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 COMPUTATIONAL AND TECHNOLOGY

Course Code: 18CSE305J

Course Name: Artificial Intelligence

Course Experiments

Name : HARSHIT AGGARWAL

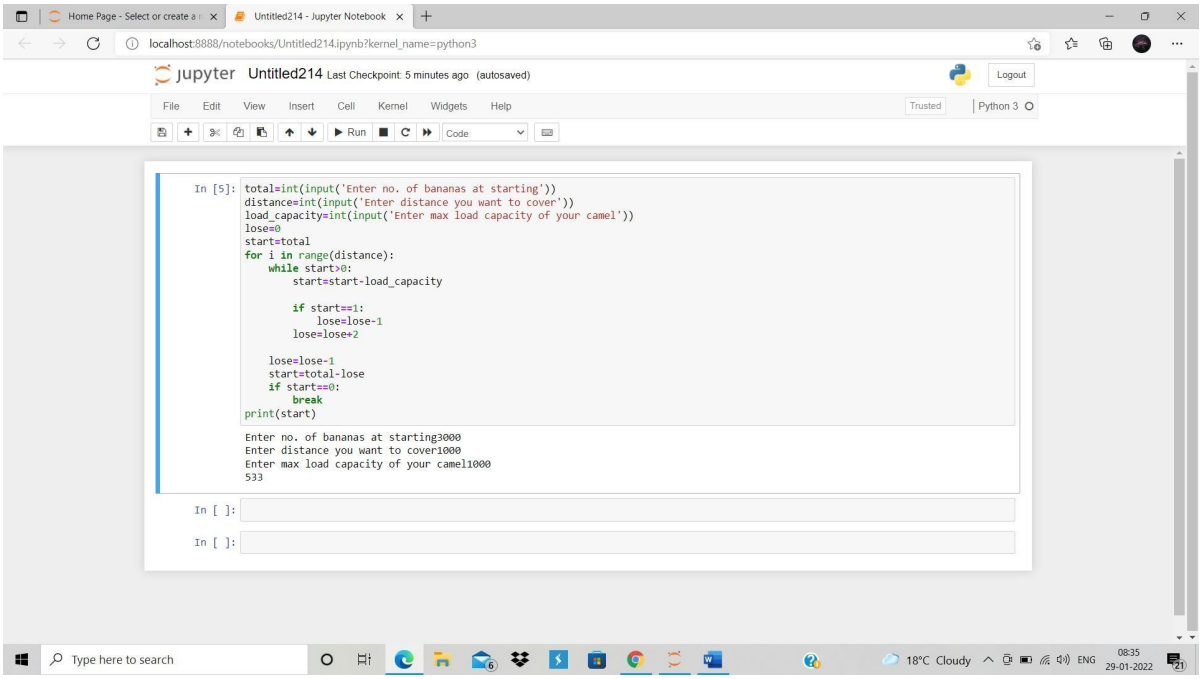
AI LAB1(CAMEL-BANANA PROBLEM)

AIM: To Implement camel-banana problem

CODE:

```
total=int(input('Enter no. of bananas at starting'))
distance=int(input('Enter distance you want to cover'))
load_capacity=int(input('Enter max load capacity of your
camel')) lose=0
start=total
for i in range(distance):
  while start>0:
    start=start-load_capacity

    if start==1:
      lose=lose-
      1
    lose=lose+2

  lose=lose-1
  start=total-lo
  se if start==0:
    break
print(star
t)
```

OUTPUT:



RESULT: Hence Camel-Banana Problem Implemented Successfully

GRAPH COLOURING PROBLEM:

AIM:To Implement Constraint-Satisfaction Problem

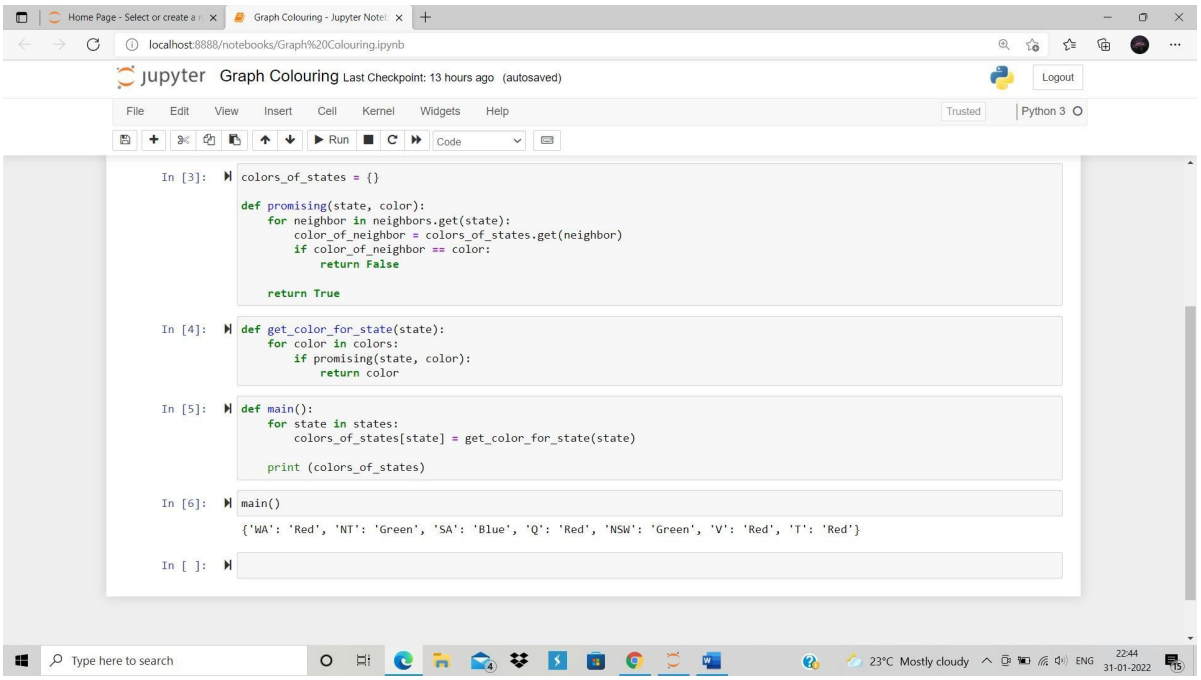
CODE:

```
colors = ['Red', 'Green', 'Blue']


states = ['WA', 'NT', 'SA', 'Q', 'NSW', 'V', 'T']


neighbors = {}

neighbors['WA'] = ['NT',

'SA']

neighbors['NT'] = ['WA', 'SA', 'Q']

neighbors['SA'] = ['WA', 'NT', 'Q', 'NSW', 'V']

neighbors['Q'] = ['NT', 'SA', 'NSW']

neighbors['NSW'] = ['SA', 'Q', 'V']

neighbors['V'] = ['SA', 'NSW']

neighbors['T'] = []


colors_of_states = {}


def promising(state, color):

  for neighbor in neighbors.get(state):

    color_of_neighbor =

    colors_of_states.get(neighbor) if

    color_of_neighbor == color:

      return False


  return True
```

```
def
    get_color_for_state(state
    ): for color in colors:
```

```
    if promising(state, color):
        return color


def main():
  for state in states:
    colors_of_states[state] = get_color_for_state(state)


  print (colors_of_states)



main()
```

OUTPUT:



RESULT: Hence Constraint-satisfaction problem Implemented Sucessfully

harshit aggarwal
(RA1911003010782)

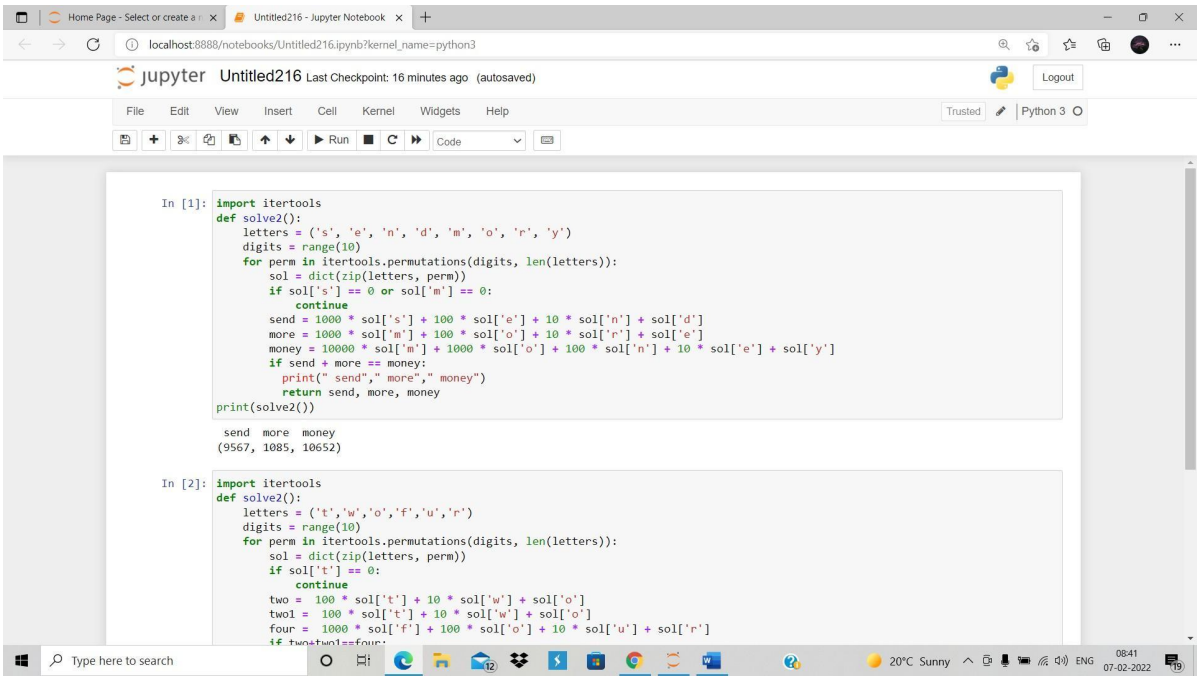
Cryptoarithmetic Puzzle

AIM:To implement Cryptoarithmetic Puzzle

CODE:

```
import
itertools def
solve2():
   letters = ('s', 'e', 'n', 'd', 'm', 'o', 'r', 'y')
   digits = range(10)
   for perm in itertools.permutations(digits, len(letters)):
      sol = dict(zip(letters, perm))
      if sol['s'] == 0 or sol['m'] ==
      0:
         continue
      send = 1000 * sol['s'] + 100 * sol['e'] + 10 * sol['n'] + sol['d']
      more = 1000 * sol['m'] + 100 * sol['o'] + 10 * sol['r'] + sol['e']
      money = 10000 * sol['m'] + 1000 * sol['o'] + 100 * sol['n'] + 10 * sol['e'] + sol['y']
      if send + more == money:
        print(" send"," more"," money")
        return send, more, money
print(solve2())
```

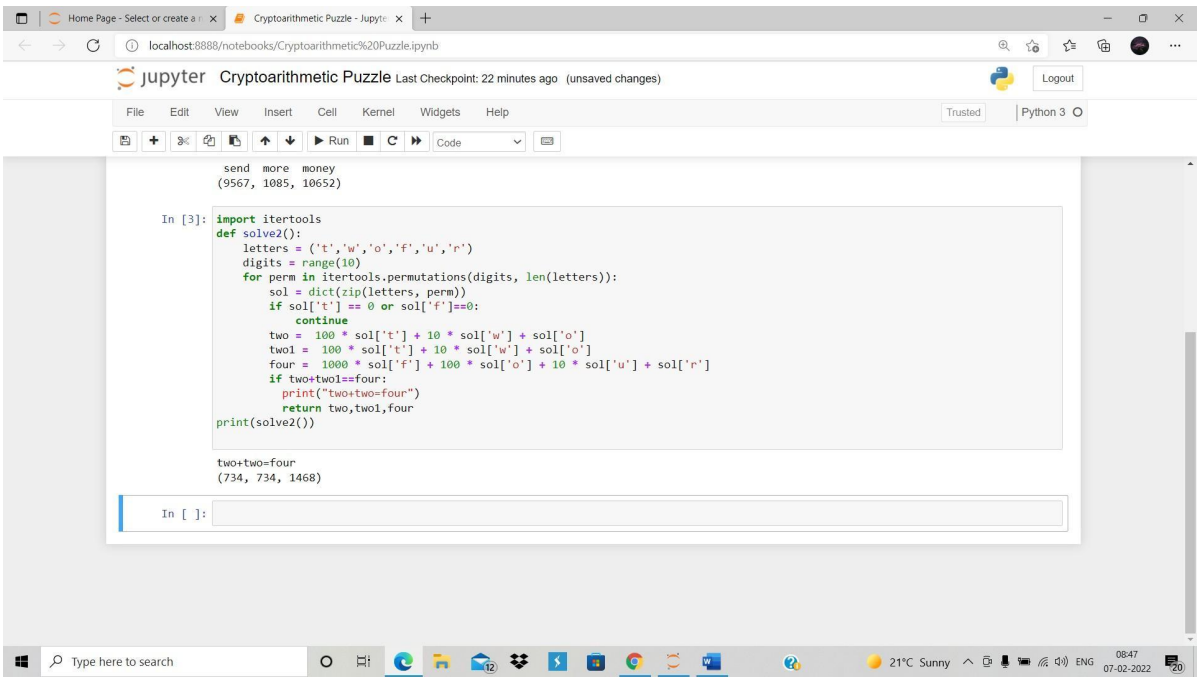
OUTPUT:SEND+MORE=MONEY:



2) TWO+TWO=FOU R: CODE:

```
import itertools
```

```
def solve2():
   letters = ('t','w','o','f','u','r')
   digits = range(10)
   for perm in itertools.permutations(digits, len(letters)):
      sol = dict(zip(letters, perm))
      if sol['t'] == 0:
         continue
      two = 100 * sol['t'] + 10 * sol['w'] + sol['o']
      two1 = 100 * sol['t'] + 10 * sol['w'] + sol['o']
      four = 1000 * sol['f'] + 100 * sol['o'] + 10 * sol['u'] + sol['r']
      if two+two1==four:
        print("two+two=four")
        return two,two1,four
print(solve2())
```



RESULT:Hence Cryptarithmetic Puzzle Impletented Sucessfully

harshit aggarwal
(RA1911003010782)

AI EXPERIMENT 4(DFS-BFS-FOR-ANY-TOY-PROBLEM)

AIM:TO implement DFS-BFS for any toy

problem CODE:

```
from queue import Queue



class NQueens:


    def _init_(self,
        size): self.size =
        size


    def solve_dfs(self):
        if self.size < 1:
            return []
        solutions = []
        stack = [[]]
        while stack:
            solution = stack.pop()
            if self.conflict(solution):
                continue
            row = len(solution)
            if row == self.size:
                solutions.append(solution)
                continue
            for col in range(self.size):
                queen = (row, col)
                queens =
                solution.copy()
                queens.append(queen)
                stack.append(queens)
        return solutions


    def solve_bfs(self):
```

AI EXPERIMENT 4(DFS-BFS-FOR-ANY-TOY-PROBLEM)

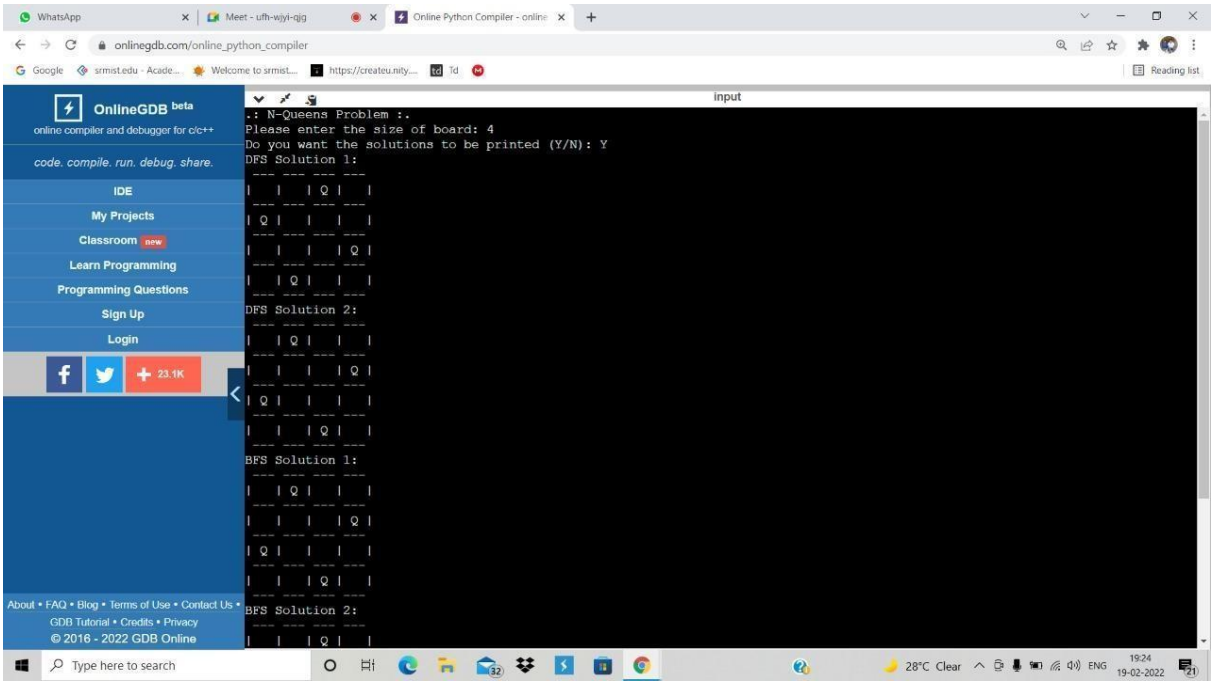
```
    if self.size < 1:
        return []
    solutions = []
    queue = Queue()
    queue.put([])
    while not queue.empty():
        solution = queue.get()
        if
        self.conflict(solution):
            continue
        row = len(solution)
        if row == self.size:
            solutions.append(solution)
            continue
        for col in range(self.size):
            queen = (row, col)
            queens =
            solution.copy()
            queens.append(queen)
            queue.put(queens)
    return solutions


def conflict(self, queens):
    for i in range(1, len(queens)):
        for j in range(0, i):
            a, b = queens[i]
            c, d = queens[j]
            if a == c or b == d or abs(a - c) == abs(b - d):
                return True
    return False


def print(self, queens):
    for i in
    range(self.size):
```

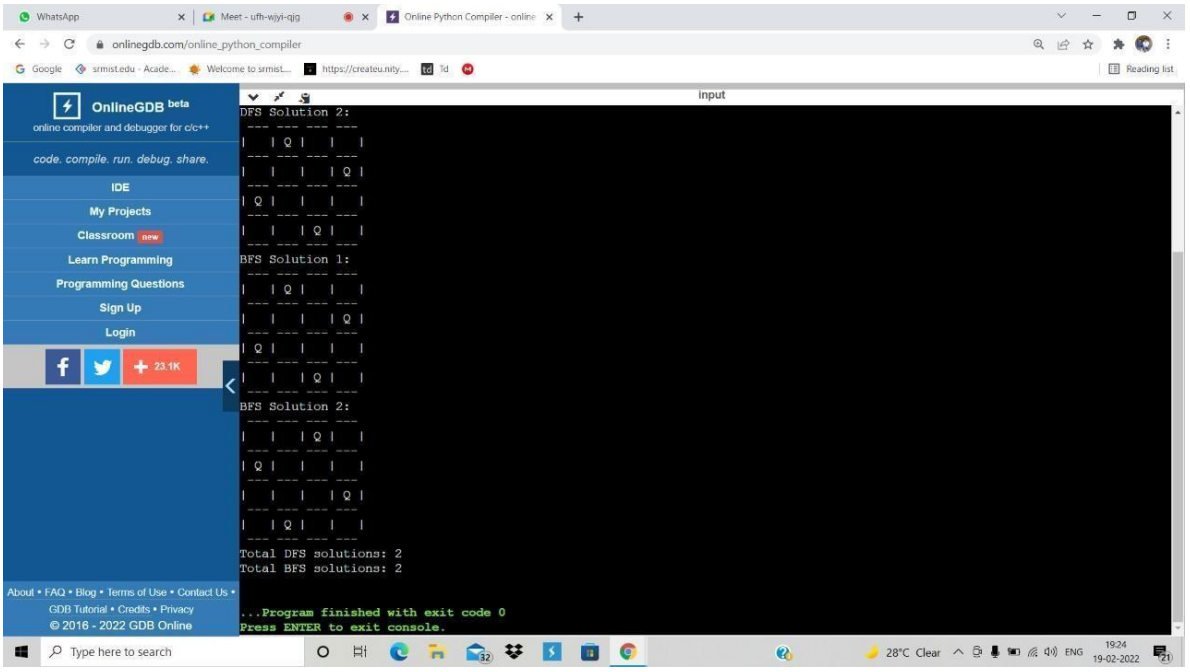
AI EXPERIMENT 4(DFS-BFS-FOR-ANY-TOY-PROBLEM)

```
            print(' ---' * self.size)
            for j in
            range(self.size):
                p = 'Q' if (i, j) in queens else ' '
                print('| %s ' % p, end='')
            print('|')
        print(' ---' *
self.size) def main():
    print('.: N-Queens Problem :.')
    size = int(input('Please enter the size of board: '))
    print_solutions = input('Do you want the solutions to be printed (Y/N): ').lower() ==
    'y' n_queens = NQueens(size)
    dfs_solutions                    =
    n_queens.solve_dfs()    bfs_solutions
    =          n_queens.solve_bfs()          if
    print_solutions:
        for i, solution in enumerate(dfs_solutions):
            print('DFS Solution %d:' % (i + 1))
            n_queens.print(solution)
        for i, solution in enumerate(bfs_solutions):
            print('BFS Solution %d:' % (i + 1))
            n_queens.print(solution)
    print('Total DFS solutions: %d' % len(dfs_solutions))
    print('Total BFS solutions: %d' % len(bfs_solutions))


if_name____== '_main_':
    main()
```

CODE OUPUT:

AI EXPERIMENT 4(DFS-BFS-FOR-ANY-TOY-PROBLEM)





Ex No.5: *Implementation of Best First Search and A* Search for an Application*

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 Priority Queue.

Algorithm :

1) Create an empty
   PriorityQueue
   PriorityQueue **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

A Algorithm*

A heuristic algorithm sacrifices optimality, with precision and accuracy for speed, to solve problems faster and more efficiently.

All graphs have different nodes or points which the algorithm has to take, to reach the final node. The paths between these nodes all have a numerical value, which is considered as the weight of the path. The total of all path's transverse gives you the cost of that route.

Initially, the Algorithm calculates the cost to all its immediate neighbouring nodes, 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 among them. 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 node

h(n) = heuristic approximation of the node's value. This is not a real value but an approximation cost

Algorithm :

1) Make an open list containing starting node . 2)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

3) Else : Put the current node in the list and check its neighbors

4) · 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

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

6) 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:

7) Add it to the open list and set its g

Code :

Best First Search

```
from queue import PriorityQueue v =
5
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, 5)
addedge(0, 2, 1)
addedge(2, 3, 2)
addedge(1, 4, 1)
addedge(3, 4, 2)
source = 0
target = 4 best_first_search(source,
target, v)
```

Output:

In [19]:

```
from queue import PriorityQueue
v = 5
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, 5)
addedge(0, 2, 1)
addedge(2, 3, 2)
addedge(1, 4, 1)
addedge(3, 4, 2)
source = 0
target = 4
best_first_search(source, target, v)
```

```
0 2 3 4
```

In [ ]:

A* Search

```
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': 1,
            'B': 1,
            'C': 1,
            'D': 1
        }

        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', 1), ('C', 3), ('D', 7)],
    'B': [('D', 5)],
    'C': [('D', 12)]
}
graph1 = Graph(adjacency_list)
graph1.a_star_algorithm('A', 'D')
```

Output:

```
                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', 1), ('C', 3), ('D', 7)],
    'B': [('D', 5)],
    'C': [('D', 12)]
}
graph1 = Graph(adjacency_list)
graph1.a_star_algorithm('A', 'D')
```

```
Path found: ['A', 'B', 'D']
Out[18]: ['A', 'B', 'D']
```

```
In [ ]:
```

Result :

A* and best first search algorithms were implemented successfully.

Ex No.6: *Implementation of uncertain methods for an Application (Fuzzy Logic)*

Aim:

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

Algorithm:

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

Optimization Technique:

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

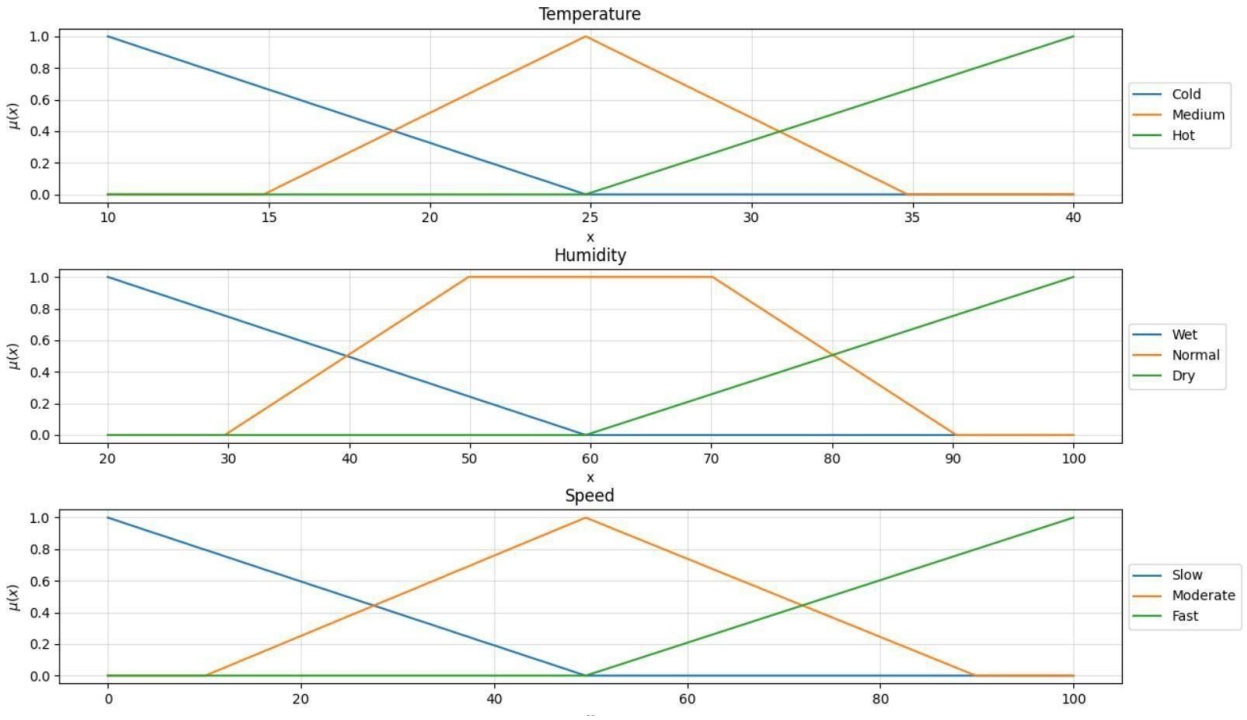
Code:

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

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

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

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

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

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

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

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

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

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

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

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

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

Output:



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

Ex No.7: *Implementation of Unification and Resolution Algorithm*

Unification

In logic and computer science, unification is a process of automatically solving equations between symbolic terms. Unification has several interesting applications, notably in logic programming. Unification is just like pattern matching, except that both terms can contain variables. So, we can no longer say one is the pattern term and the other the constant term. For example:

- First term: f(a, V, bar(D))
- Second term f(D, k, bar(a))

Given two such terms, finding a variable substitution that will make them equivalent is called unification. In this case the substitution is {D=a, V=k}.

Note that there is an infinite number of possible unifiers for some solvable unification problem. For example, given:

- First term: f(X, Y)
- Second term: f(Z, g(X))

We have the substitution {X=Z, Y=g(X)} but also something like {X=K, Z=K, Y=g(K)} and {X=j(K), Z=j(K), Y=g(j(K))} and so on. The first substitution is the simplest one, and also the most general. It's called the most general unifier or most general unifier (MGU) . Intuitively, the most general unifier (MGU) can be turned into any other unifier by performing another substitution. For example {X=Z, Y=g(X)} can be turned into {X=j(K), Z=j(K), Y=g(j(K))} by applying the substitution {Z=j(K)} to it. Note that the reverse doesn't work, as we can't turn the second into the first by using a substitution. So, we say that {X=Z, Y=g(X)} is the most general unifier for the two given terms, and it's the most general unifier (MGU) we want to find.

Algorithm :

```
1: procedure Unify(t1,t2)
2:     Inputs
3:         t1,t2: atoms or terms
4:     Output
5:         most general unifier of t1 and t2 if it exists or ⊥ otherwise
6:     Local
7:         E: a set of equality statements
8:         S: substitution
9:     E←{t1=t2}
10:     S={}
11:     while E≠{} do
12:         select and remove α=β from
E 13:       if β is not identical to α then
14:             if α is a variable then
15:                 replace α with β everywhere in E and S
16:                 S←{α/β}∪S
17:             else if β is a variable then
18:                 replace β with α everywhere in E and S
19:                 S←{β/α}∪S
20:             else if α is p(α1,…,αn) and β is p(β1,…,βn) then
21:                 E←E∪{α1=β1,…,αn=βn}
22:             else
23:                 return ⊥
24:     return S
```

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 == '(' or i == ')':
            return False

    return True


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

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

    return predicate_symbol, arg_list


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

    flag = True
    while flag:
```

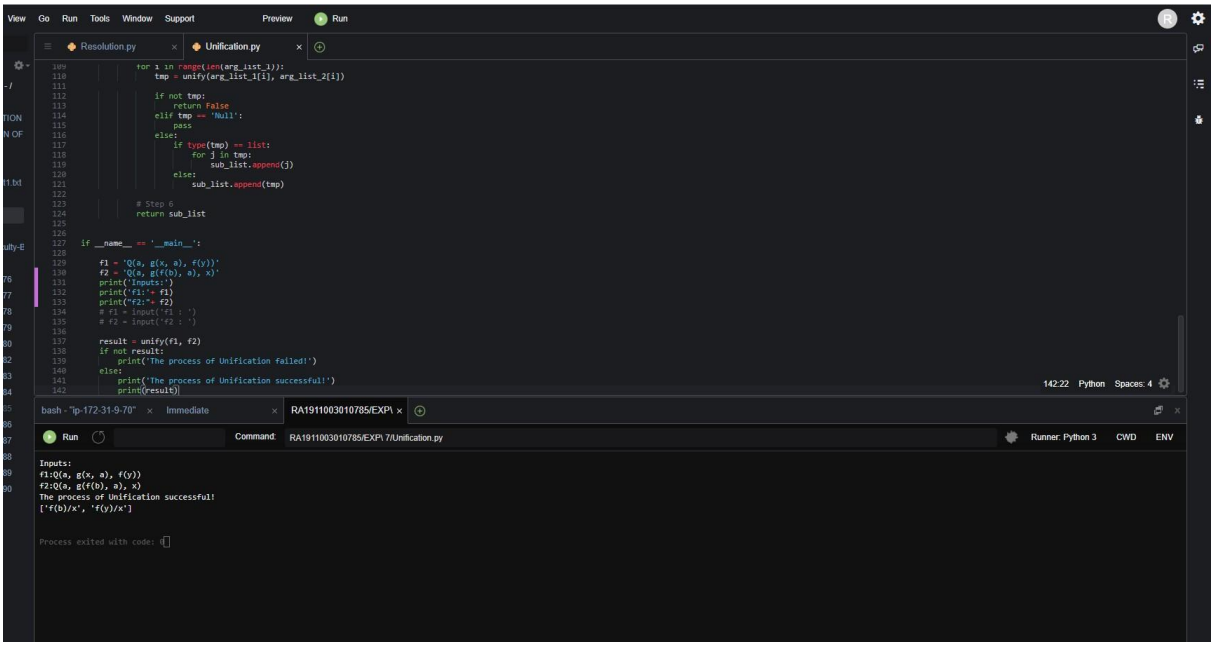
```
        flag = False

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

    return arg_list


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

    return False


def unify(expr1, expr2):

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

```
                tmp
                =
            else: str(
                exp
                r1)
                + '/'
                +
                str(
                exp
                r2)
                retu
                rn
                tmp

predicate_symbol_1, arg_list_1 = process_expression(expr1)
predicate_symbol_2, arg_list_2 = process_expression(expr2)

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

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

        if not tmp: return
            False
        elif tmp == 'Null':
            pass
        else:
            if type(tmp) == list:
                for j in tmp:
                    sub_list.append(j)
            else:
                sub_list.append(tmp)

    # Step 6 return
    sub_list
```

```
if __name__ == '__main__':

    f1 = 'Q(a, g(x, a), f(y))'
    f2 = 'Q(a, g(f(b), a), x)'
```

```
print('Inputs:')
print('f1:'+ f1)
print("f2:"+ f2)
# f1 = input('f1 : ')
# f2 = input('f2 : ')

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

Output:



Resolution

Resolution method is an inference rule which is used in both Propositional as well as First-order Predicate Logic in different ways. This method is basically used for proving the

satisfiability of a sentence. In resolution method, we use Proof by Refutation technique to prove the given statement.
The key idea for the resolution method is to use the knowledge base and negated goal to obtain null clause (which indicates contradiction). Resolution method is also called Proof by Refutation. Since the knowledge base itself is consistent, the contradiction must be introduced by a negated goal. As a result, we have to conclude that the original goal is true.

Algorithm :

1. Convert the given axiom into clausal form, i.e., disjunction form.
2. Apply and proof the given goal using negation rule.
3. Use those literals which are needed to prove.
4. Solve the clauses together and achieve the goal.

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

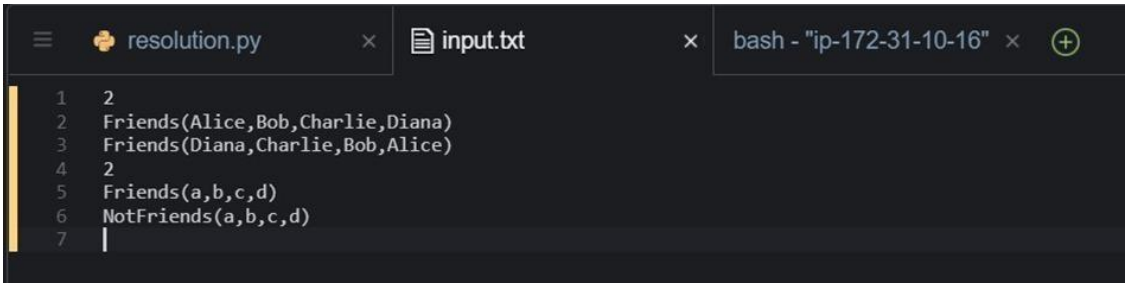
```
        else:
            results.append("FALSE")

    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

                                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 "Constant", new)

                                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_ind
                                                                            ex] = True

                                                                        if
                                                                            self.resolve(newQueryStac
                                                                            k, 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('C:/shushrut/studies/SRM University/SEM
6/AI/7-Unification Resolutiion/Resolution/Input/input_1.txt')
    knowledgeBase = KB(inputSentences_)
    knowledgeBase.prepareKB()
    results_ = knowledgeBase.askQueries(inputQueries_)
    printOutput("output.txt", results_)
```

Output:



Result :

Unification and resolution were implemented successfully.

Artificial Inteligence

Lab Exercise 8 Machine Learning Algorithm

harshit

aggarwal

RA19110030107

82

Aim :

To implement any learning algorithm (classification/ regression/ clustering etc.) usingclassical Machine learning technique.

Problem Statement

Dataset: The dataset Belongs to classic UCI Machine Learning Repository Given

Breast Cancer Dataset

Features related to the Breast cancer

Aim is to predict whether the Tumor is Benign or Malignant

Divided into two classes where 2 - Benign and 4- Malignant

Method

Training the dataset with different Machine Learning models and Concluding which Model Gives the Highest Accuracy

Algorithm :

Importing Libraries

Data Preprocessing

Splitting Data into test set and training set

Feature Scaling

Training data in Random forest classification

Predict for a single value

Classification

it also can Reduce the Accuracy of the model

X is the Independent Variable and Y is the Dependent Variable

Printing the Confusion Matrix

Now that we know Random Forest algorithm gives highest accuracy , trying to predict the class .
The class is predicted according to the values of the repective features
Therefore it has predicted that for these set of feature values the tumor is going to be Benign i.e class 2

Program:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

dataset = pd.read_csv('Breast Cancer
Dataset.csv') dataset.drop('Sample code number',
  axis='columns', inplace=True)
#Dropping the Sample Code Number as it has no influence over the
Classification #it also can Reduce the Accuracy of the model

X = dataset.iloc[:,
:-1].values y =
dataset.iloc[:, -1].values
#X is the Independent Variable and Y is the Dependent Variable
dataset.head()

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
from sklearn.preprocessing import
StandardScaler sc = StandardScaler()
X_train =

sc.fit_transform(X_train) X_test

= sc.transform(X_test)

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, accuracy_score
classifier_forest = RandomForestClassifier(n_estimators = 10, criterion
= 'entropy')
classifier_forest.fit(X_train, y_train)
y_pred = classifier_forest.predict(X_test)
cm_forest = confusion_matrix(y_test,
y_pred) print(cm_forest)
acc_score_forest = accuracy_score(y_test, y_pred)
##Printing the Confusion Matrix



#To Easily know which has highest accuracy
```

```
x = ["Naive Bayes", "Decision Tree", "Logistic Regression", "K_NN","Random
Forest","SVM","Kernal SVM"]
y = [100*acc_score_bayes,100*acc_score_tree, 100*acc_score_log_reg,
100*acc_score_knn,100*acc_score_forest,100*acc_score_SVM_lin,100*acc_sco
re_SVM_ rbf]
fig = plt.figure()
ax =

fig.add_axes([0,0,1,1])

plt.barh(x, y)

for index, value in
    enumerate(y):
    plt.text(value, index,
             str(value))

plt.show()

#Now that we know Random Forest algorithm gives highest accuracy , trying to
predict the class .
#The class is predicted according to the values of the repective features
print(classifier_forest.predict(sc.transform([[4,8,1,2,2,5,3,2,1]])))
#Therefore it has predicted that for these set of feature values the tumor
is going to be Benign i.e class 2
```

OUTPUT:

	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	Class
0	5	1	1	1	2	1	3	1	1	2
1	5	4	4	5	7	10	3	2	1	2
2	3	1	1	1	2	2	3	1	1	2
3	6	8	8	1	3	4	3	7	1	2
4	4	1	1	3	2	1	3	1	1	2

CONFUSION MATRIX:

```
[[90  1]
 [ 4 42]]
```

Observation:

From the above Accuracy Scores Random Forest Classifier ML model has given the highest Accuracy

Observing the Confusion Matrix

- Out of 91 Dependent Values(class) only 1 value was predicted wrong and 90 values were predicted correctly which in turn gave the high accuraccy

RESULT:

Hence a Classification algorithm was implemented

Artificial Inteligence

Lab Exercise 9

Deep Learning Algorithm

harshit
aggarwal
RA19110030107
82

Aim :

To Apply deep learning methods to solve an application

Algorithm :

Building a Volume Controller with OpenCV can be accomplished with

Importing Libraries

- **CV2**
- **Mediapipe**
- **Math**
- **Ctypes**
- **Comtypes**
- **Pycaw**
- **Numpy**

Step 1. Detect Hand landmarks

Step 2. Calculate the distance between thumb tip and index finger tip.

Step 3. Map the distance of thumb tip and index finger tip with volume range.

Step 4. For my case, distance between thumb tip and index finger tip was within the range of 15 – 220 and the volume range was from -63.5 – 0.0.

Program:

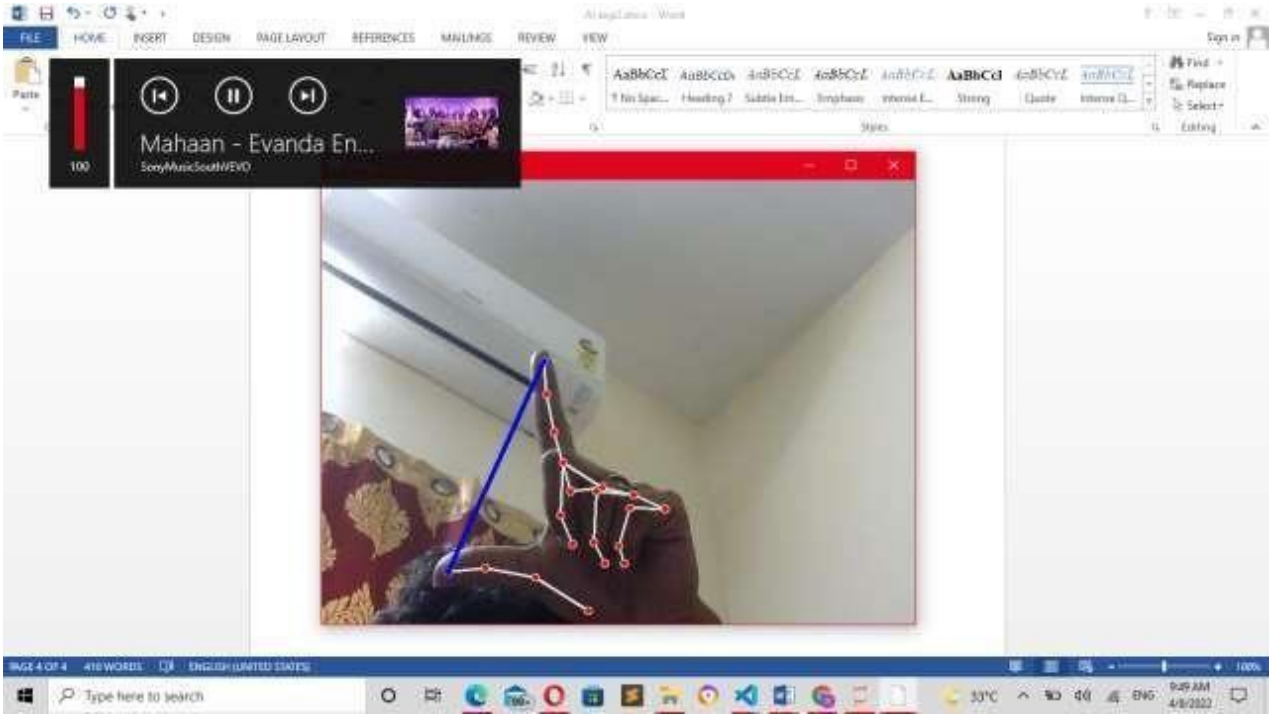
```
import cv2
import mediapipe
as mp from math
import hypot

POINTER from comtypes
import CLSCTX_ALL
```

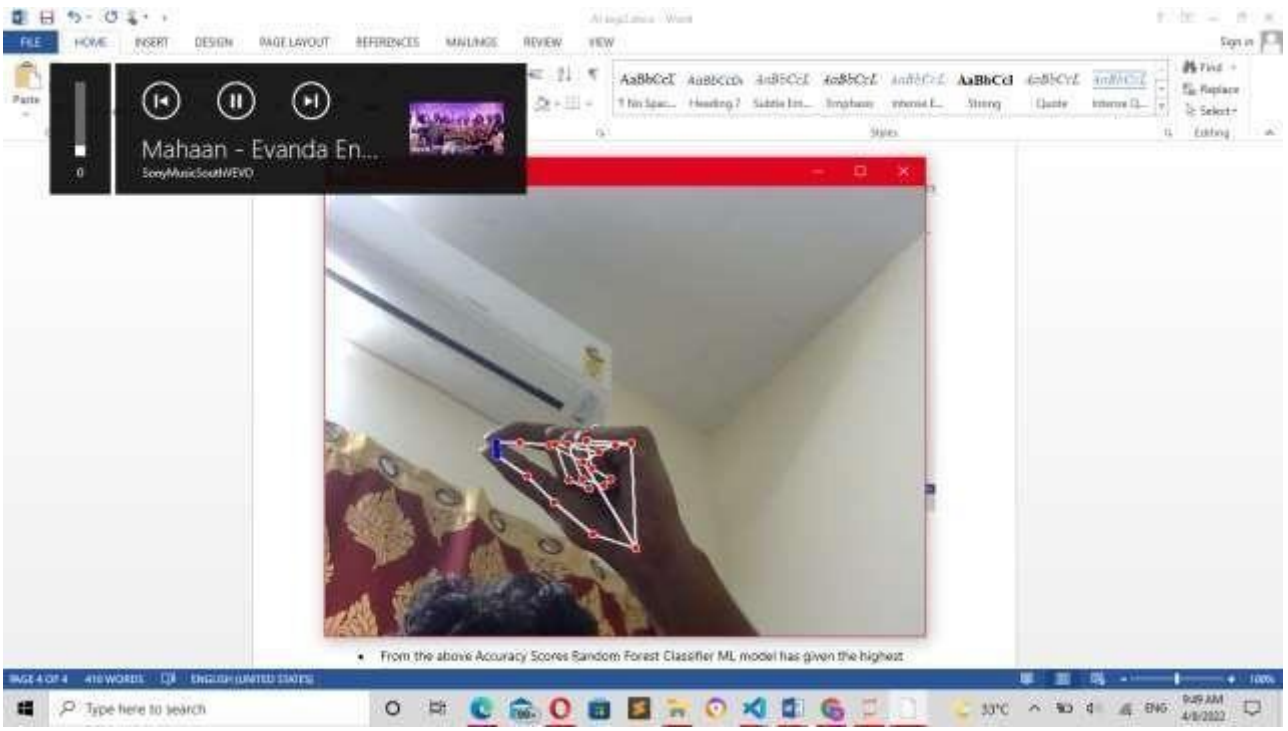
```
from pycaw.pycaw import AudioUtilities,
IAudioEndpointVolume import numpy as np

cap = cv2.VideoCapture(0)

mpHands =
mp.solutions.hands
hands =
mpHands.Hands()
mpDraw = mp.solutions.drawing_utils

devices = AudioUtilities.GetSpeakers()
interface = devices.Activate(IAudioEndpointVolume._iid_,
CLSCTX_ALL, None) volume = cast(interface,
POINTER(IAudioEndpointVolume))
volMin,volMax =


volume.GetVolumeRange()[:2] while


True:


    success,img = cap.read()
    imgRGB =
    cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
    results = hands.process(imgRGB)

    lmList = []
    if results.multi_hand_landmarks:
        for handlandmark in
            results.multi_hand_landmarks: for id,lm
            in enumerate(handlandmark.landmark):
                h,w,_ = img.shape
                cx,cy =
                int(lm.x*w),int(lm.y*h
                )
                lmList.append([id,cx,c
                y])


mpDraw.draw_landmarks(img,handlandmark,mpHands.HA
ND_CONNECTION S)
```

```
if lmList != []:
    x1,y1 = lmList[4][1],lmList[4][2]
    x2,y2 =  lmList[8][1],lmList[8][2]

    cv2.circle(img,(x1,y1),4,(255,0,0),cv2.FILLED)
    cv2.circle(img,(x2,y2),4,(255,0,0),cv2.FILLED)
    cv2.line(img,(x1,y1),(x2,y2),(255,0,0),3)
```

```
    length = hypot(x2-x1,y2-y1)

    vol =
    np.interp(length,[15,220],[volMin,v
    olMax]) print(vol,length)
    volume.SetMasterVolumeLevel(vol,
    None)

    # Hand range 15 - 220

    # Volume range -63.5 - 0.0


cv2.imshow('Image',img)
if cv2.waitKey(1) &
    0xff==ord('q'): break
```

OUTPUT:





Observation:

- Gesture recognition helps computers to understand human body language. This helps to build a more potent link between humans and machines, rather than just the basic text user interfaces or graphical user interfaces (GUIs).
- an interface which will capture human hand gesture dynamically and will control the volume level. For this, Deep Learning techniques such as Yolo model, Inception Net model+LSTM, 3-D CNN+LSTM and Time Distributed CNN+LSTM have been studied to compare the results of hand detection.
- The results of Yolo model outperform the other three models. The models were trained using Kaggle and 20% of the videos available in 20 billion jester dataset. After the hand detection in captured frames, the next step is to control the system volume depending on direction of hand movement.

- The hand movement direction is determined by generating and locating the bounding box on the detected hand.

RESULT:

Hence Deep learning method using open CV to control volume was implemented

```
Index(['target', 'ids', 'date', 'flag', 'user', 'text'], dtype='object')
```

```
print('length of data is', len(df))
```

```
length of data is 1271109
```

```
df.info()
```

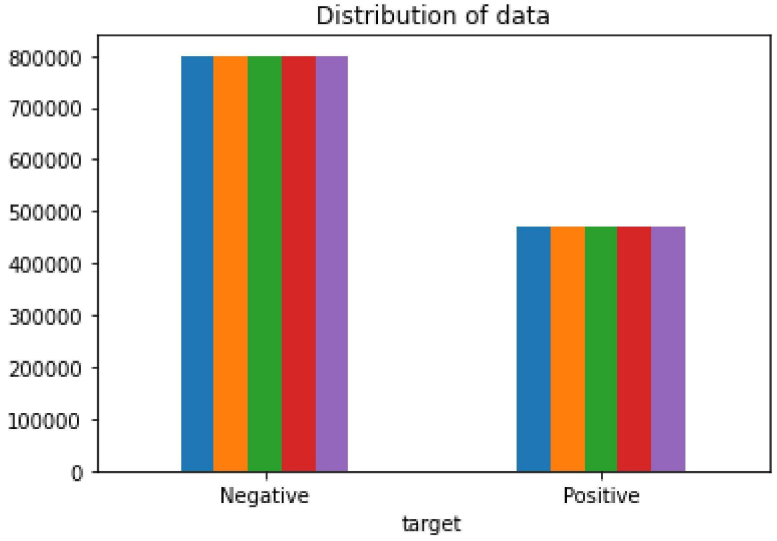
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1271109 entries, 0 to 1271108
Data columns (total 6 columns):
 #   Column  Non-Null Count    Dtype
---  ------  --------------    -----
 0   target  1271109 non-null  int64
 1   ids     1271109 non-null  int64
                               object
df.columns
 3   flag    1271108 non-null  object
 4   user    1271108 non-null  object
 5   text    1271108 non-null  object
dtypes: int64(2), object(4)
memory usage: 58.2+ MB
 2   date    1271109 non-null
```

```
print('Count of columns in the data is:  ', len(df.columns))
print('Count of rows in the data is:  ', len(df))
```

```
Count of columns in the data is:   6
Count of rows in the data is:      1271109
```

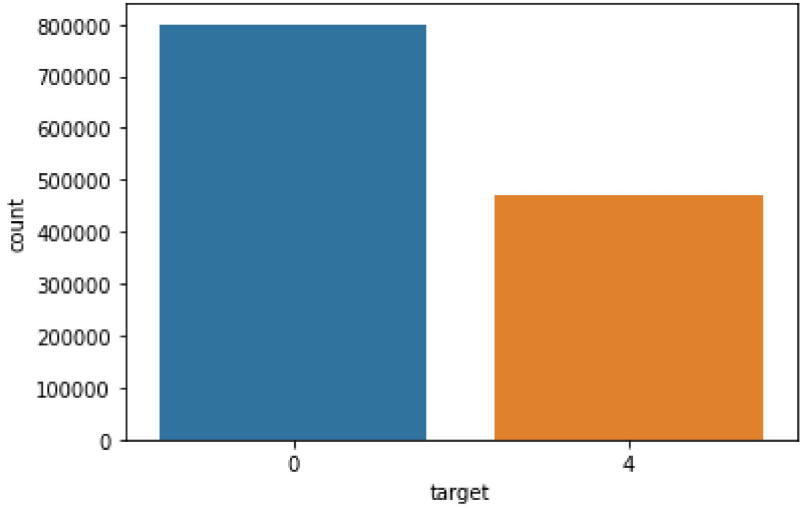
Data Visualisation

```
# Plotting the distribution for dataset.
```




```
import seaborn as sns
sns.countplot(x='target', data=df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff5d4224950>
```




```
data=df[['text','target']]
```

```
data['target'] = data['target'].replace(4,1)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
```

```
        is trying to be set on a copy of
A value Try using .loc[row_indexer,col_indexer] = value instead a slice from a DataFrame.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cop
  """Entry point for launching an IPython kernel.
```

```
data['target'].unique()
```

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

```
data_pos = data[data['target'] == 1]
data_neg = data[data['target'] == 0]
```

```
data_pos = data_pos.iloc[:int(20000)]
data_neg = data_neg.iloc[:int(20000)]
```

```
dataset = pd.concat([data_pos, data_neg])
dataset['text']=dataset['text'].str.lower()
dataset['text'].tail()
```

```
19995    not much time off this weekend, work trip to m...
19996                              one more day of holidays
19997    feeling so down right now .. i hate you damn h...
19998    geez,i hv to read the whole book of personalit...
19999    i threw my sign at donnie and he bent over to ...
Name: text, dtype: object
```

```
stopwordlist = ['a', 'about', 'above', 'after', 'again', 'ain', 'all', 'am', 'an',
             'and','any','are', 'as', 'at', 'be', 'because', 'been', 'before',
             'being', 'below', 'between','both', 'by', 'can', 'd', 'did', 'do',
             'does', 'doing', 'down', 'during', 'each','few', 'for', 'from',
             'further', 'had', 'has', 'have', 'having', 'he', 'her', 'here',
             'hers', 'herself', 'him', 'himself', 'his', 'how', 'i', 'if', 'in',
             'into','is', 'it', 'its', 'itself', 'just', 'll', 'm', 'ma',
             'me', 'more', 'most','my', 'myself', 'now', 'o', 'of', 'on', 'once',
             'only', 'or', 'other', 'our', 'ours','ourselves', 'out', 'own', 're','s', 'same', 'she', "shes", 'should', "shouldve",'so', 'some'
             't', 'than', 'that', "thatll", 'the', 'their', 'theirs', 'them',
             'themselves', 'then', 'there', 'these', 'they', 'this', 'those',
```

```
            'through', 'to', 'too','under', 'until', 'up', 've', 'very', 'was',
            'we', 'were', 'what', 'when', 'where','which','while', 'who', 'whom',
            'why', 'will', 'with', 'won', 'y', 'you', "youd","youll", "youre",
            "youve", 'your', 'yours', 'yourself', 'yourselves']
```

Cleaning and removing stopwords

```
STOPWORDS = set(stopwordlist)
def cleaning_stopwords(text):
    return " ".join([word for word in str(text).split() if word not in STOPWORDS])
dataset['text'] = dataset['text'].apply(lambda text: cleaning_stopwords(text))
dataset['text'].head()
```

```
800000                love @health4uandpets u guys r best!!
800001    im meeting one besties tonight! cant wait!! - ...
800002    @darealsunisakim thanks twitter add, sunisa! g...
800003    sick really cheap hurts much eat real food plu...
800004                      @lovesbrooklyn2 effect everyone
Name: text, dtype: object
```

Cleaning and removing punctuations

```
import string
english_punctuations = string.punctuation
punctuations_list = english_punctuations
def cleaning_punctuations(text):
    translator = str.maketrans('', '', punctuations_list)
    return text.translate(translator)
dataset['text']= dataset['text'].apply(lambda x: cleaning_punctuations(x))
dataset['text'].tail()
```

Cleaning and removing repeating characters

```
def cleaning_repeating_char(text):
    return re.sub(r'(.)1+', r'1', text)
```

```
dataset['text'] = dataset['text'].apply(lambda x: cleaning_repeating_char(x))
dataset['text'].tail()
```

```
19995    not much time off weekend, work trip malmï¿½ f...
19996                                  one day holidays
19997                  feeling right .. hate damn humprey
19998    geez,i hv read whole book personality types em...
19999    threw sign donnie bent over get but thingee ma...
```

```
def cleaning_URLs(data):
    return re.sub('((www.[^s]+)|(https?://[^s]+))',' ',data)
dataset['text'] = dataset['text'].apply(lambda x: cleaning_URLs(x))
dataset['text'].tail()

def cleaning_numbers(data):
    return re.sub('[0-9]+', '', data)
dataset['text'] = dataset['text'].apply(lambda x: cleaning_numbers(x))
dataset['text'].tail()
```

```
19995    not much time off weekend, work trip malmï¿½ f...
19996                                  one day holidays
19997                  feeling right .. hate damn humprey
19998    geez,i hv read whole book personality types em...
19999    threw sign donnie bent over get but thingee ma...
Name: text, dtype: object
```

Getting Tokenization of tweet text

```
from nltk.tokenize import RegexpTokenizer
tokenizer = RegexpTokenizer(r'w+')
dataset['text'] = dataset['text'].apply(tokenizer.tokenize)
dataset['text'].head()
```

```
800000          [w]
800001          [w]
800002    [w, w, w]
800003           []
800004           []
Name: text, dtype: object
```

Applying Stemming

```
import nltk
st = nltk.PorterStemmer()
def stemming_on_text(data):
    text = [st.stem(word) for word in data]
    return data
dataset['text']= dataset['text'].apply(lambda x: stemming_on_text(x))
dataset['text'].head()
```

```
800000          []
800001         [w]
800002    [w, w, w]
800003          []
800004          []
Name: text, dtype: object
```

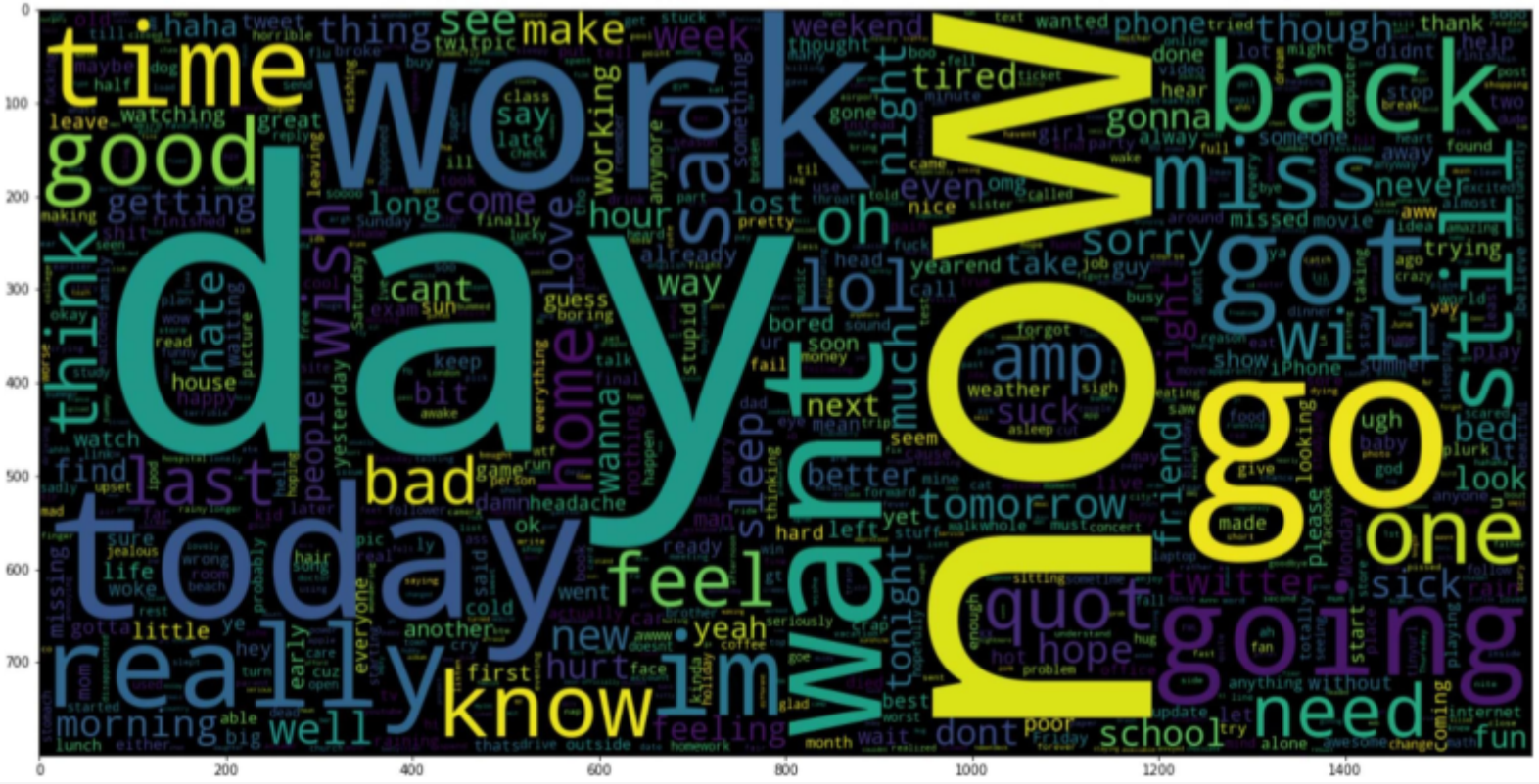
Applying Lemmatizer

```
import nltk
nltk.download('wordnet')
lm = nltk.WordNetLemmatizer()
def lemmatizer_on_text(data):
    text = [lm.lemmatize(word) for word in data]
    return data
dataset['text'] = dataset['text'].apply(lambda x: lemmatizer_on_text(x))
dataset['text'].head()
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Unzipping corpora/wordnet.zip.
800000          []
800001         [w]
800002    [w, w, w]
800003          []
800004          []
Name: text, dtype: object
```

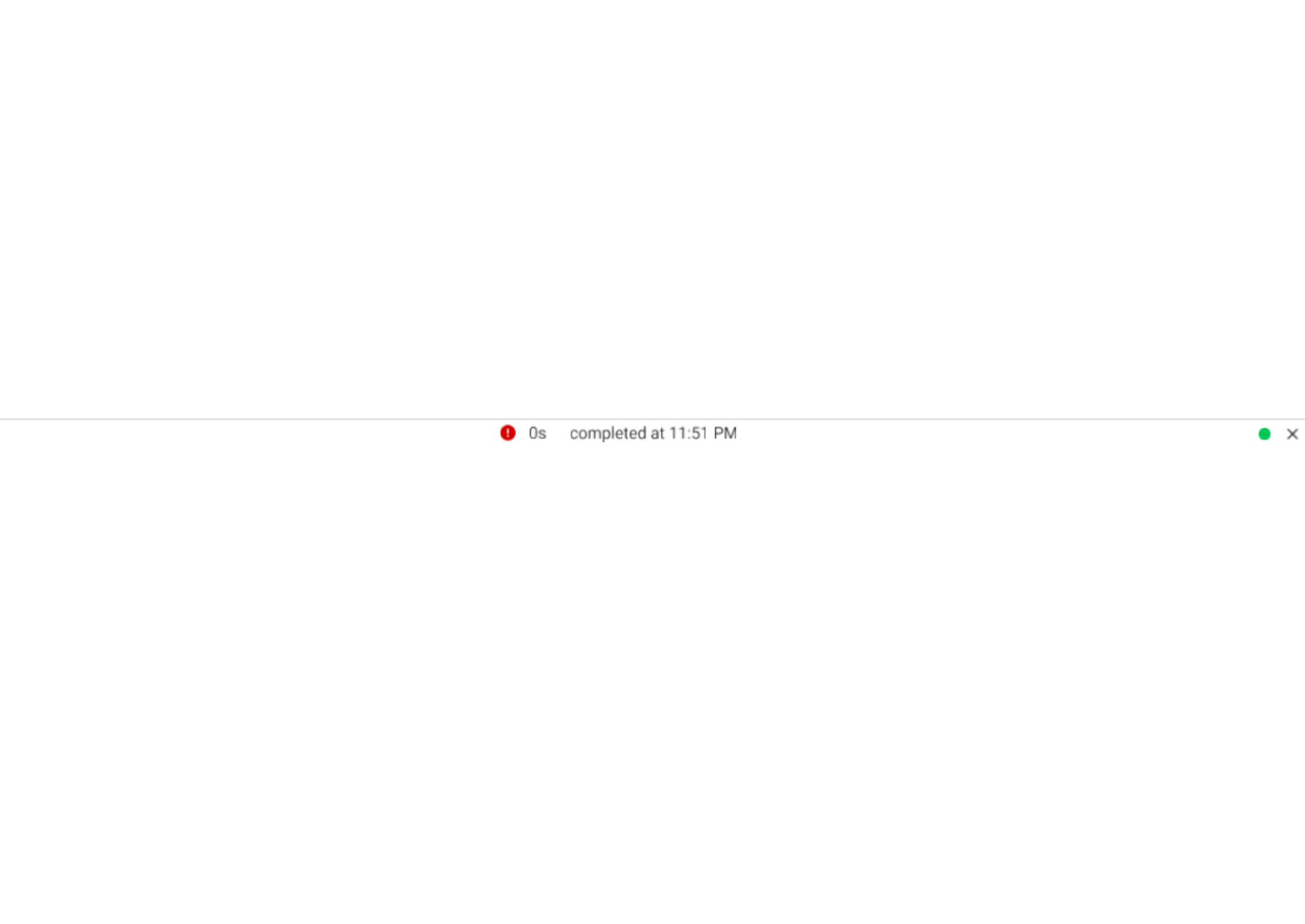
```
X=data.text
y=data.target
```

```
<matplotlib.image.AxesImage at 0x7ff5d40cad10>
```




```
data_neg = data['text'][:800000]
plt.figure(figsize = (20,20))
wc = WordCloud(max_words = 1000 , width = 1600 , height = 800,
               collocations=False).generate(" ".join(data_neg))
plt.imshow(wc)
```

Sharan Prasath S
Sharan Prasath S


0s completed at 11:51 PM