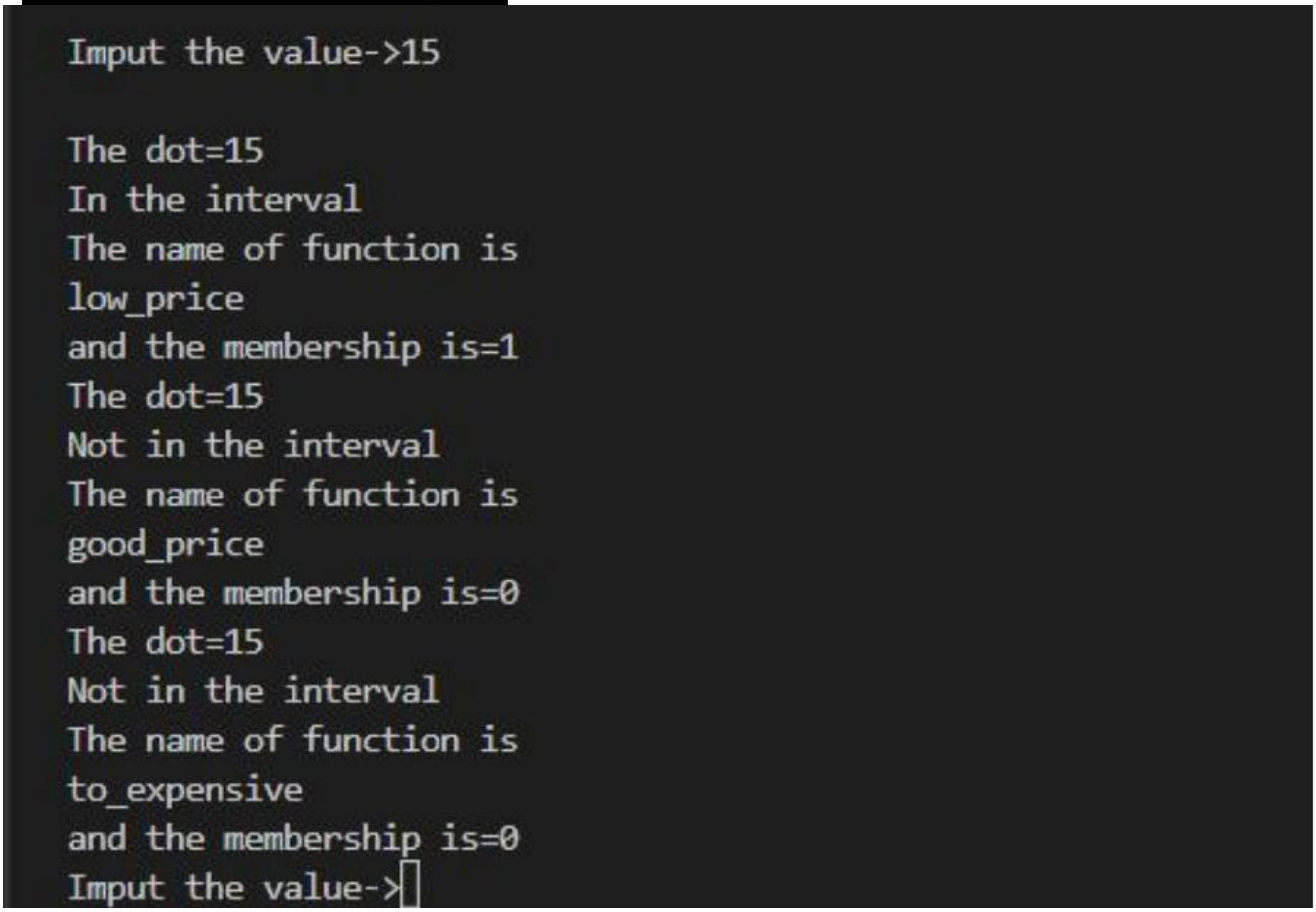
```
#include <iostream>
#include <cmath>
#include <cstring>

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)
return (dRight-t)/(dRight-dMiddle); else
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); else
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);

}

}
while(true);

return EXIT_SUCCESS;
}
```

Screenshots of the Outputs



```
Imput the value->15

The dot=15
In the interval
The name of function is
low_price
and the membership is=1
The dot=15
Not in the interval
The name of function is
good_price
and the membership is=0
The dot=15
Not in the interval
The name of function is
to_expensive
and the membership is=0
Imput the value->
```

Signature of the Student

MAINAK CHAUDHURI

Date: **Ex No:** 7.1	**Title of the Lab** Implementation of Unification in SWI Prolog	**Name:** Mainak Chaudhuri **Registration Number:** RA1911027010039 **Section:** N1 **Lab Batch:** 1 **Day Order:** 3

AIM:

To implement Unification in SWI Prolog.

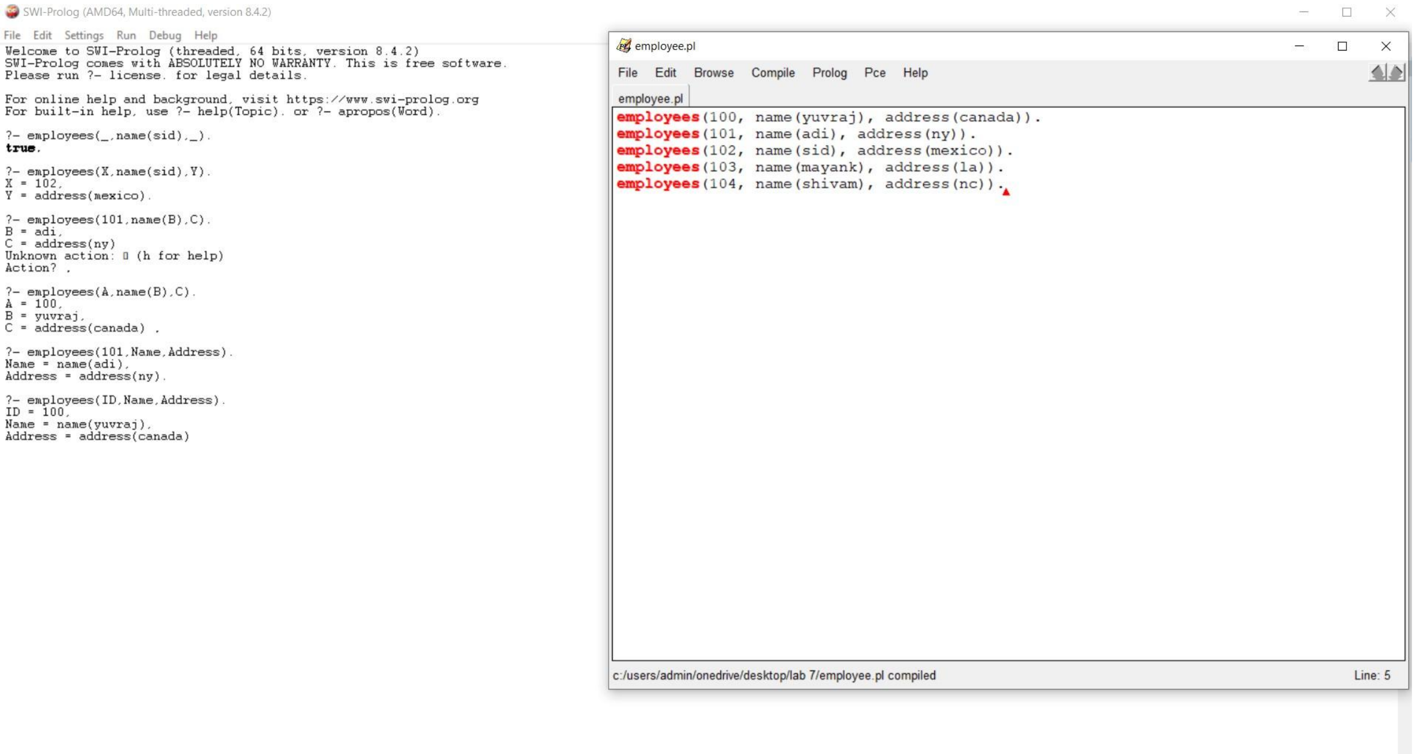
Description of the Concept or Problem given:

Prolog uses the unification technique, and it is a very general form of matching technique. In unification, one or more variables being given value to make the two call terms identical. This process is called binding the variables to values. For example, Prolog can unify the terms cat(A), and cat(mary) by binding variable A to atom mary that means we are giving the value mary to variable A.

Manual Solution:

```
1. If Y1 or Y2 is a variable or constant, then:
a) If Y1 , or Y2 are identical, then return NIL.
b) Else if Y1 is a variable,
        a. then if Y1, occurs in Y2, then return FAILURE
        b. Else return {({Y2,/Y1)).
c) Else if Y2 is a variable,
        a. If Y2 occurs in Y1, then return FAILURE,
        b. Else return {(Y1/Y2)}.
d) Else return FAILURE.
2. If the initial Predicate symbol in Y1, and Y2 are not same, then return FAILURE.
3. If Y1 and Y2 have a different number of arguments, then return FAILURE.
4. Set Substitution set(SUBST) to NIL.
5. For i=1 to the number of elements in Y1.
a) Call Unify function with the ith element of Y1, and ith element of Y2, and put the result into S.
b) If S=failure then returns Failure
c) If S =/= NIL then do,
        a. Apply S to the remainder of both L1 and L2.
        b. SUBST = APPEND(S, SUBST).
6. Return SUBST.
```

Screenshots of the Outputs:



SWI-Prolog (AMD64, Multi-threaded, version 8.4.2)
File Edit Settings Run Debug Help

```
Welcome to SWI-Prolog (threaded, 64 bits, version 8.4.2)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- employees(_,name(sid),_).
true.

?- employees(X,name(sid),Y).
X = 102,
Y = address(mexico).

?- employees(101,name(B),C).
B = adi,
C = address(ny)
Unknown action: 0 (h for help)
Action? .

?- employees(A,name(B),C).
A = 100,
B = yuvraj,
C = address(canada) .

?- employees(101,Name,Address).
Name = name(adi),
Address = address(ny).

?- employees(ID,Name,Address).
ID = 100,
Name = name(yuvraj),
Address = address(canada)
```

```
employees(100, name(yuvraj), address(canada)).
employees(101, name(adi), address(ny)).
employees(102, name(sid), address(mexico)).
employees(103, name(mayank), address(la)).
employees(104, name(shivam), address(nc)).
```

c:/users/admin/onedrive/desktop/lab 7/employee.pl compiled

Signature of the Student

[MAINAK CHAUDHURI]

Date: **Ex No:** 7.2	**Title of the Lab** Implementation of Resolution in SWI Prolog	**Name:** Mainak Chaudhuri **Registration Number:** RA1911027010039 **Section:** N1 **Lab Batch:** 1 **Day Order:** 3

AIM:

To implement Resolution in SWI Prolog.

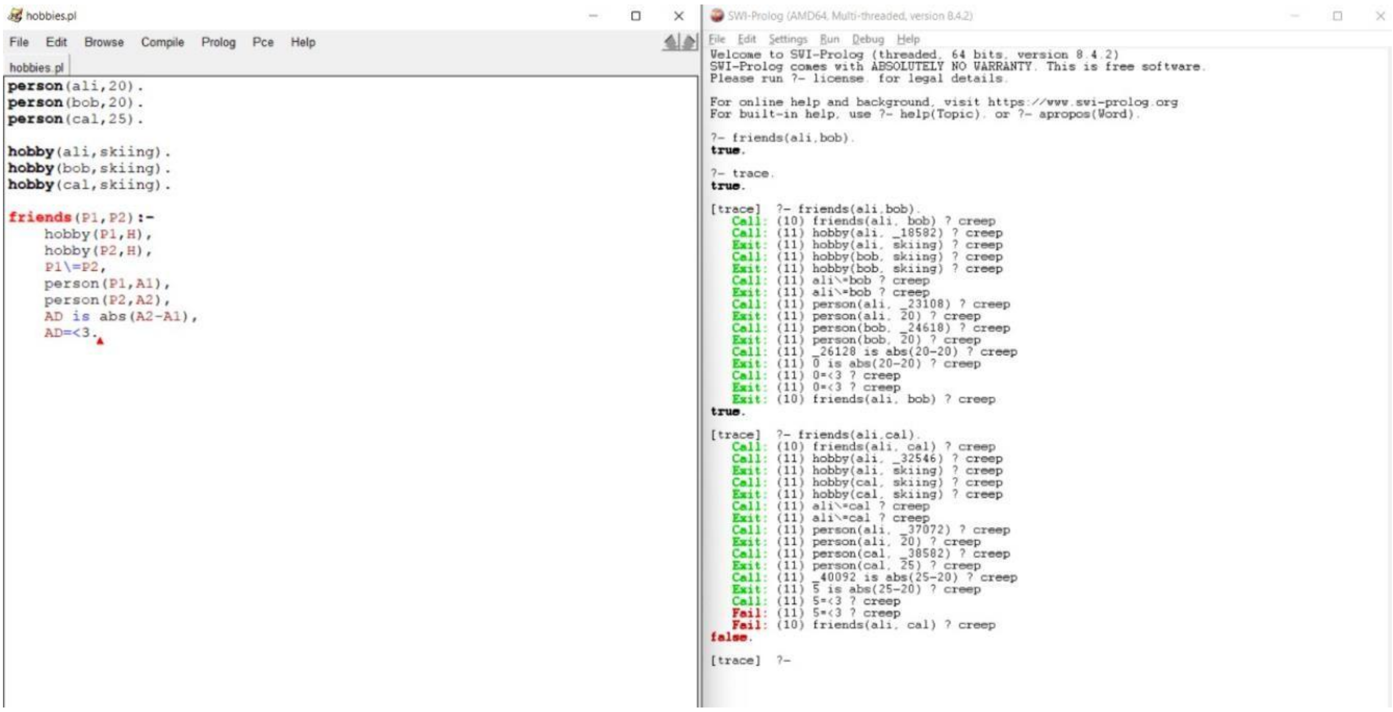
Description of the Concept or Problem given:

In simple words resolution is inference mechanism. Let's say we have clauses m :- b. and t :- p, m, z. So from that we can infer t :- p, b, z. - that is called resolution. Means, when you resolve two clauses you get one new clause. Another easy example, we have two sentences (1) All women like shopping. (2) Olivia is a woman. Now we ask query 'Who likes shopping'. So, by resolving above sentences we can have one new sentence Olivia likes shopping.

Manual Solution:

1. Conversion of facts into first-order logic.
2. Convert FOL statements into CNF
3. Negate the statement which needs to prove (proof by contradiction)
4. Draw resolution graph (unification).

Screenshots of the Outputs:


hobbies.pl
File Edit Browse Compile Prolog Pce Help
hobbies.pl
SWI-Prolog (AMD64, Multi-threaded, version 8.4.2)
File Edit Settings Run Debug Help

```prolog
person(ali,20).
person(bob,20).
person(cal,25).

hobby(ali,skiing).
hobby(bob,skiing).
hobby(cal,skiing).

friends(P1,P2):-
    hobby(P1,H),
    hobby(P2,H),
    P1\=P2,
    person(P1,A1),
    person(P2,A2),
    AD is abs(A2-A1),
    AD=<3.
```

```
Welcome to SWI-Prolog (threaded, 64 bits, version 8.4.2)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- friends(ali,bob).
true.

?- trace.
true.

[trace]  ?- friends(ali,bob).
   Call: (10) friends(ali, bob) ? creep
   Call: (11) hobby(ali, _18582) ? creep
   Exit: (11) hobby(ali, skiing) ? creep
   Call: (11) hobby(bob, skiing) ? creep
   Exit: (11) hobby(bob, skiing) ? creep
   Call: (11) ali\=bob ? creep
   Exit: (11) ali\=bob ? creep
   Call: (11) person(ali, _23108) ? creep
   Exit: (11) person(ali, 20) ? creep
   Call: (11) person(bob, _24618) ? creep
   Exit: (11) person(bob, 20) ? creep
   Call: (11) _26128 is abs(20-20) ? creep
   Exit: (11) 0 is abs(20-20) ? creep
   Call: (11) 0=<3 ? creep
   Exit: (11) 0=<3 ? creep
   Exit: (10) friends(ali, bob) ? creep
true.

[trace]  ?- friends(ali,cal).
   Call: (10) friends(ali, cal) ? creep
   Call: (11) hobby(ali, _32546) ? creep
   Exit: (11) hobby(ali, skiing) ? creep
   Call: (11) hobby(cal, skiing) ? creep
   Exit: (11) hobby(cal, skiing) ? creep
   Call: (11) ali\=cal ? creep
   Exit: (11) ali\=cal ? creep
   Call: (11) person(ali, _37072) ? creep
   Exit: (11) person(ali, 20) ? creep
   Call: (11) person(cal, _38582) ? creep
   Exit: (11) person(cal, 25) ? creep
   Call: (11) _40092 is abs(25-20) ? creep
   Exit: (11) 5 is abs(25-20) ? creep
   Call: (11) 5=<3 ? creep
   Fail: (11) 5=<3 ? creep
   Fail: (10) friends(ali, cal) ? creep
false.

[trace]  ?-
```

Signature of the Student

[MAINAK CHAUDHURI]

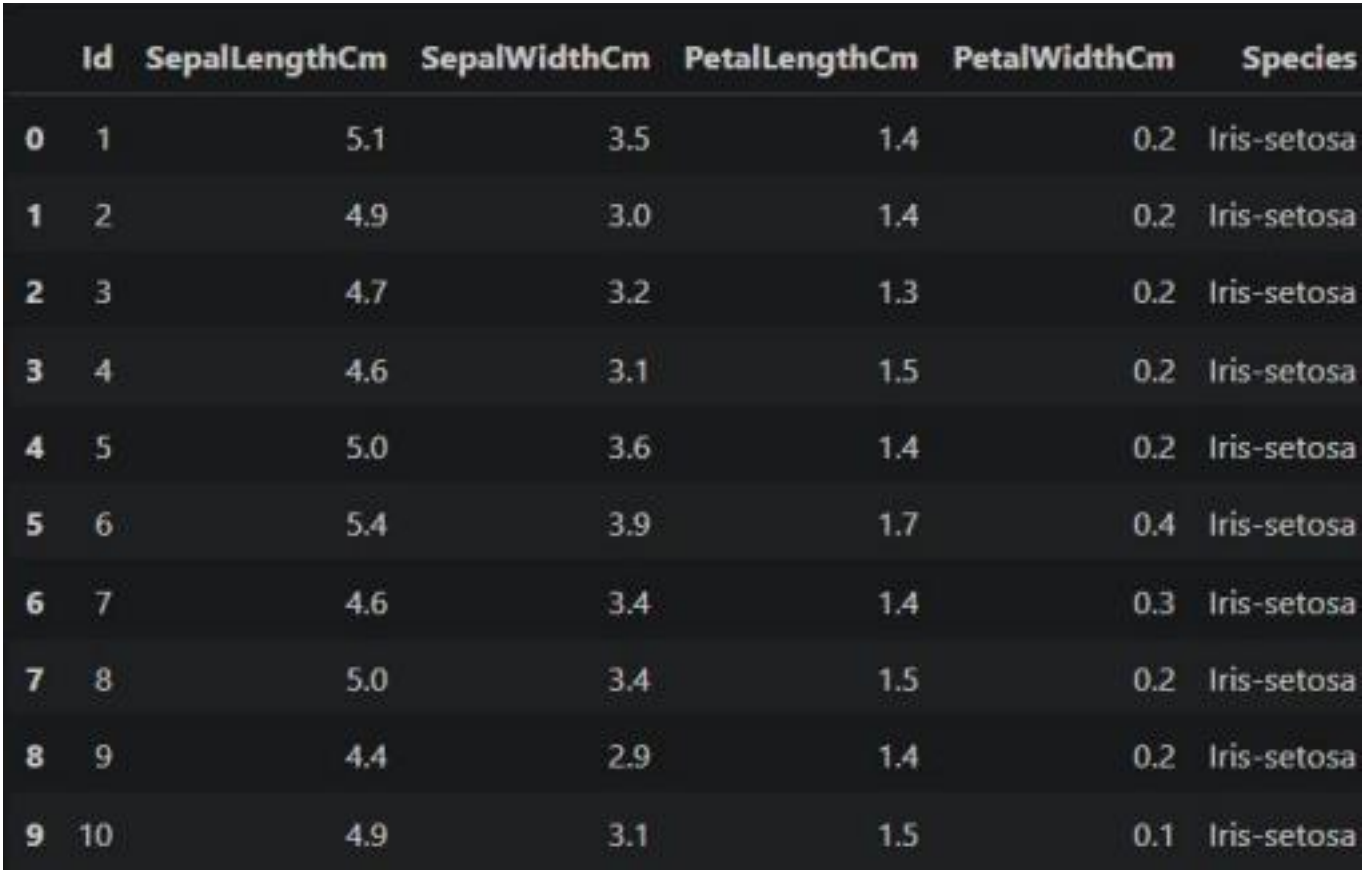
REG	RA1911027010039
NAME	MAINAK CHAUDHURI
EXP	8 Implementation of a Supervised Machine Le algorithms for an experimental

AIM : To find the best fitting algorithm for an Iris classifier dataset.

About the dataset:

The Iris is a violet-blue flower which has many species. However, for this experiment we will use 3 almost identical-looking species. They are *Setosa, Versicolor and Viginica*. The flowers of each individual species have some ranges of measures of Sepal and Petal widths and lengths. In this experiment we will try tounderstand something similar.

Glimpse of the Dataset:

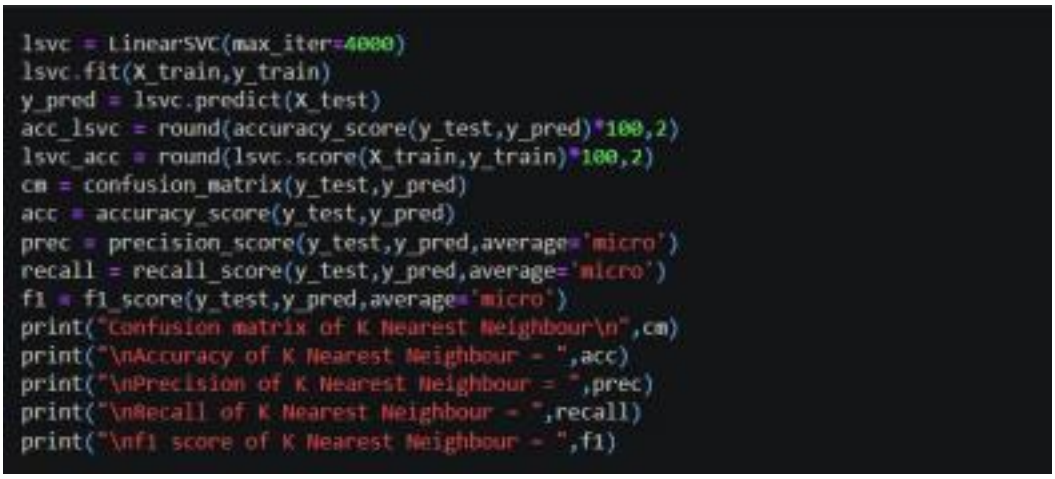


	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
5	6	5.4	3.9	1.7	0.4	Iris-setosa
6	7	4.6	3.4	1.4	0.3	Iris-setosa
7	8	5.0	3.4	1.5	0.2	Iris-setosa
8	9	4.4	2.9	1.4	0.2	Iris-setosa
9	10	4.9	3.1	1.5	0.1	Iris-setosa

We shall make use of this dataset for our experiment.

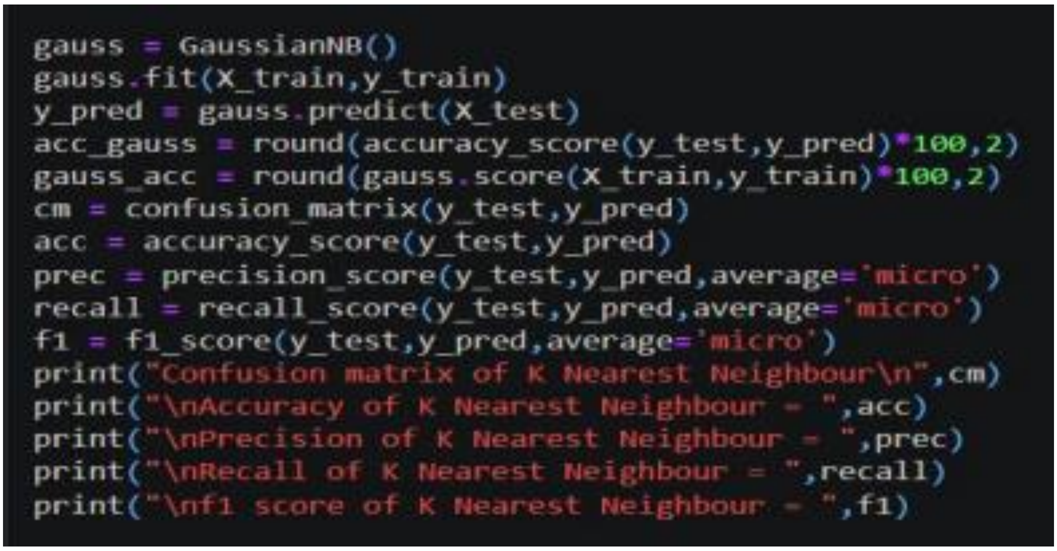
Procedure:

Proposed algorithm #1 : Support Vector Classifier



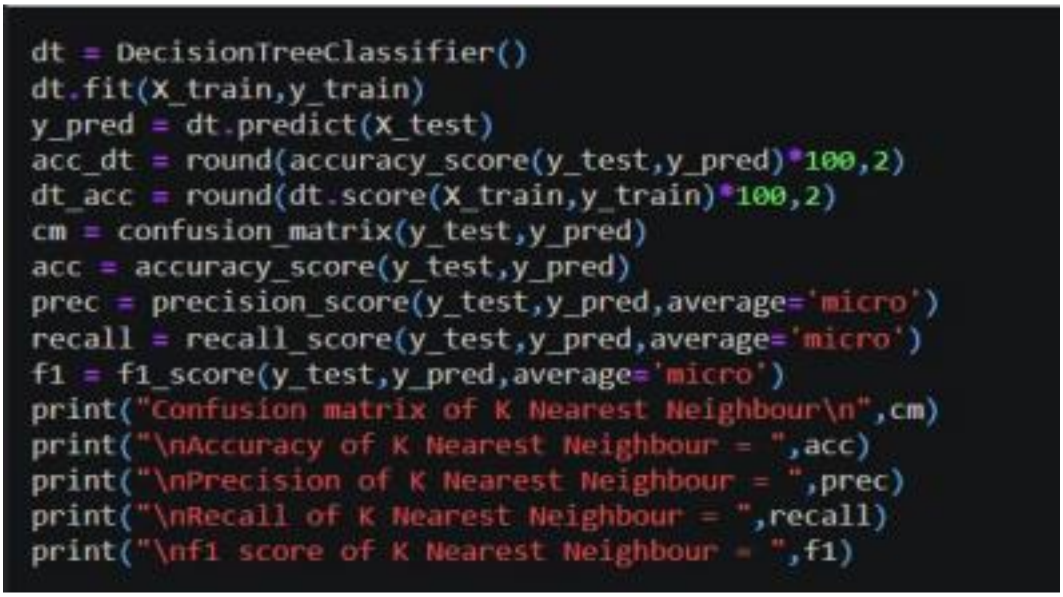
```
lsvc = LinearSVC(max_iter=4000)
lsvc.fit(X_train,y_train)
y_pred = lsvc.predict(X_test)
acc_lsvc = round(accuracy_score(y_test,y_pred)*100,2)
lsvc_acc = round(lsvc.score(X_train,y_train)*100,2)
cm = confusion_matrix(y_test,y_pred)
acc = accuracy_score(y_test,y_pred)
prec = precision_score(y_test,y_pred,average='micro')
recall = recall_score(y_test,y_pred,average='micro')
f1 = f1_score(y_test,y_pred,average='micro')
print("Confusion matrix of K Nearest Neighbour\n",cm)
print("\nAccuracy of K Nearest Neighbour = ",acc)
print("\nPrecision of K Nearest Neighbour = ",prec)
print("\nRecall of K Nearest Neighbour = ",recall)
print("\nf1 score of K Nearest Neighbour = ",f1)
```

Proposed algorithm #2 : Gaussian Naive Bayes



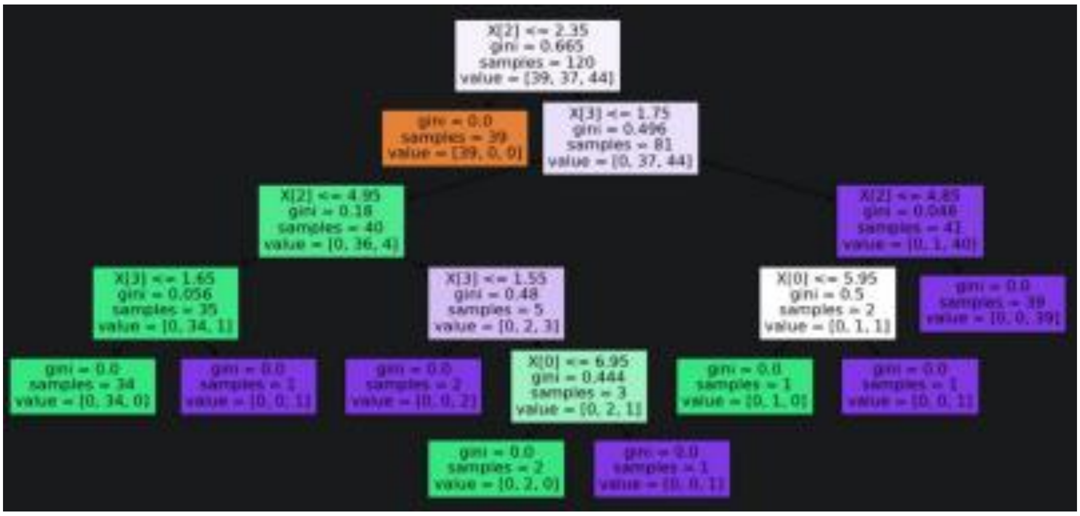
```
gauss = GaussianNB()
gauss.fit(X_train,y_train)
y_pred = gauss.predict(X_test)
acc_gauss = round(accuracy_score(y_test,y_pred)*100,2)
gauss_acc = round(gauss.score(X_train,y_train)*100,2)
cm = confusion_matrix(y_test,y_pred)
acc = accuracy_score(y_test,y_pred)
prec = precision_score(y_test,y_pred,average='micro')
recall = recall_score(y_test,y_pred,average='micro')
f1 = f1_score(y_test,y_pred,average='micro')
print("Confusion matrix of K Nearest Neighbour\n",cm)
print("\nAccuracy of K Nearest Neighbour = ",acc)
print("\nPrecision of K Nearest Neighbour = ",prec)
print("\nRecall of K Nearest Neighbour = ",recall)
print("\nf1 score of K Nearest Neighbour = ",f1)
```

Proposed algorithm #3 : Decision Tree Classifier

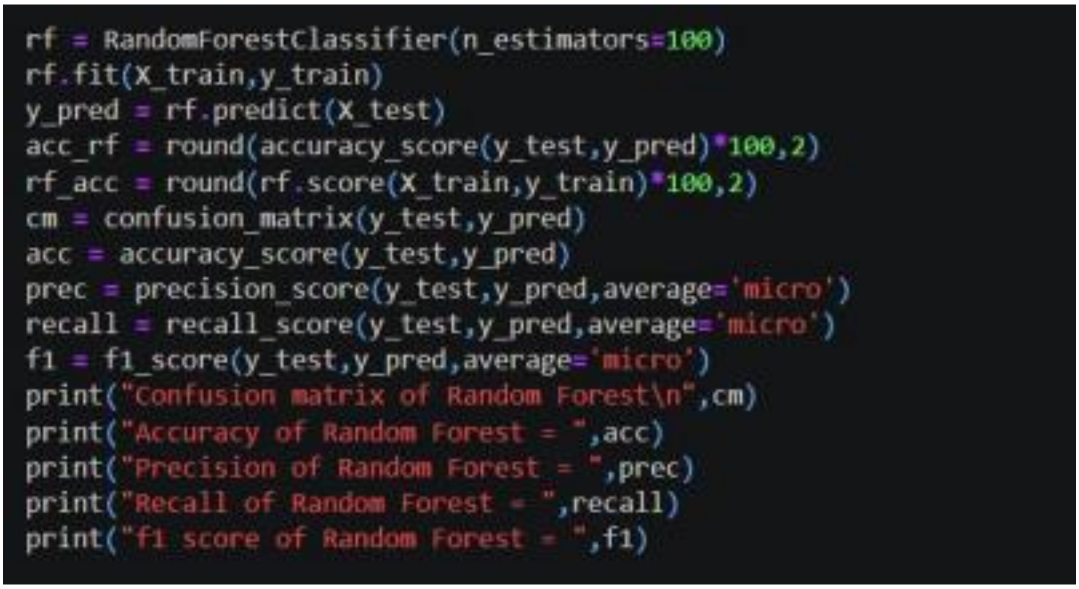


```
dt = DecisionTreeClassifier()
dt.fit(X_train,y_train)
y_pred = dt.predict(X_test)
acc_dt = round(accuracy_score(y_test,y_pred)*100,2)
dt_acc = round(dt.score(X_train,y_train)*100,2)
cm = confusion_matrix(y_test,y_pred)
acc = accuracy_score(y_test,y_pred)
prec = precision_score(y_test,y_pred,average='micro')
recall = recall_score(y_test,y_pred,average='micro')
f1 = f1_score(y_test,y_pred,average='micro')
print("Confusion matrix of K Nearest Neighbour\n",cm)
print("\nAccuracy of K Nearest Neighbour = ",acc)
print("\nPrecision of K Nearest Neighbour = ",prec)
print("\nRecall of K Nearest Neighbour = ",recall)
print("\nf1 score of K Nearest Neighbour = ",f1)
```

Decision Tree Generated:

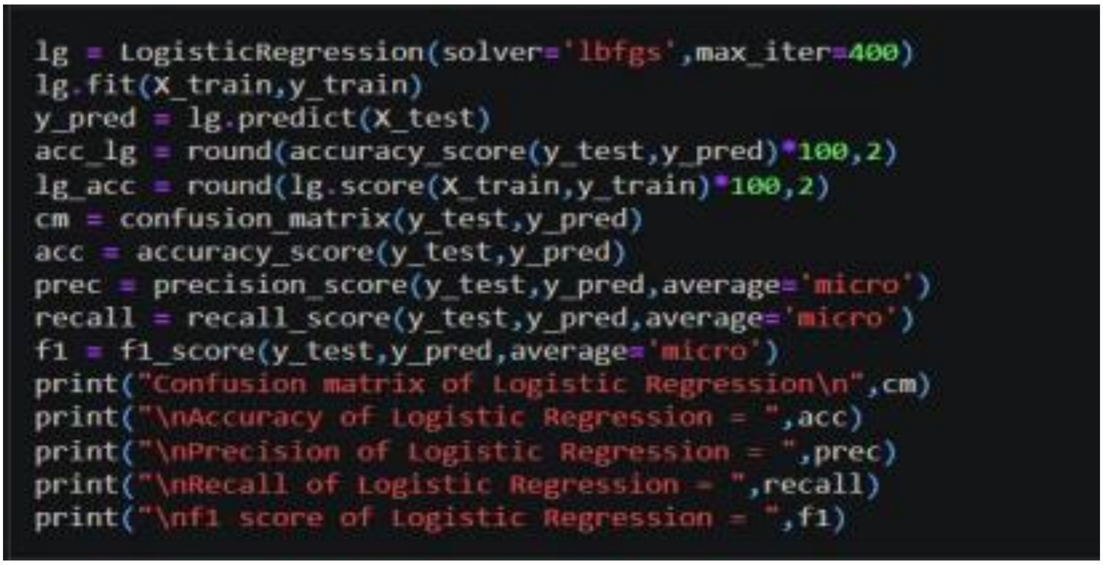


Proposed algorithm #4 : Random Forest Classifier



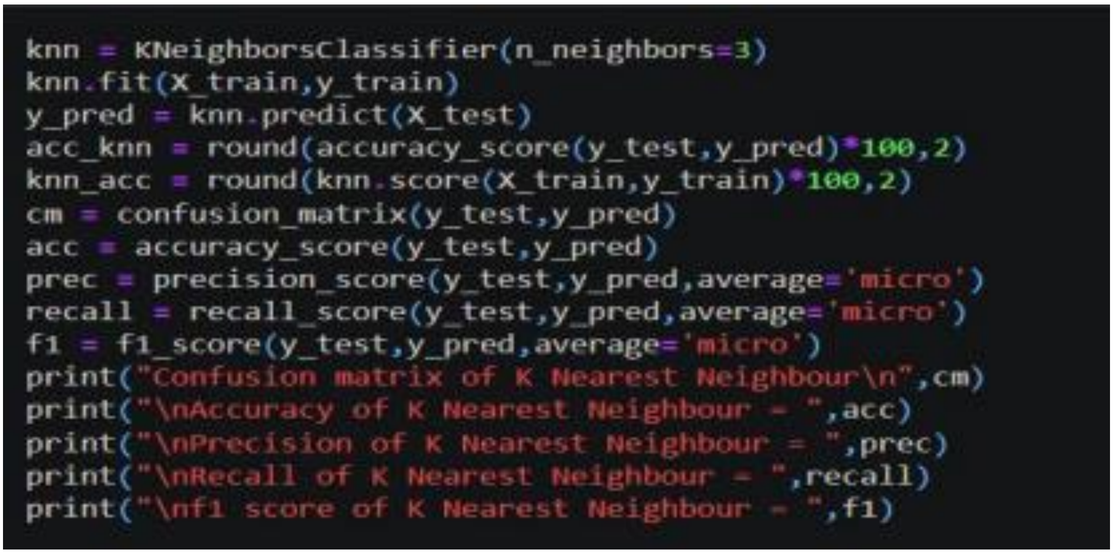
```
rf = RandomForestClassifier(n_estimators=100)
rf.fit(X_train,y_train)
y_pred = rf.predict(X_test)
acc_rf = round(accuracy_score(y_test,y_pred)*100,2)
rf_acc = round(rf.score(X_train,y_train)*100,2)
cm = confusion_matrix(y_test,y_pred)
acc = accuracy_score(y_test,y_pred)
prec = precision_score(y_test,y_pred,average='micro')
recall = recall_score(y_test,y_pred,average='micro')
f1 = f1_score(y_test,y_pred,average='micro')
print("Confusion matrix of Random Forest\n",cm)
print("Accuracy of Random Forest = ",acc)
print("Precision of Random Forest = ",prec)
print("Recall of Random Forest = ",recall)
print("f1 score of Random Forest = ",f1)
```

Proposed algorithm #5 : Logistic Regression



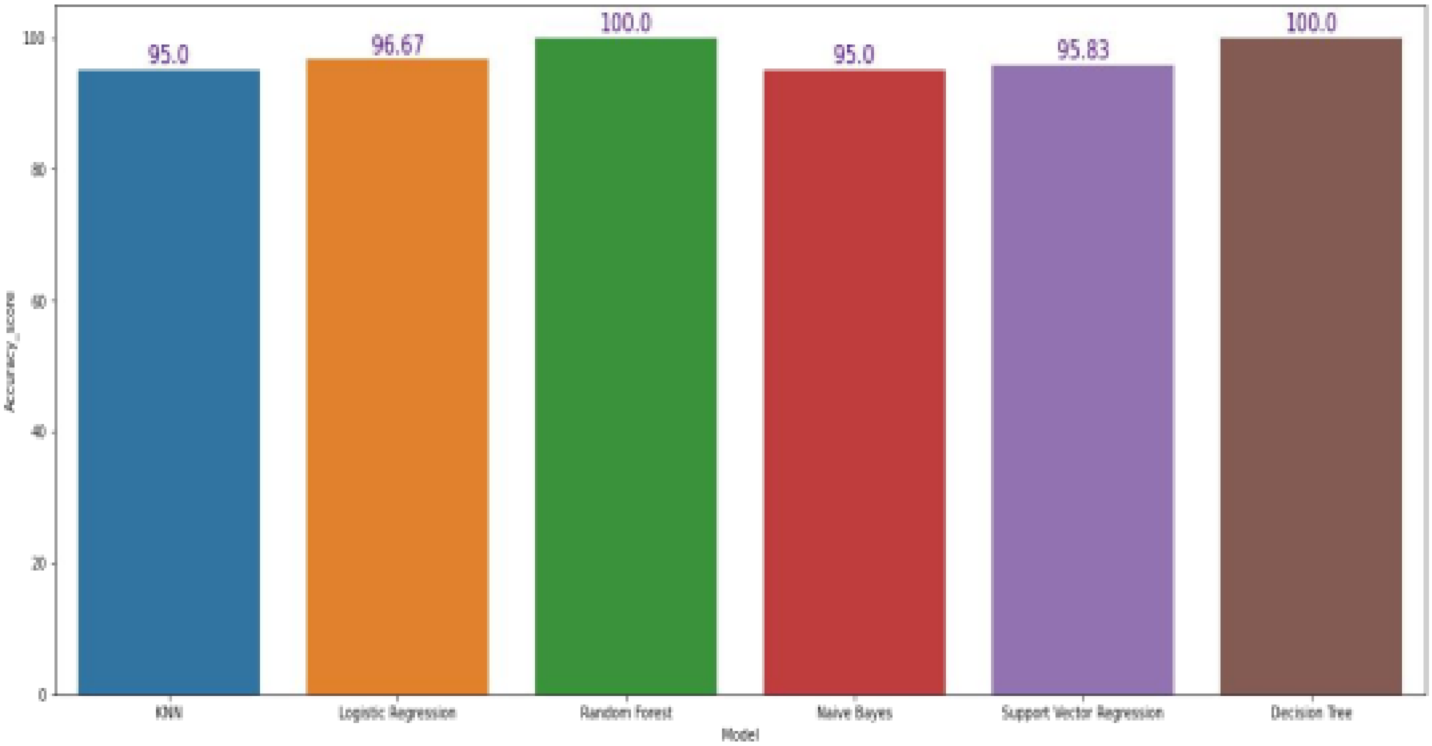
```
lg = LogisticRegression(solver='lbfgs',max_iter=400)
lg.fit(X_train,y_train)
y_pred = lg.predict(X_test)
acc_lg = round(accuracy_score(y_test,y_pred)*100,2)
lg_acc = round(lg.score(X_train,y_train)*100,2)
cm = confusion_matrix(y_test,y_pred)
acc = accuracy_score(y_test,y_pred)
prec = precision_score(y_test,y_pred,average='micro')
recall = recall_score(y_test,y_pred,average='micro')
f1 = f1_score(y_test,y_pred,average='micro')
print("Confusion matrix of Logistic Regression\n",cm)
print("\nAccuracy of Logistic Regression = ",acc)
print("\nPrecision of Logistic Regression = ",prec)
print("\nRecall of Logistic Regression = ",recall)
print("\nf1 score of Logistic Regression = ",f1)
```

Proposed algorithm #6 : K-Nearest Neighbours



```
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train,y_train)
y_pred = knn.predict(X_test)
acc_knn = round(accuracy_score(y_test,y_pred)*100,2)
knn_acc = round(knn.score(X_train,y_train)*100,2)
cm = confusion_matrix(y_test,y_pred)
acc = accuracy_score(y_test,y_pred)
prec = precision_score(y_test,y_pred,average='micro')
recall = recall_score(y_test,y_pred,average='micro')
f1 = f1_score(y_test,y_pred,average='micro')
print("Confusion matrix of K Nearest Neighbour\n",cm)
print("\nAccuracy of K Nearest Neighbour = ",acc)
print("\nPrecision of K Nearest Neighbour = ",prec)
print("\nRecall of K Nearest Neighbour = ",recall)
print("\nf1 score of K Nearest Neighbour = ",f1)
```

Evaluation of Performance:



Conclusion:

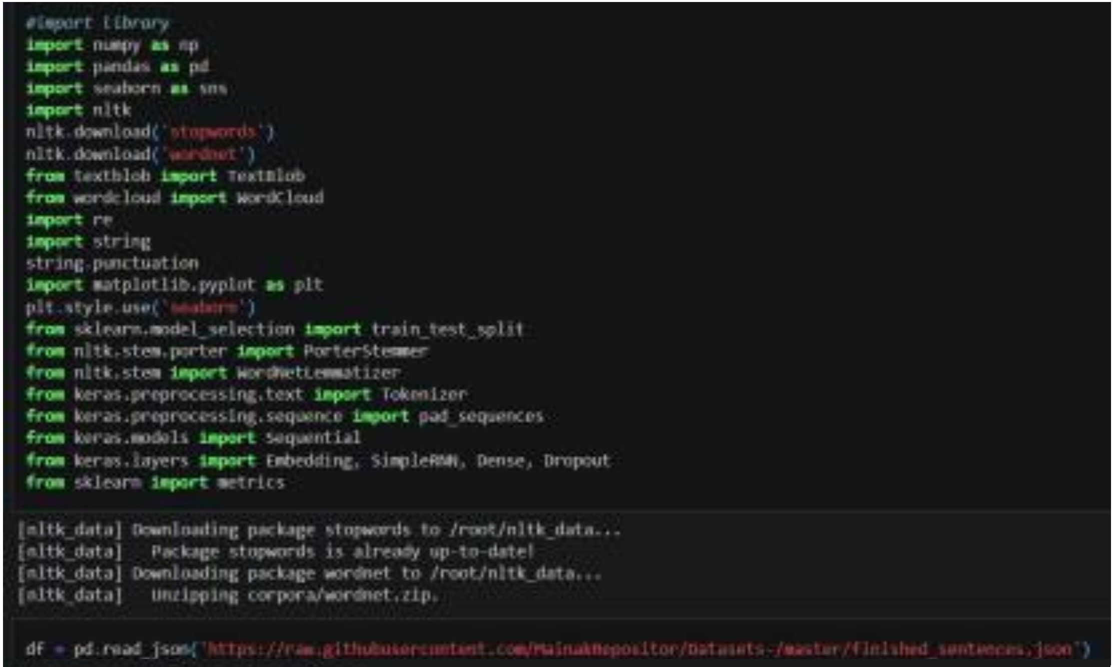
The model accuracy of the Random Forest and Decision Tree are 100%. But this doesnotmean that the models are the best. Rather they are overfitted and will tendtoyieldastereotypic result for any more variations in the dataset. So we will rule themout.

Apart from them, we can see that Logistic Regression has a the highest accuracyscoreof96.67% which is a very good accuracy. This also has least probable changes of overfitting.So, the best model for the analysis is “**Logistic Regression”**

REG.	No. **RA1911027010039**
NAME	MAINAK CHAUDHURI
LAB	No. 9
TOPIC	Analysis using NLP

AIM : Classify complete and incomplete sentences.

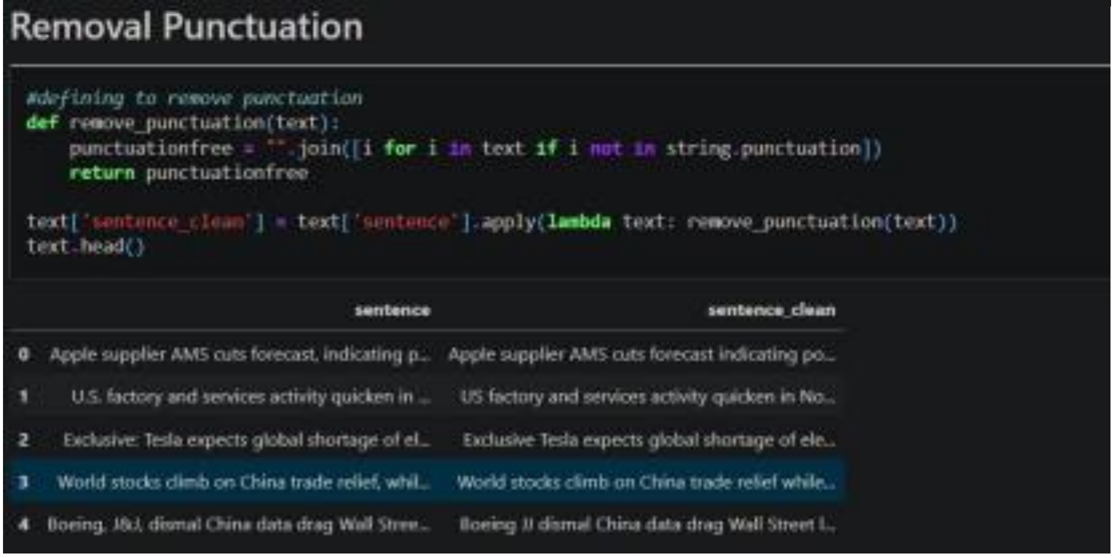
1. Importing necessary packages for NLP



```
#import library
import numpy as np
import pandas as pd
import seaborn as sns
import nltk
nltk.download('stopwords')
nltk.download('wordnet')
from textblob import TextBlob
from wordcloud import WordCloud
import re
import string
string.punctuation
import matplotlib.pyplot as plt
plt.style.use('seaborn')
from sklearn.model_selection import train_test_split
from nltk.stem.porter import PorterStemmer
from nltk.stem import WordNetLemmatizer
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Embedding, SimpleRNN, Dense, Dropout
from sklearn import metrics
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Unzipping corpora/wordnet.zip.
```

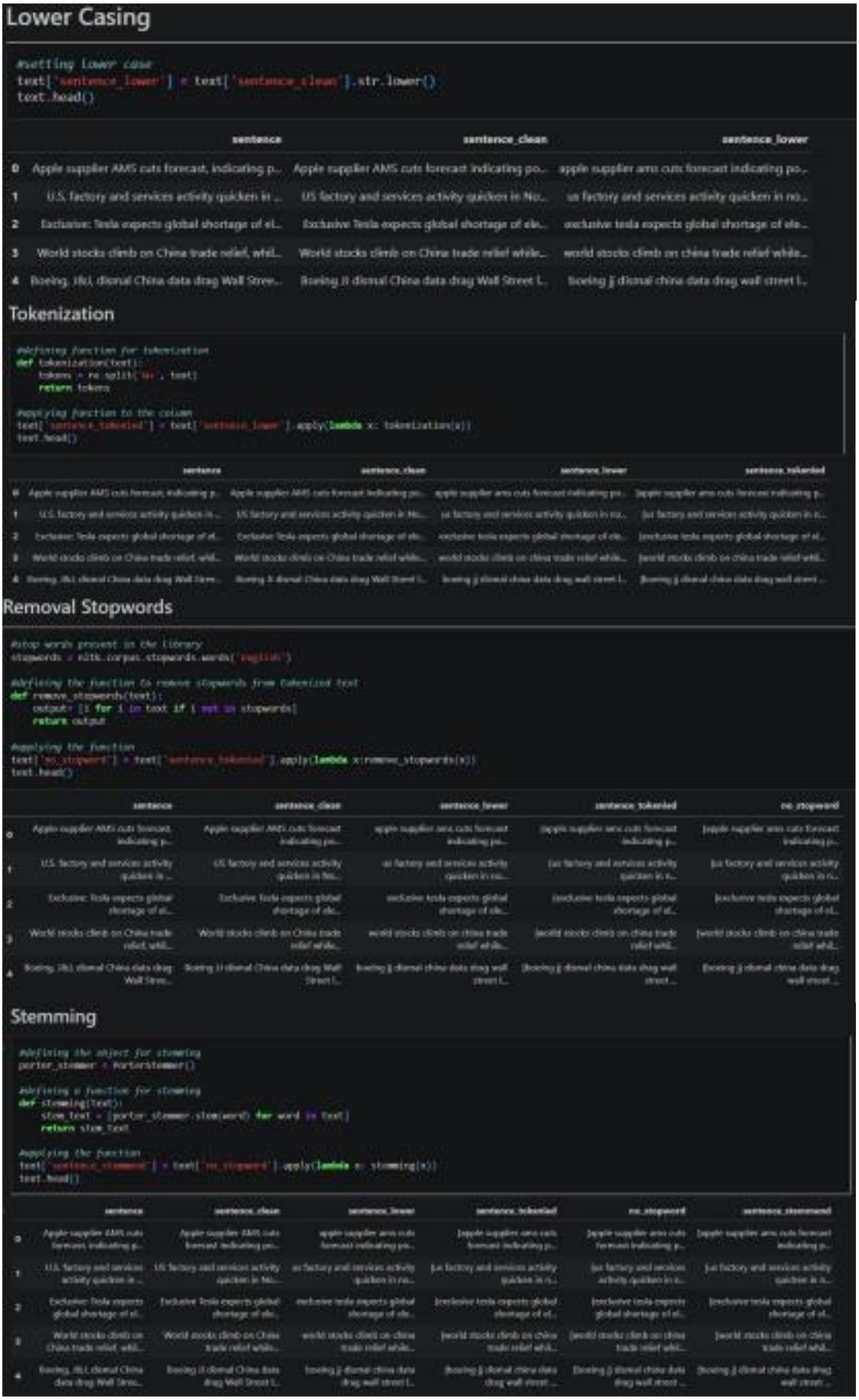
```
df = pd.read_json('https://raw.githubusercontent.com/MainakRepositor/Datasets-/master/finished_sentences.json')
```



Removal Punctuation

```
#defining to remove punctuation
def remove_punctuation(text):
    punctuationfree = "".join([i for i in text if i not in string.punctuation])
    return punctuationfree

text['sentence_clean'] = text['sentence'].apply(lambda text: remove_punctuation(text))
text.head()
```

	sentence	sentence_clean
0	Apple supplier AMS cuts forecast, indicating p...	Apple supplier AMS cuts forecast indicating po...
1	U.S. factory and services activity quicken in ...	US factory and services activity quicken in No...
2	Exclusive: Tesla expects global shortage of el...	Exclusive Tesla expects global shortage of ele...
3	World stocks climb on China trade relief, whil...	World stocks climb on China trade relief while...
4	Boeing, J&J, dismal China data drag Wall Stree...	Boeing JJ dismal China data drag Wall Street l...



Lower Casing

```
#setting lower case
text['sentence_lower'] = text['sentence_clean'].str.lower()
text.head()
```

	sentence	sentence_clean	sentence_lower
0	Apple supplier AMS cuts forecast, indicating p...	Apple supplier AMS cuts forecast indicating po...	apple supplier ams cuts forecast indicating po...
1	U.S. factory and services activity quicken in ...	US factory and services activity quicken in No...	us factory and services activity quicken in no...
2	Exclusive: Tesla expects global shortage of el...	Exclusive Tesla expects global shortage of ele...	exclusive tesla expects global shortage of ele...
3	World stocks climb on China trade relief, whil...	World stocks climb on China trade relief while...	world stocks climb on china trade relief while...
4	Boeing, J&J, dismal China data drag Wall Stree...	Boeing JJ dismal China data drag Wall Street L...	boeing jj dismal china data drag wall street l...

Tokenization

```
#defining function for tokenization
def tokenization(text):
    tokens = re.split('\W+', text)
    return tokens

#applying function to the column
text['sentence_tokenied'] = text['sentence_lower'].apply(lambda x: tokenization(x))
text.head()
```

	sentence	sentence_clean	sentence_lower	sentence_tokenied
0	Apple supplier AMS cuts forecast, indicating p...	Apple supplier AMS cuts forecast indicating po...	apple supplier ams cuts forecast indicating po...	[apple supplier ams cuts forecast indicating p...
1	U.S. factory and services activity quicken in ...	US factory and services activity quicken in No...	us factory and services activity quicken in no...	[us factory and services activity quicken in n...
2	Exclusive: Tesla expects global shortage of el...	Exclusive Tesla expects global shortage of ele...	exclusive tesla expects global shortage of ele...	[exclusive tesla expects global shortage of el...
3	World stocks climb on China trade relief, whil...	World stocks climb on China trade relief while...	world stocks climb on china trade relief while...	[world stocks climb on china trade relief whil...
4	Boeing, J&J, dismal China data drag Wall Stree...	Boeing JJ dismal China data drag Wall Street L...	boeing jj dismal china data drag wall street l...	[boeing jj dismal china data drag wall street ...

Removal Stopwords

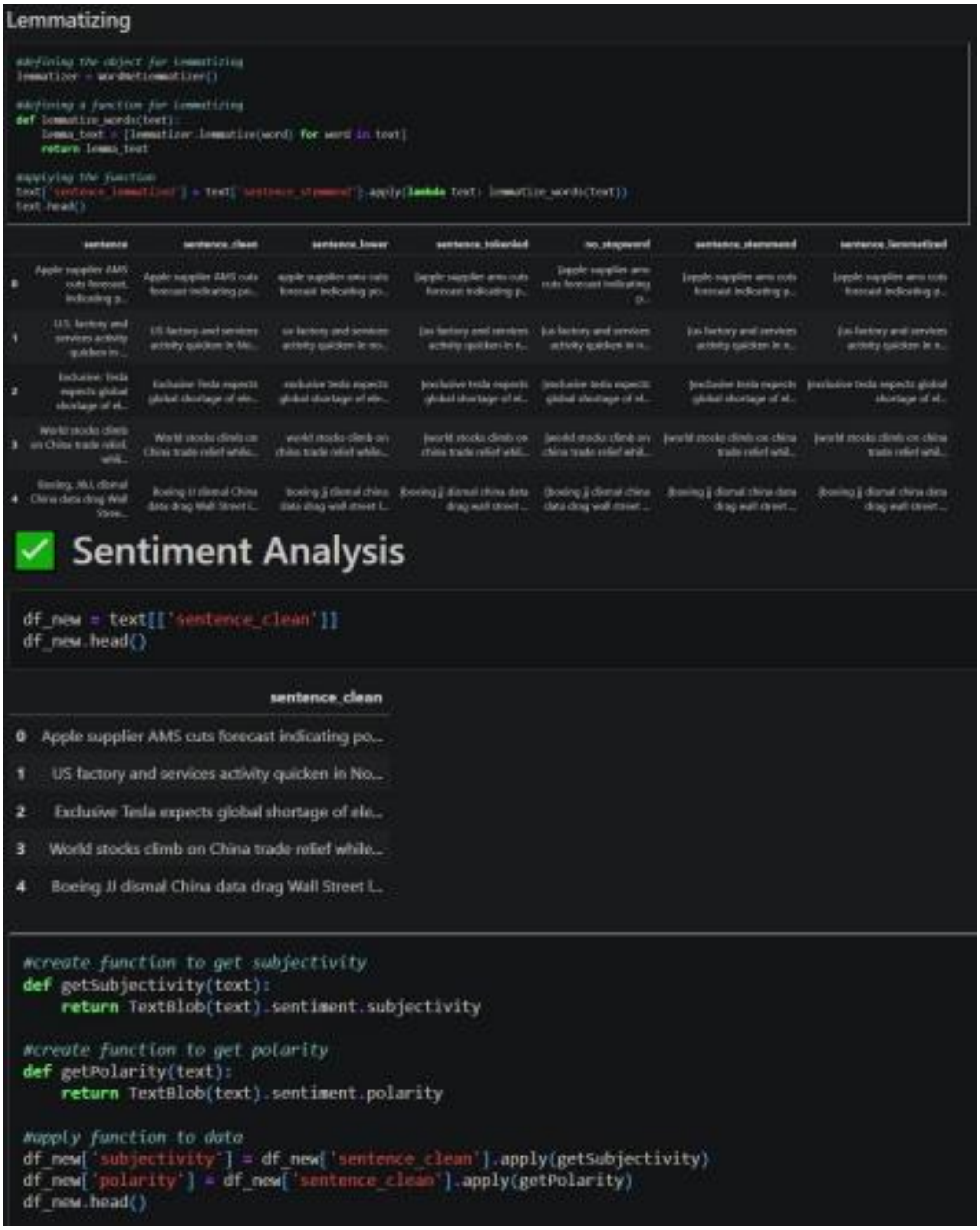
```
#stop words present in the library
stopwords = nltk.corpus.stopwords.words('english')

#defining the function to remove stopwords from tokenized text
def remove_stopwords(text):
    output= [i for i in text if i not in stopwords]
    return output

#applying the function
text['no_stopword'] = text['sentence_tokenied'].apply(lambda x:remove_stopwords(x))
text.head()
```

	sentence	sentence_clean	sentence_lower	sentence_tokenied	no_stopword
0	Apple supplier AMS cuts forecast, indicating p...	Apple supplier AMS cuts forecast indicating po...	apple supplier ams cuts forecast indicating po...	[apple supplier ams cuts forecast indicating p...	[apple supplier ams cuts forecast indicating p...
1	U.S. factory and services activity quicken in ...	US factory and services activity quicken in No...	us factory and services activity quicken in no...	[us factory and services activity quicken in n...	[us factory and services activity quicken in n...
2	Exclusive: Tesla expects global shortage of el...	Exclusive Tesla expects global shortage of ele...	exclusive tesla expects global shortage of ele...	[exclusive tesla expects global shortage of el...	[exclusive tesla expects global shortage of el...
3	World stocks climb on China trade relief, whil...	World stocks climb on China trade relief while...	world stocks climb on china trade relief while...	[world stocks climb on china trade relief whil...	[world stocks climb on china trade relief whil...
4	Boeing, J&J, dismal China data drag Wall Stree...	Boeing JJ dismal China data drag Wall Street L...	boeing jj dismal china data drag wall street l...	[boeing jj dismal china data drag wall street ...	[boeing jj dismal china data drag wall street ...

Stemming

```
#defining the object for stemming
porter_stemmer = PorterStemmer()

#defining a function for stemming
def stemming(text):
    stem_text = [porter_stemmer.stem(word) for word in text]
    return stem_text

#applying the function
text['sentence_stemmed'] = text['no_stopword'].apply(lambda x: stemming(x))
text.head()
```

	sentence	sentence_clean	sentence_lower	sentence_tokenied	no_stopword	sentence_stemmed
0	Apple supplier AMS cuts forecast, indicating p...	Apple supplier AMS cuts forecast indicating po...	apple supplier ams cuts forecast indicating po...	[apple supplier ams cuts forecast indicating p...	[apple supplier ams cuts forecast indicating p...	[apple supplier ams cuts forecast indicating p...
1	U.S. factory and services activity quicken in ...	US factory and services activity quicken in No...	us factory and services activity quicken in no...	[us factory and services activity quicken in n...	[us factory and services activity quicken in n...	[us factory and services activity quicken in n...
2	Exclusive: Tesla expects global shortage of el...	Exclusive Tesla expects global shortage of ele...	exclusive tesla expects global shortage of ele...	[exclusive tesla expects global shortage of el...	[exclusive tesla expects global shortage of el...	[exclusive tesla expects global shortage of el...
3	World stocks climb on China trade relief, whil...	World stocks climb on China trade relief while...	world stocks climb on china trade relief while...	[world stocks climb on china trade relief whil...	[world stocks climb on china trade relief whil...	[world stocks climb on china trade relief whil...
4	Boeing, J&J, dismal China data drag Wall Stree...	Boeing JJ dismal China data drag Wall Street L...	boeing jj dismal china data drag wall street l...	[boeing jj dismal china data drag wall street ...	[boeing jj dismal china data drag wall street ...	[boeing jj dismal china data drag wall street ...



Lemmatizing

```
#defining the object for lemmatizing
lemmatizer = WordNetLemmatizer()

#defining a function for lemmatizing
def lemmatize_words(text):
    lemma_text = [lemmatizer.lemmatize(word) for word in text]
    return lemma_text

#applying the function
text['sentence_lemmatized'] = text['sentence_stemmed'].apply(lambda text: lemmatize_words(text))
text.head()
```

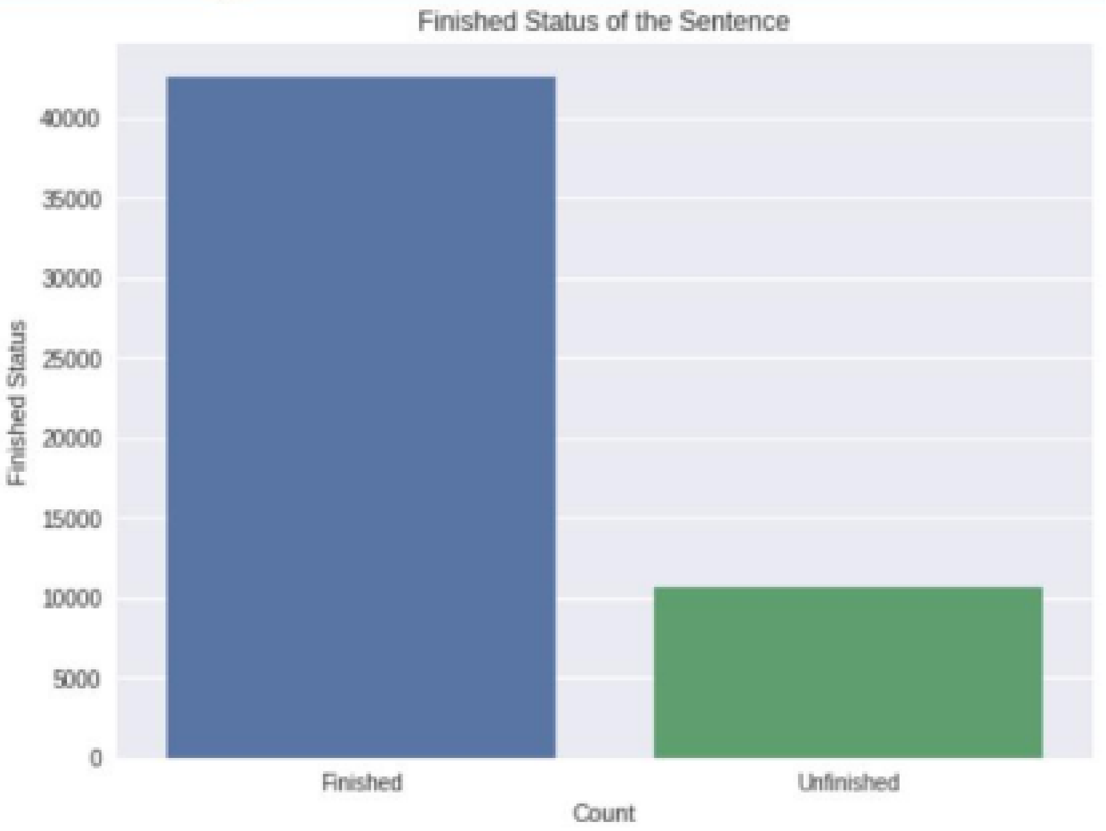
	sentence	sentence_clean	sentence_lower	sentence_tokenized	no_stopword	sentence_stemmed	sentence_lemmatized
0	Apple supplier AMS cuts forecast, indicating p...	Apple supplier AMS cuts forecast indicating po...	apple supplier ams cuts forecast indicating po...	[apple supplier ams cuts forecast indicating p...	[apple supplier ams cuts forecast indicating p...	[apple supplier ams cuts forecast indicating p...	[apple supplier ams cuts forecast indicating p...
1	U.S. factory and services activity quicken in ...	US factory and services activity quicken in No...	us factory and services activity quicken in no...	[us factory and services activity quicken in n...	[us factory and services activity quicken in n...	[us factory and services activity quicken in n...	[us factory and services activity quicken in n...
2	Exclusive: Tesla expects global shortage of el...	Exclusive Tesla expects global shortage of ele...	exclusive tesla expects global shortage of ele...	[exclusive tesla expects global shortage of el...	[exclusive tesla expects global shortage of el...	[exclusive tesla expects global shortage of el...	[exclusive tesla expects global shortage of el...
3	World stocks climb on China trade relief, whil...	World stocks climb on China trade relief while...	world stocks climb on china trade relief while...	[world stocks climb on china trade relief whil...	[world stocks climb on china trade relief whil...	[world stocks climb on china trade relief whil...	[world stocks climb on china trade relief whil...
4	Boeing, J&J, dismal China data drag Wall Stree...	Boeing JJ dismal China data drag Wall Street I...	boeing jj dismal china data drag wall street l...	[boeing jj dismal china data drag wall street ...	[boeing jj dismal china data drag wall street ...	[boeing jj dismal china data drag wall street...	[boeing jj dismal china data drag wall street...

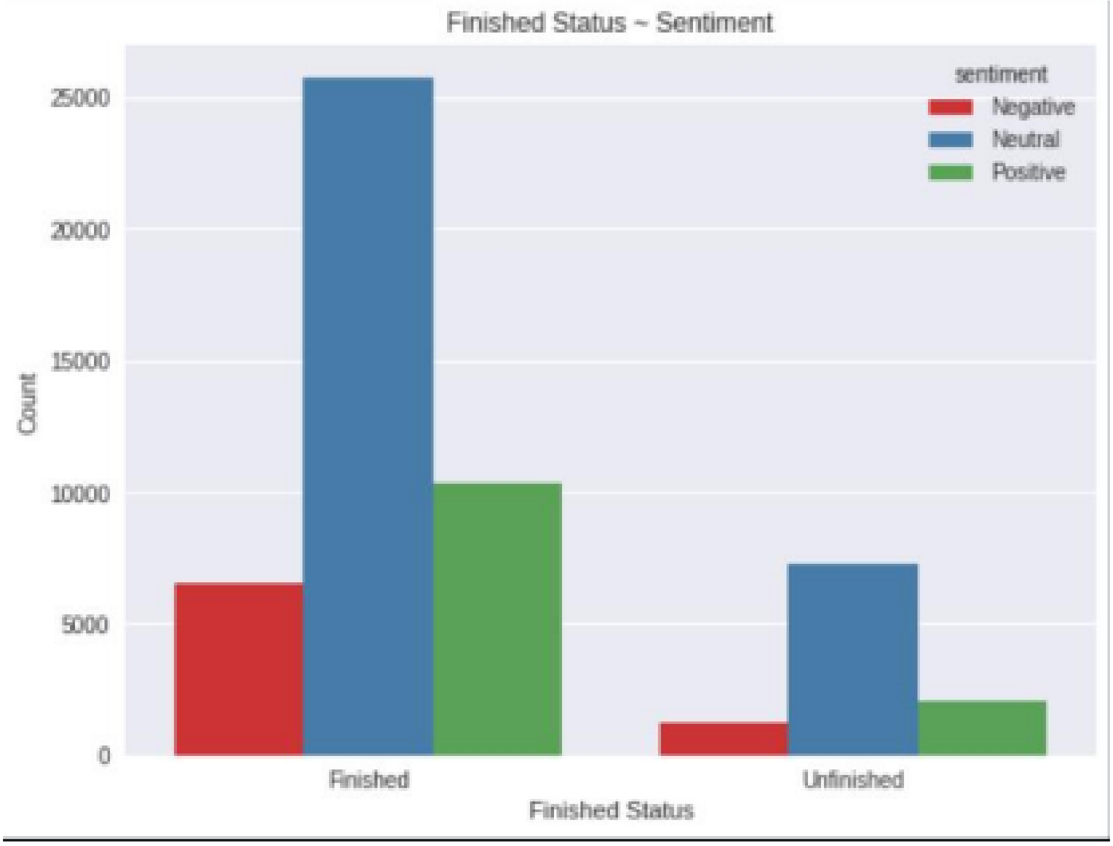
✅ Sentiment Analysis

```
df_new = text[['sentence_clean']]
df_new.head()
```

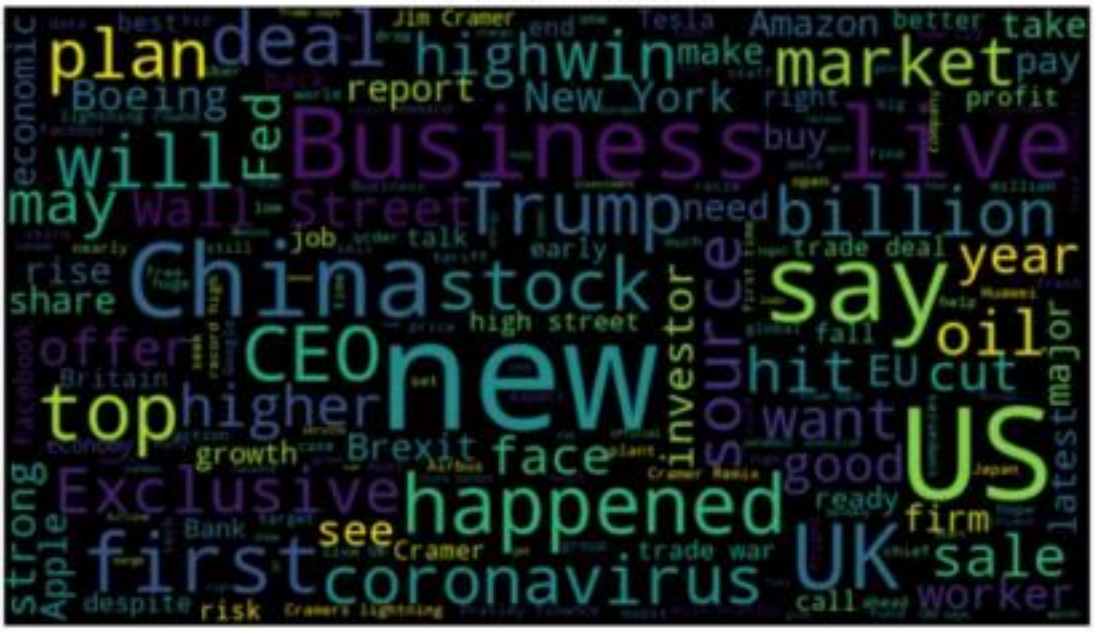
	sentence_clean
0	Apple supplier AMS cuts forecast indicating po...
1	US factory and services activity quicken in No...
2	Exclusive Tesla expects global shortage of ele...
3	World stocks climb on China trade relief while...
4	Boeing JJ dismal China data drag Wall Street I...

```
#create function to get subjectivity
def getSubjectivity(text):
    return TextBlob(text).sentiment.subjectivity

#create function to get polarity
def getPolarity(text):
    return TextBlob(text).sentiment.polarity

#apply function to data
df_new['subjectivity'] = df_new['sentence_clean'].apply(getSubjectivity)
df_new['polarity'] = df_new['sentence_clean'].apply(getPolarity)
df_new.head()
```

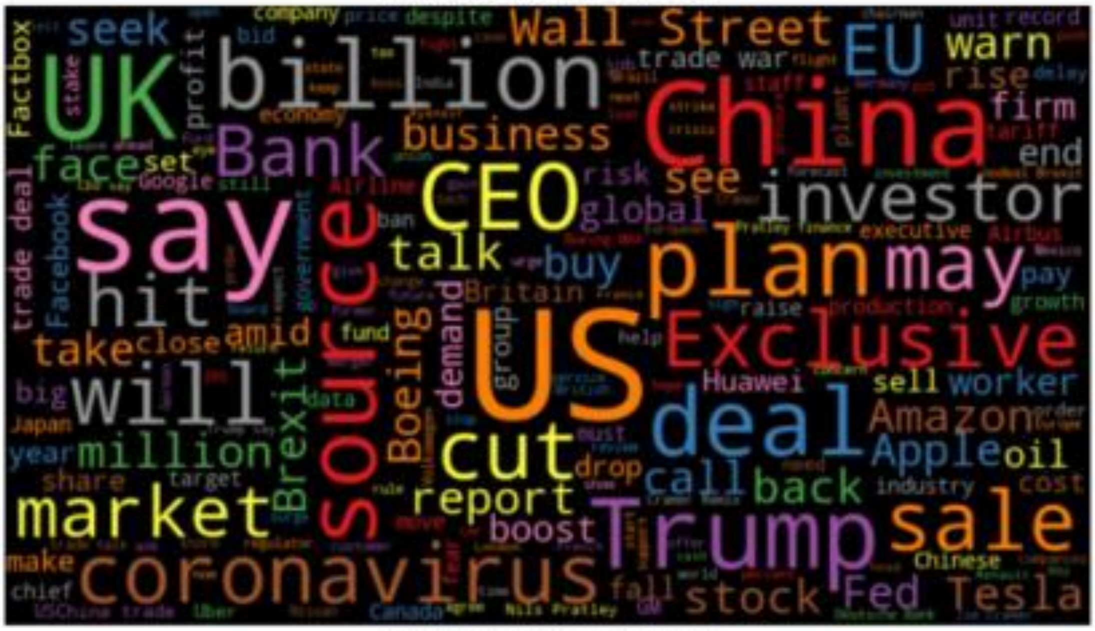

Finished Status of the Sentence
Finished Status
40000
35000
30000
25000
20000
15000
10000
5000
0
Finished
Unfinished
Count


Finished Status ~ Sentiment
sentiment
Negative
Neutral
Positive
Count
25000
20000
15000
10000
5000
0
Finished
Unfinished
Finished Status

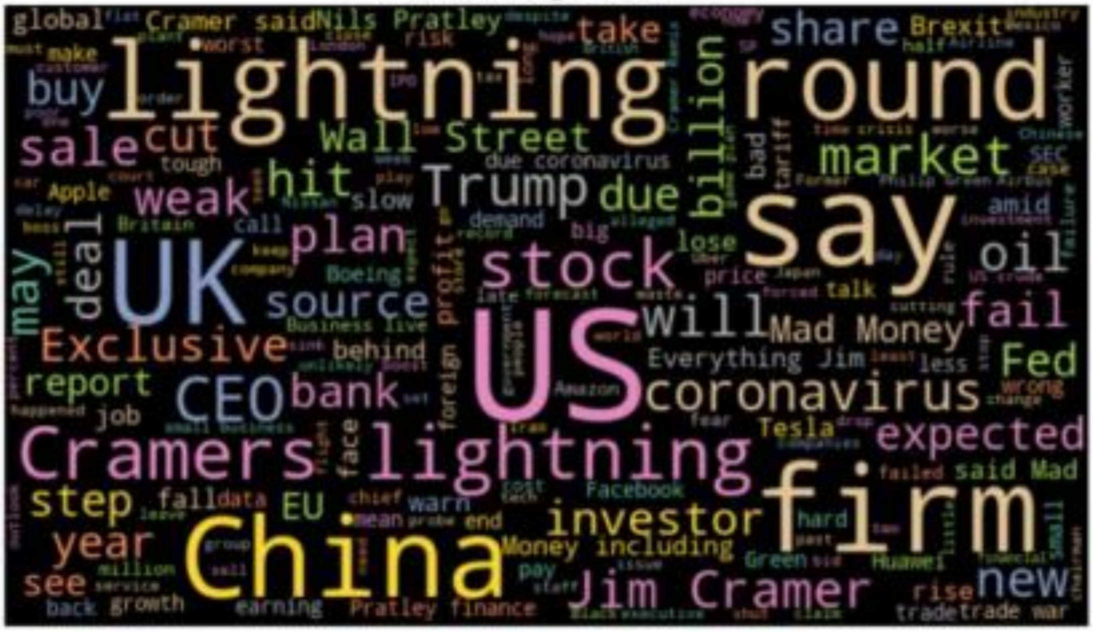
Wordcloud of Positive Sentence

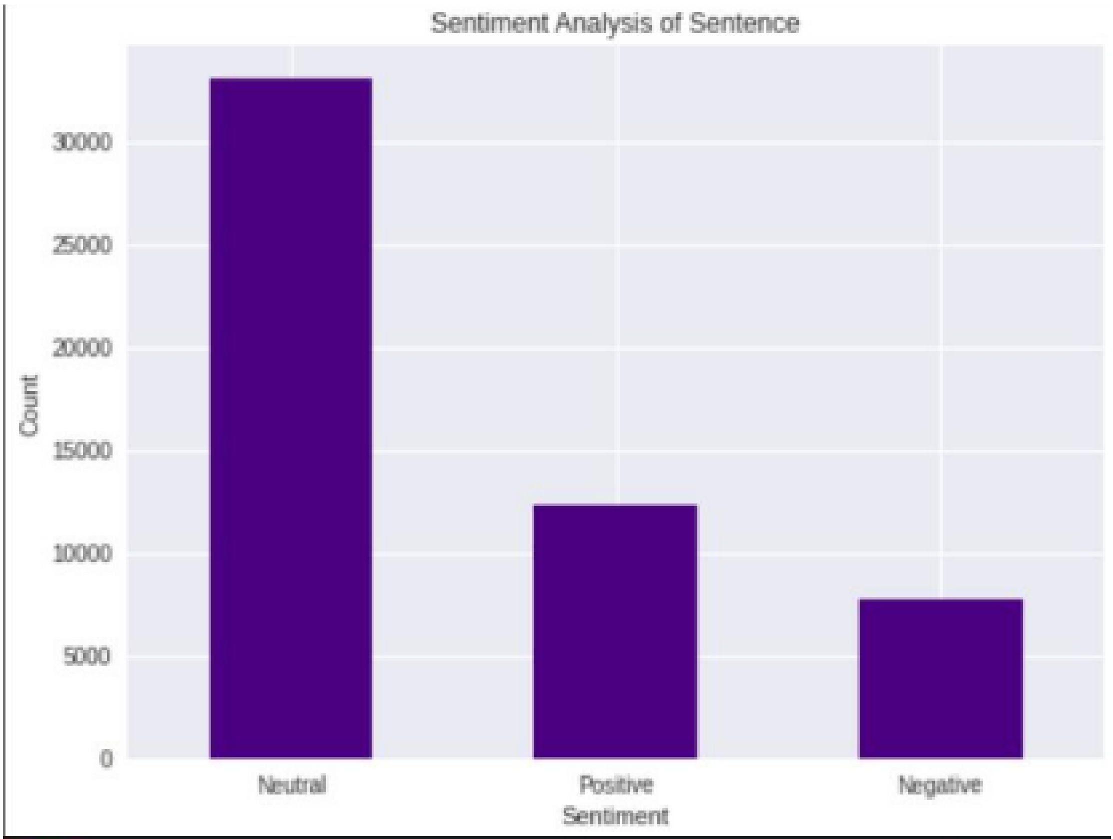


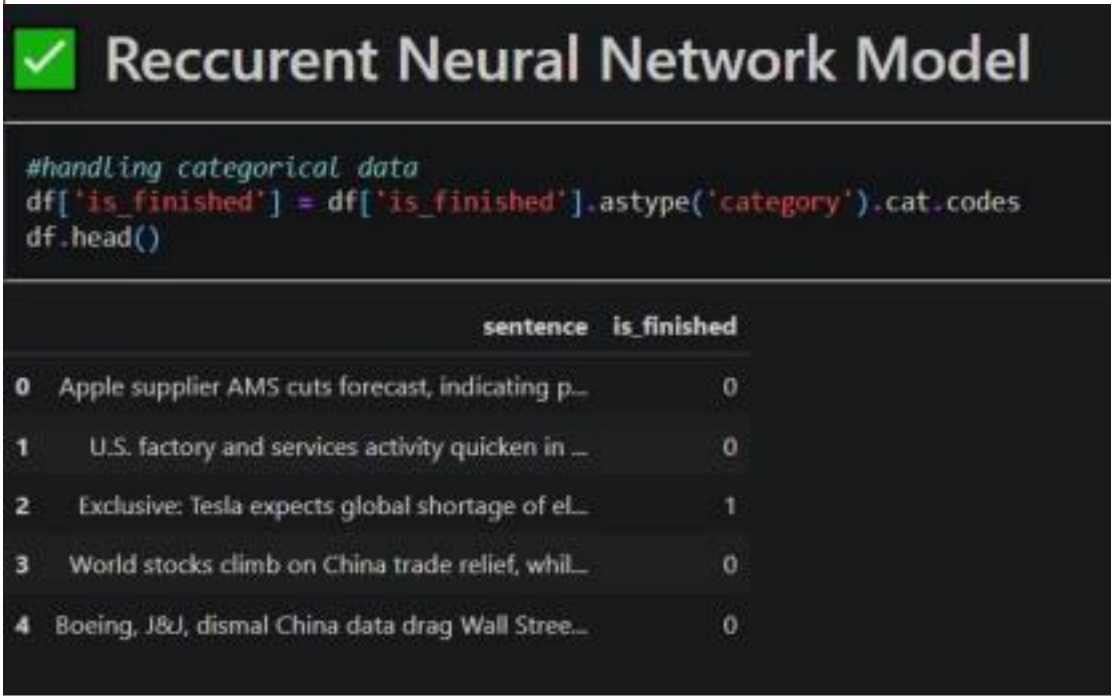
Wordcloud of Neutral Sentence



Wordcloud of Negative Sentence







✅ Reccurent Neural Network Model

```
#handling categorical data
df['is_finished'] = df['is_finished'].astype('category').cat.codes
df.head()
```

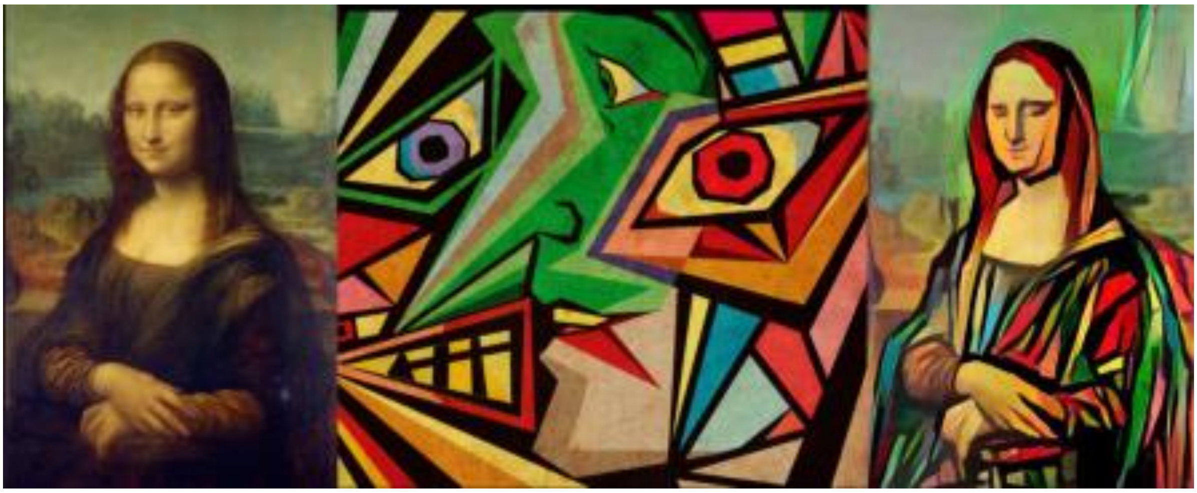
	sentence	is_finished
0	Apple supplier AMS cuts forecast, indicating p...	0
1	U.S. factory and services activity quicken in ...	0
2	Exclusive: Tesla expects global shortage of el...	1
3	World stocks climb on China trade relief, whil...	0
4	Boeing, J&J, dismal China data drag Wall Stree...	0

Conclusion: Therefore the incomplete sentences are analysed with91%accuracy and segmented properly.

REG. Nc	RA1911027010039
NAME	MAINAK CHAUDHURI
Exp	No. 10 Application of Deep Learning Model on anApplicatio

AIM : Transfer Image Style from one picture to other using GAN(General Adversarial Network)

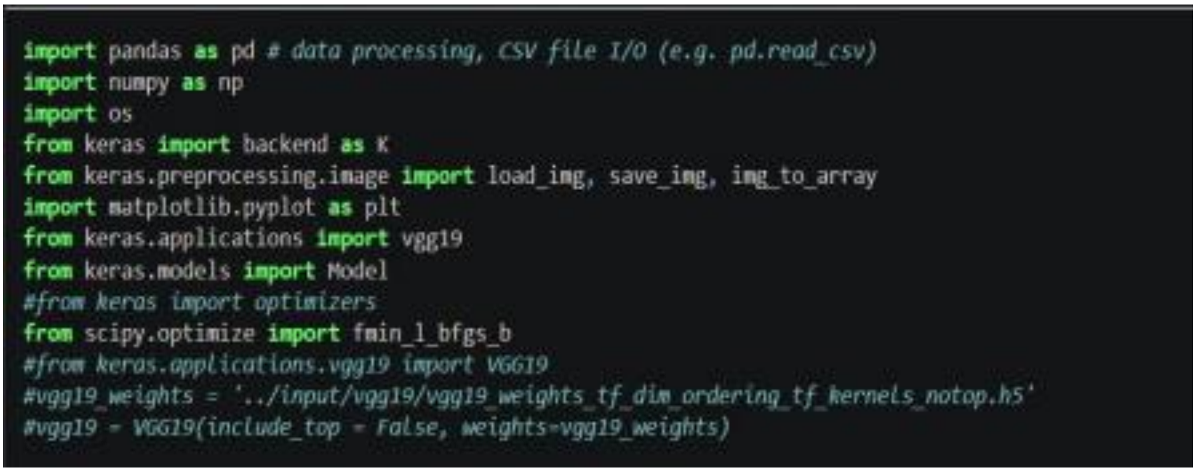
Target:



Steps:

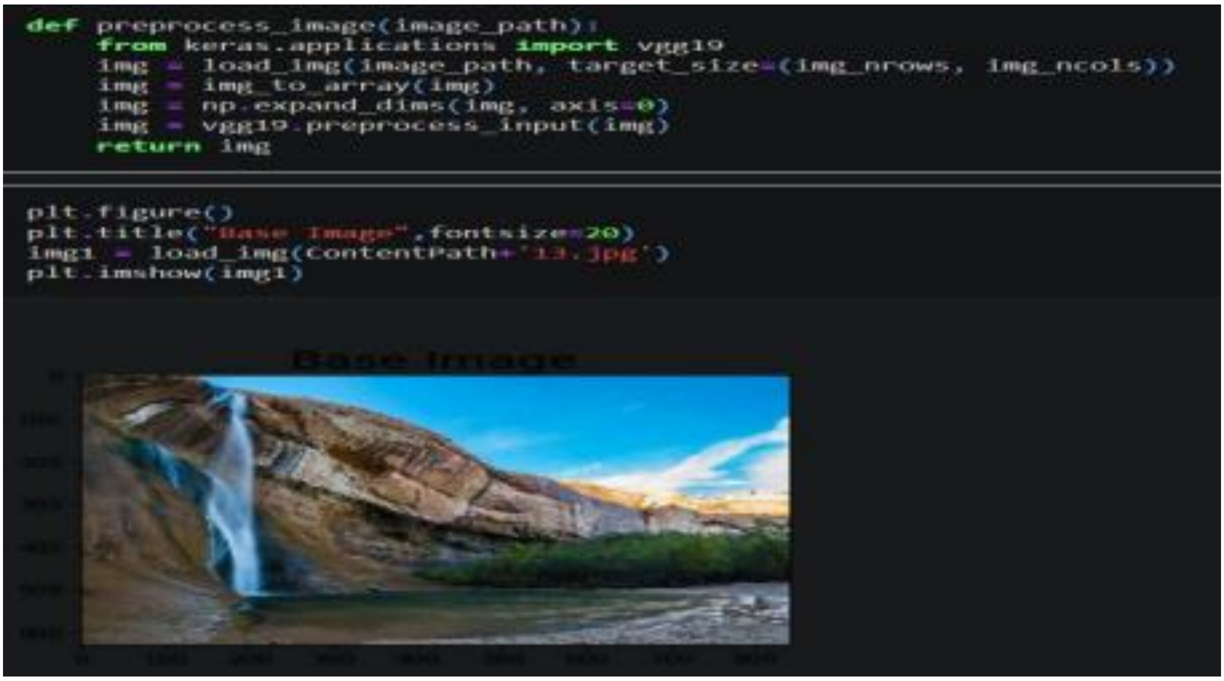
1. Obtain the actual or base image.
2. Obtain the style image.
3. Read the pixels of the base image.
4. Generate a statistical model of the pixels and their colour, depth andintensities.
5. Remove each pixel of the actual image and regenerate the same withthepixels of the style image.
6. The image matrix and pixel statistics helps the newer pixels of the styleimage to adjust in the exact places and do the needful.
7. Thus the final image will be obtained with the imposed style.

Importing the necessary packages:



```python
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import numpy as np
import os
from keras import backend as K
from keras.preprocessing.image import load_img, save_img, img_to_array
import matplotlib.pyplot as plt
from keras.applications import vgg19
from keras.models import Model
#from keras import optimizers
from scipy.optimize import fmin_l_bfgs_b
#from keras.applications.vgg19 import VGG19
#vgg19_weights = '../input/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5'
#vgg19 = VGG19(include_top = False, weights=vgg19_weights)
```

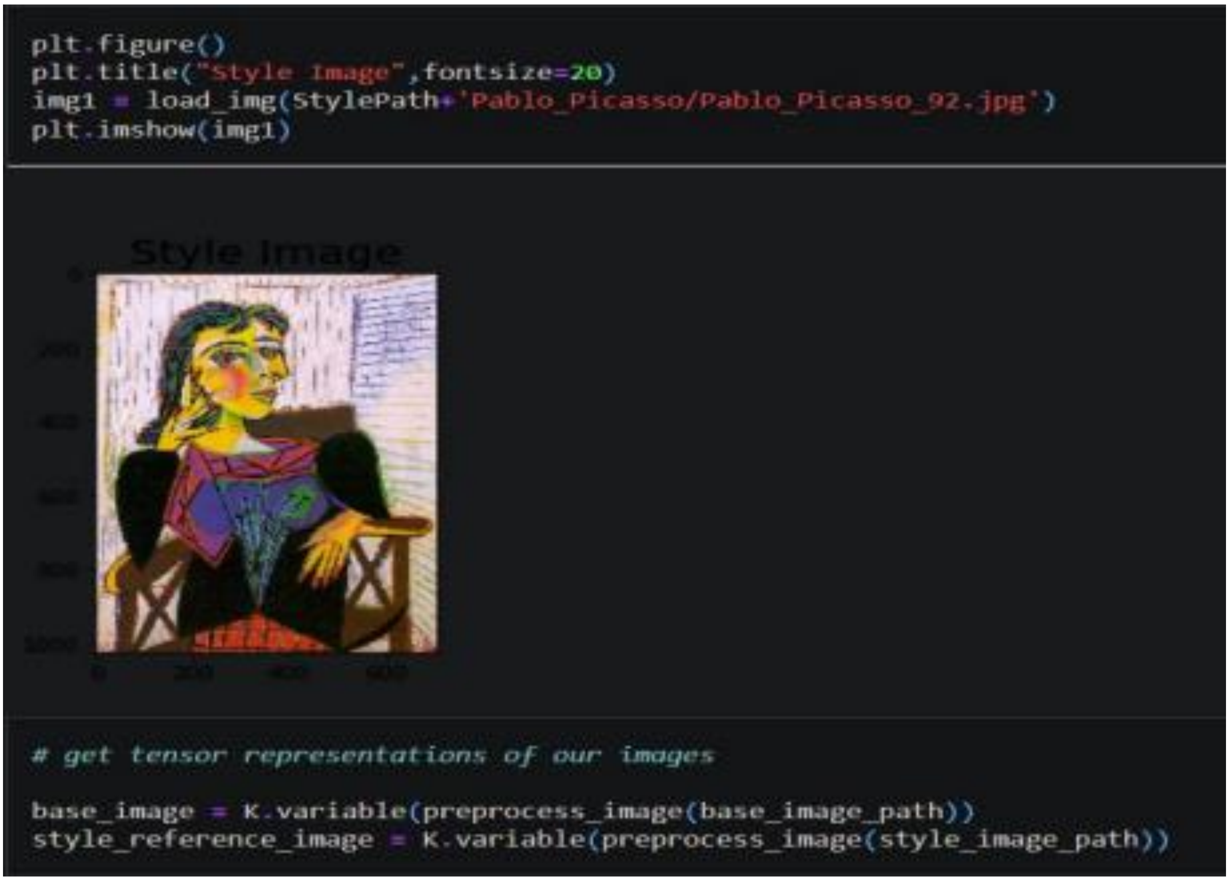
BASE IMAGE:



```python
def preprocess_image(image_path):
    from keras.applications import vgg19
    img = load_img(image_path, target_size=(img_nrows, img_ncols))
    img = img_to_array(img)
    img = np.expand_dims(img, axis=0)
    img = vgg19.preprocess_input(img)
    return img
```

```python
plt.figure()
plt.title("Base Image",fontsize=20)
img1 = load_img(ContentPath+'13.jpg')
plt.imshow(img1)
```

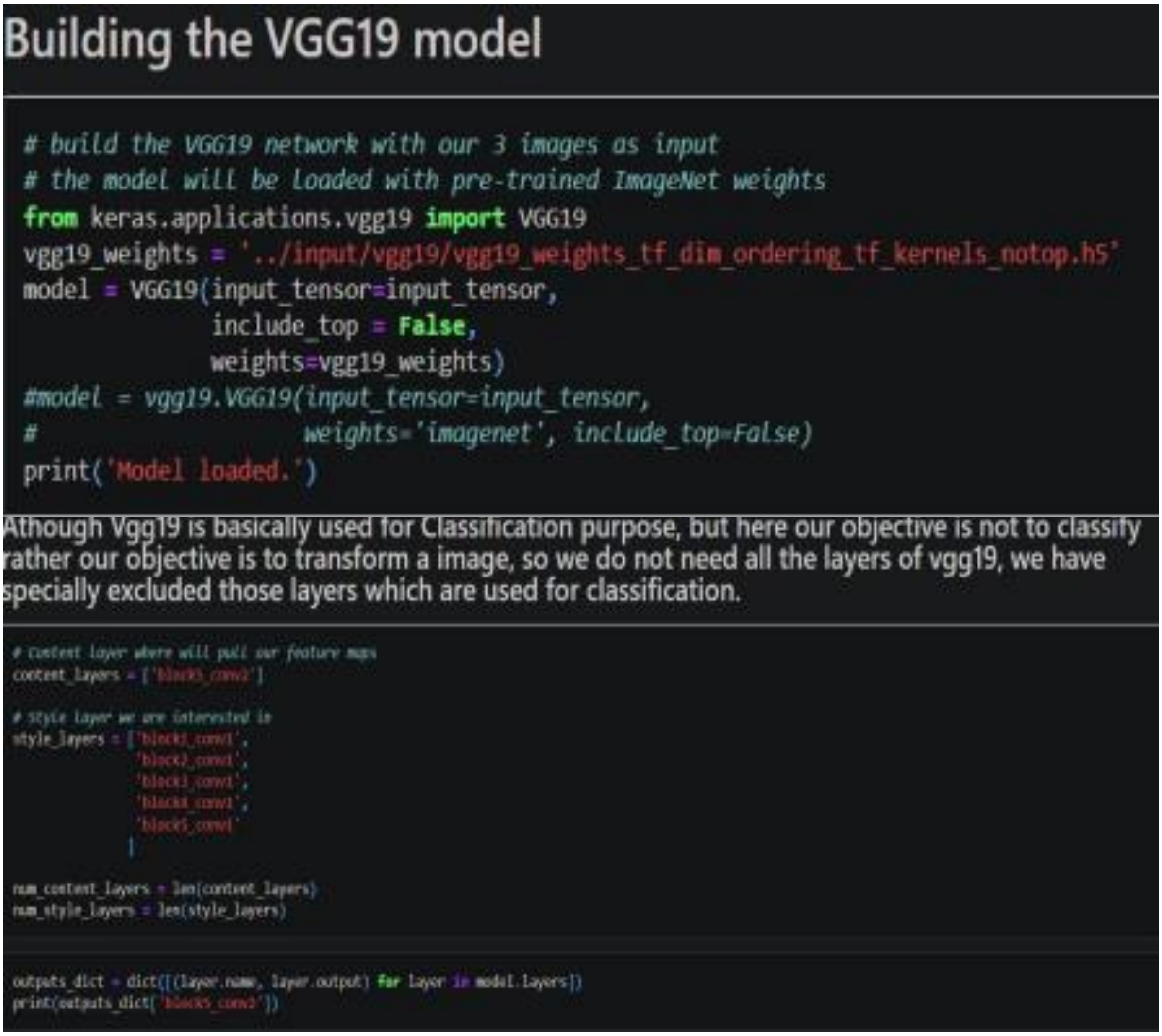
STYLE IMAGE:



```
plt.figure()
plt.title("Style Image",fontsize=20)
img1 = load_img(StylePath+'Pablo_Picasso/Pablo_Picasso_92.jpg')
plt.imshow(img1)
```

```
# get tensor representations of our images

base_image = K.variable(preprocess_image(base_image_path))
style_reference_image = K.variable(preprocess_image(style_image_path))
```

ALGORITHMS IN BETWEEN:



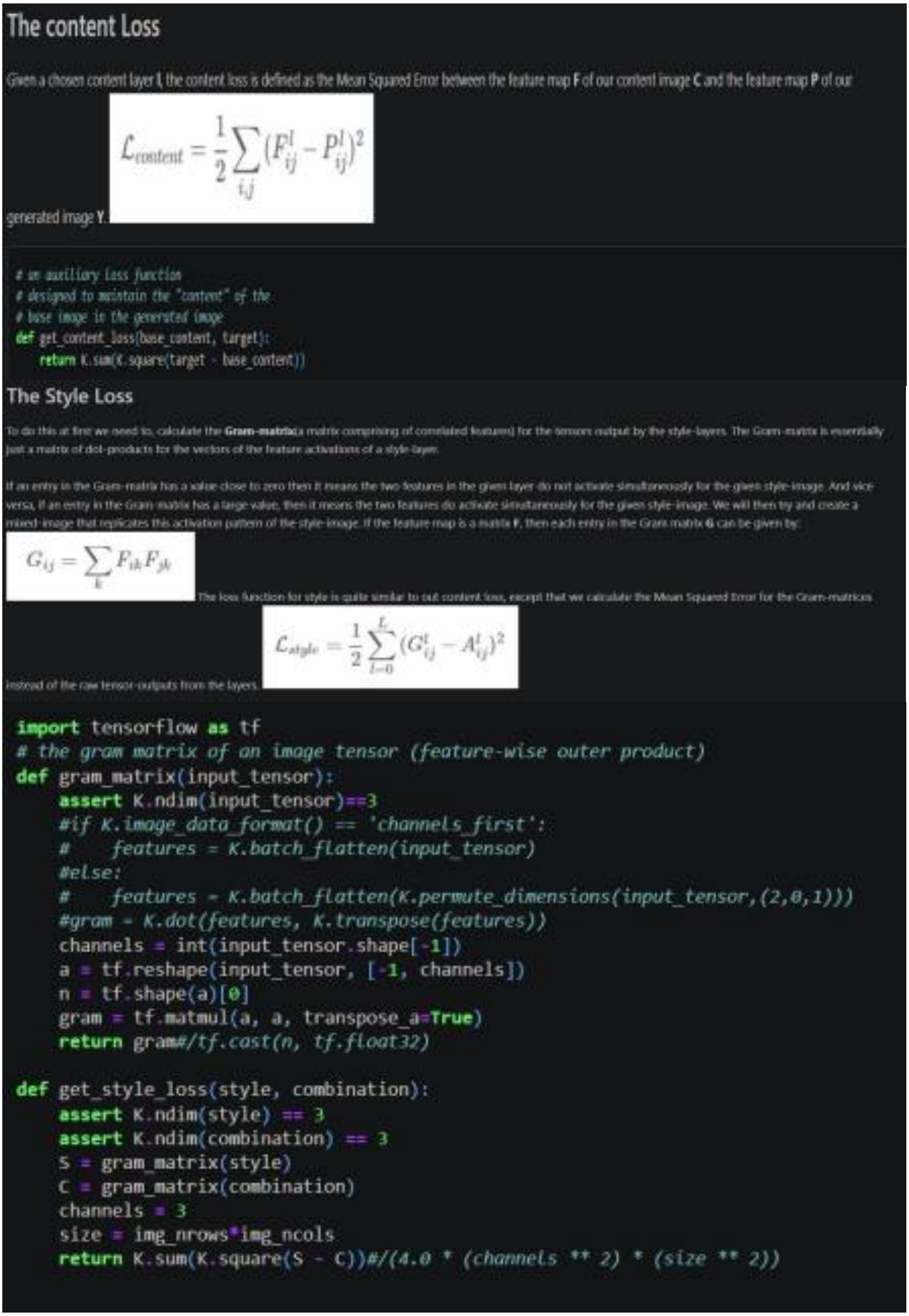
Building the VGG19 model

```
# build the VGG19 network with our 3 images as input
# the model will be loaded with pre-trained ImageNet weights
from keras.applications.vgg19 import VGG19
vgg19_weights = '../input/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5'
model = VGG19(input_tensor=input_tensor,
              include_top = False,
              weights=vgg19_weights)
#model = vgg19.VGG19(input_tensor=input_tensor,
#                    weights='imagenet', include_top=False)
print('Model loaded.')
```

Athough Vgg19 is basically used for Classification purpose, but here our objective is not to classify rather our objective is to transform a image, so we do not need all the layers of vgg19, we have specially excluded those layers which are used for classification.

```
# Content layer where will pull our feature maps
content_layers = ['block5_conv2']

# Style layer we are interested in
style_layers = ['block1_conv1',
                'block2_conv1',
                'block3_conv1',
                'block4_conv1',
                'block5_conv1'
               ]

num_content_layers = len(content_layers)
num_style_layers = len(style_layers)
```

```
outputs_dict = dict([(layer.name, layer.output) for layer in model.layers])
print(outputs_dict['block5_conv2'])
```



The content Loss

Given a chosen content layer **l**, the content loss is defined as the Mean Squared Error between the feature map **F** of out content image **C** and the feature map **P** of our generated image **Y**.

$$\mathcal{L}_{content} = \frac{1}{2}\sum_{i,j}(F^l_{ij} - P^l_{ij})^2$$

```
# an auxiliary loss function
# designed to maintain the "content" of the
# base image in the generated image
def get_content_loss(base_content, target):
    return K.sum(K.square(target - base_content))
```

The Style Loss

To do this at first we need to, calculate the **Gram-matrix**(a matrix comprising of correlated features) for the tensors output by the style-layers. The Gram-matrix is essentially just a matrix of dot-products for the vectors of the feature activations of a style-layer.

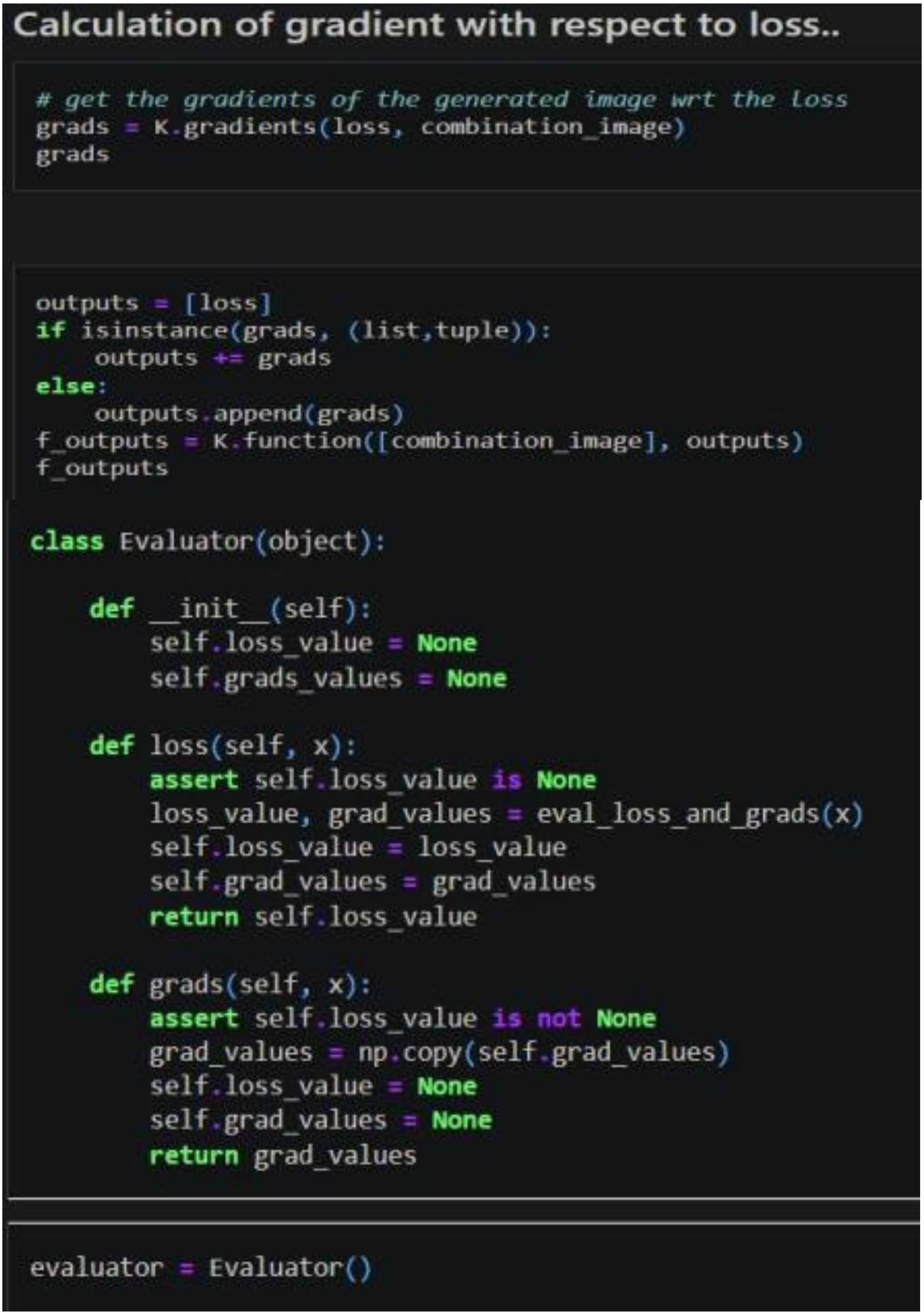
If an entry in the Gram-matrix has a value close to zero then it means the two features in the given layer do not activate simultaneously for the given style-image. And vice versa, if an entry in the Gram-matrix has a large value, then it means the two features do activate simultaneously for the given style-image. We will then try and create a mixed-image that replicates this activation pattern of the style-image. If the feature map is a matrix **F**, then each entry in the Gram matrix **G** can be given by:

$$G_{ij} = \sum_k F_{ik}F_{jk}$$

The loss function for style is quite similar to out content loss, except that we calculate the Mean Squared Error for the Gram-matrices instead of the raw tensor-outputs from the layers.

$$\mathcal{L}_{style} = \frac{1}{2}\sum_{l=0}^{L}(G^l_{ij} - A^l_{ij})^2$$

```
import tensorflow as tf
# the gram matrix of an image tensor (feature-wise outer product)
def gram_matrix(input_tensor):
    assert K.ndim(input_tensor)==3
    #if K.image_data_format() == 'channels_first':
    #    features = K.batch_flatten(input_tensor)
    #else:
    #    features = K.batch_flatten(K.permute_dimensions(input_tensor,(2,0,1)))
    #gram = K.dot(features, K.transpose(features))
    channels = int(input_tensor.shape[-1])
    a = tf.reshape(input_tensor, [-1, channels])
    n = tf.shape(a)[0]
    gram = tf.matmul(a, a, transpose_a=True)
    return gram#/tf.cast(n, tf.float32)

def get_style_loss(style, combination):
    assert K.ndim(style) == 3
    assert K.ndim(combination) == 3
    S = gram_matrix(style)
    C = gram_matrix(combination)
    channels = 3
    size = img_nrows*img_ncols
    return K.sum(K.square(S - C))#/(4.0 * (channels ** 2) * (size ** 2))
```



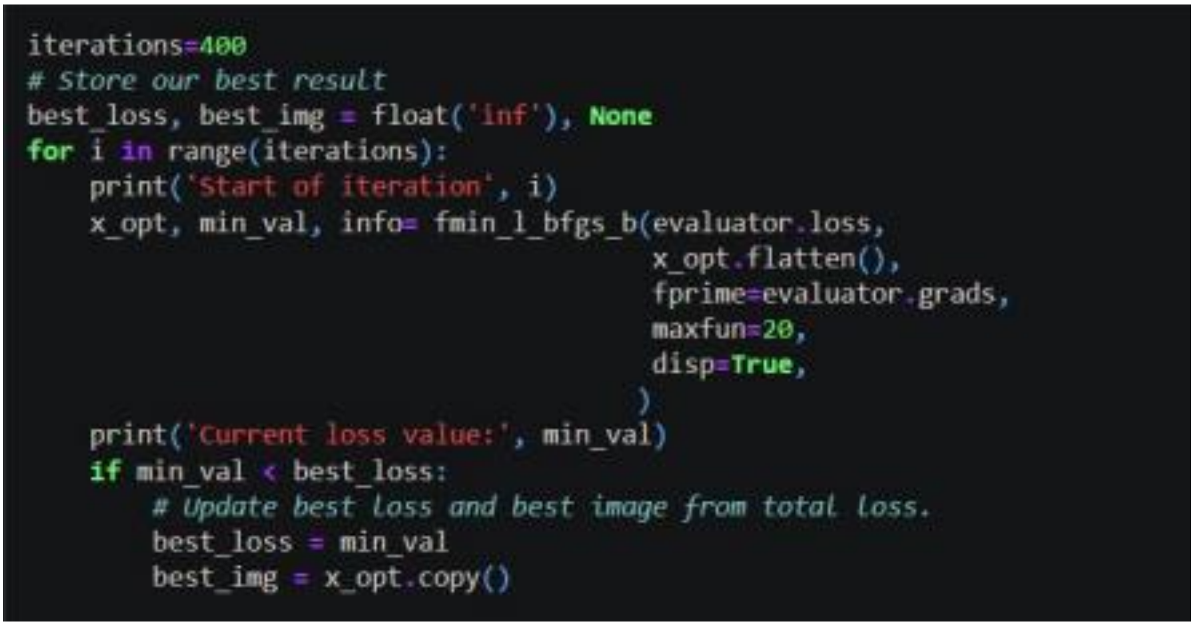
Calculation of gradient with respect to loss..

```python
# get the gradients of the generated image wrt the loss
grads = K.gradients(loss, combination_image)
grads
```

```python
outputs = [loss]
if isinstance(grads, (list,tuple)):
    outputs += grads
else:
    outputs.append(grads)
f_outputs = K.function([combination_image], outputs)
f_outputs
```

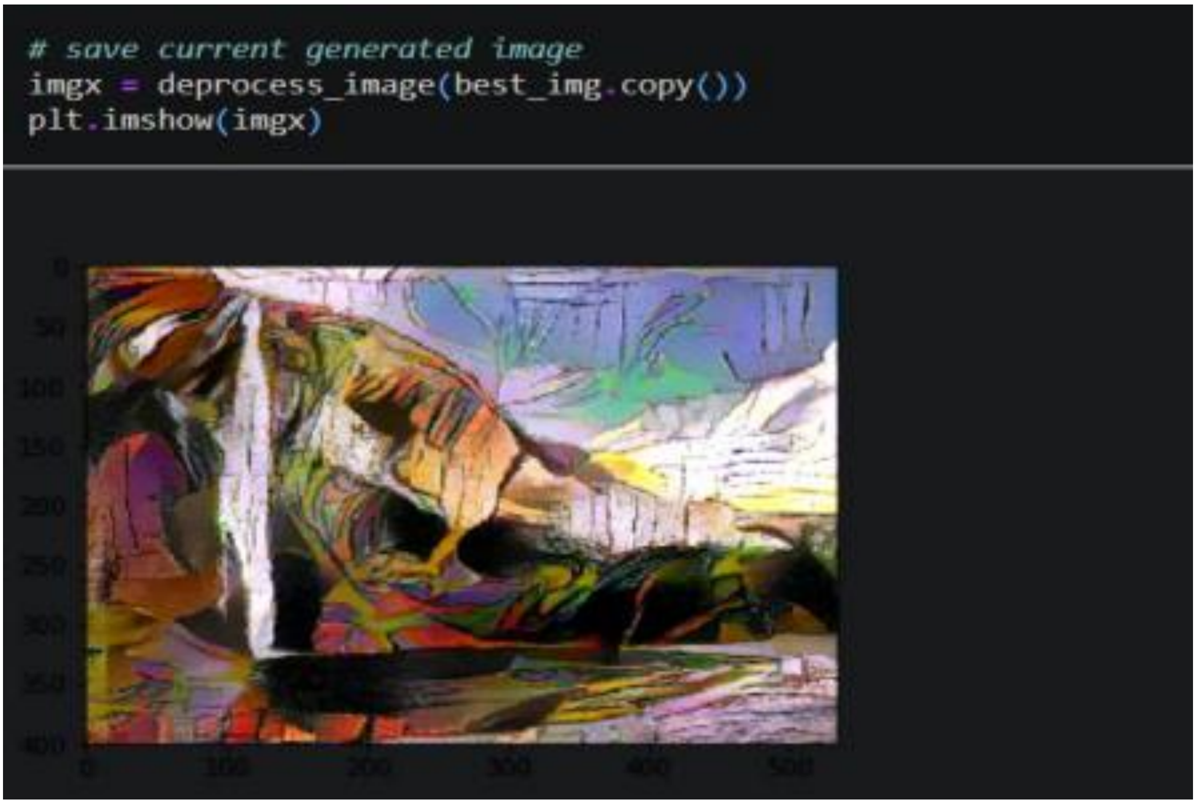
```python
class Evaluator(object):

    def __init__(self):
        self.loss_value = None
        self.grads_values = None

    def loss(self, x):
        assert self.loss_value is None
        loss_value, grad_values = eval_loss_and_grads(x)
        self.loss_value = loss_value
        self.grad_values = grad_values
        return self.loss_value

    def grads(self, x):
        assert self.loss_value is not None
        grad_values = np.copy(self.grad_values)
        self.loss_value = None
        self.grad_values = None
        return grad_values
```

```python
evaluator = Evaluator()
```



```
iterations=400
# Store our best result
best_loss, best_img = float('inf'), None
for i in range(iterations):
    print('Start of iteration', i)
    x_opt, min_val, info= fmin_l_bfgs_b(evaluator.loss,
                                        x_opt.flatten(),
                                        fprime=evaluator.grads,
                                        maxfun=20,
                                        disp=True,
                                       )
    print('Current loss value:', min_val)
    if min_val < best_loss:
        # Update best loss and best image from total loss.
        best_loss = min_val
        best_img = x_opt.copy()
```

FINAL IMAGE:



```
# save current generated image
imgx = deprocess_image(best_img.copy())
plt.imshow(imgx)
```

Conclusion:

The base image is thus style transferred using the style image and hencetheresultant image is obtained.

[PS: This is to a large extent similar to the image to sketch generation, but here the pen style and colours are also considered]