Artificial Intelligence – Experiments

**Exp no 1: Implementation of toy problems(Tic Tac Toe)**

**Link:**

<https://medium.com/byte-tales/the-classic-tic-tac-toe-game-in-python-3-1427c68b8874>

**Source Code:**

#Implementation of Two Player Tic-Tac-Toe game in Python.

''' We will make the board using dictionary

in which keys will be the location(i.e : top-left,mid-right,etc.)

and initialliy it's values will be empty space and then after every move

we will change the value according to player's choice of move. '''

theBoard = {'7': ' ' , '8': ' ' , '9': ' ' ,

'4': ' ' , '5': ' ' , '6': ' ' ,

'1': ' ' , '2': ' ' , '3': ' ' }

board\_keys = []

for key in theBoard:

board\_keys.append(key)

''' We will have to print the updated board after every move in the game and

thus we will make a function in which we'll define the printBoard function

so that we can easily print the board everytime by calling this function. '''

def printBoard(board):

print(board['7'] + '|' + board['8'] + '|' + board['9'])

print('-+-+-')

print(board['4'] + '|' + board['5'] + '|' + board['6'])

print('-+-+-')

print(board['1'] + '|' + board['2'] + '|' + board['3'])

# Now we'll write the main function which has all the gameplay functionality.

def game():

turn = 'X'

count = 0

for i in range(10):

printBoard(theBoard)

print("It's your turn," + turn + ".Move to which place?")

move = input()

if theBoard[move] == ' ':

theBoard[move] = turn

count += 1

else:

print("That place is already filled.\nMove to which place?")

continue

# Now we will check if player X or O has won,for every move after 5 moves.

if count >= 5:

if theBoard['7'] == theBoard['8'] == theBoard['9'] != ' ': # across the top

printBoard(theBoard)

print("\nGame Over.\n")

print(" \*\*\*\* " +turn + " won. \*\*\*\*")

break

elif theBoard['4'] == theBoard['5'] == theBoard['6'] != ' ': # across the middle

printBoard(theBoard)

print("\nGame Over.\n")

print(" \*\*\*\* " +turn + " won. \*\*\*\*")

break

elif theBoard['1'] == theBoard['2'] == theBoard['3'] != ' ': # across the bottom

printBoard(theBoard)

print("\nGame Over.\n")

print(" \*\*\*\* " +turn + " won. \*\*\*\*")

break

elif theBoard['1'] == theBoard['4'] == theBoard['7'] != ' ': # down the left side

printBoard(theBoard)

print("\nGame Over.\n")

print(" \*\*\*\* " +turn + " won. \*\*\*\*")

break

elif theBoard['2'] == theBoard['5'] == theBoard['8'] != ' ': # down the middle

printBoard(theBoard)

print("\nGame Over.\n")

print(" \*\*\*\* " +turn + " won. \*\*\*\*")

break

elif theBoard['3'] == theBoard['6'] == theBoard['9'] != ' ': # down the right side

printBoard(theBoard)

print("\nGame Over.\n")

print(" \*\*\*\* " +turn + " won. \*\*\*\*")

break

elif theBoard['7'] == theBoard['5'] == theBoard['3'] != ' ': # diagonal

printBoard(theBoard)

print("\nGame Over.\n")

print(" \*\*\*\* " +turn + " won. \*\*\*\*")

break

elif theBoard['1'] == theBoard['5'] == theBoard['9'] != ' ': # diagonal

printBoard(theBoard)

print("\nGame Over.\n")

print(" \*\*\*\* " +turn + " won. \*\*\*\*")

break

# If neither X nor O wins and the board is full, we'll declare the result as 'tie'.

if count == 9:

print("\nGame Over.\n")

print("It's a Tie!!")

# Now we have to change the player after every move.

if turn =='X':

turn = 'O'

else:

turn = 'X'

# Now we will ask if player wants to restart the game or not.

restart = input("Do want to play Again?(y/n)")

if restart == "y" or restart == "Y":

for key in board\_keys:

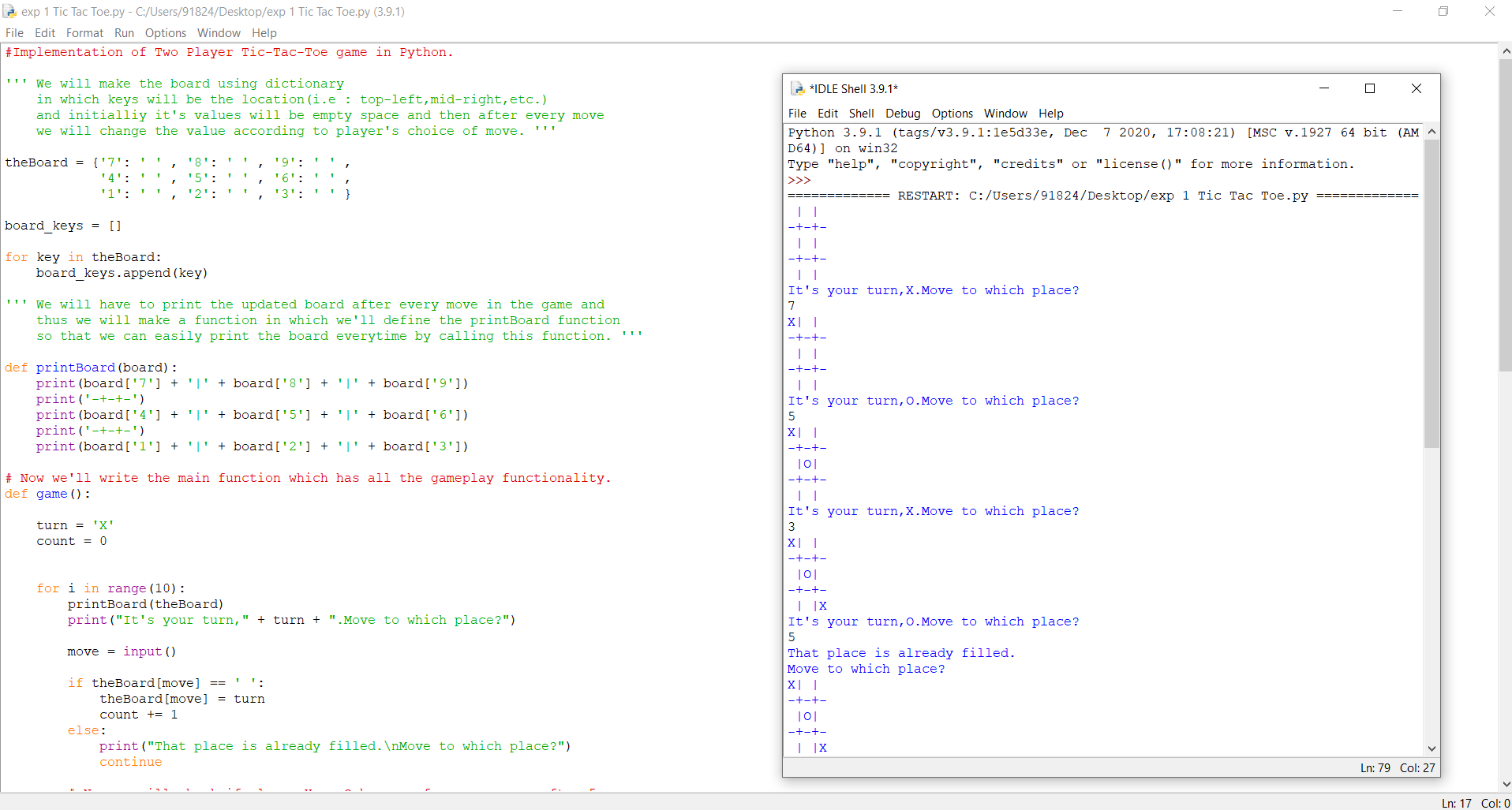
theBoard[key] = " "

game()

if \_\_name\_\_ == "\_\_main\_\_":

game()

**Output:**



**Exp no 2: Developing agent programs for real world problems (8-puzzle)**

**Link:**

<https://gist.github.com/flatline/838202>

**Source Code:**

# Solves a randomized 8-puzzle using A\* algorithm with plug-in heuristics

import random

import math

\_goal\_state = [[1,2,3],

[4,5,6],

[7,8,0]]

def index(item, seq):

"""Helper function that returns -1 for non-found index value of a seq"""

if item in seq:

return seq.index(item)

else:

return -1

class EightPuzzle:

def \_\_init\_\_(self):

# heuristic value

self.\_hval = 0

# search depth of current instance

self.\_depth = 0

# parent node in search path

self.\_parent = None

self.adj\_matrix = []

for i in range(3):

self.adj\_matrix.append(\_goal\_state[i][:])

def \_\_eq\_\_(self, other):

if self.\_\_class\_\_ != other.\_\_class\_\_:

return False

else:

return self.adj\_matrix == other.adj\_matrix

def \_\_str\_\_(self):

res = ''

for row in range(3):

res += ' '.join(map(str, self.adj\_matrix[row]))

res += '\r\n'

return res

def \_clone(self):

p = EightPuzzle()

for i in range(3):

p.adj\_matrix[i] = self.adj\_matrix[i][:]

return p

def \_get\_legal\_moves(self):

"""Returns list of tuples with which the free space may

be swapped"""

# get row and column of the empty piece

row, col = self.find(0)

free = []

# find which pieces can move there

if row > 0:

free.append((row - 1, col))

if col > 0:

free.append((row, col - 1))

if row < 2:

free.append((row + 1, col))

if col < 2:

free.append((row, col + 1))

return free

def \_generate\_moves(self):

free = self.\_get\_legal\_moves()

zero = self.find(0)

def swap\_and\_clone(a, b):

p = self.\_clone()

p.swap(a,b)

p.\_depth = self.\_depth + 1

p.\_parent = self

return p

return map(lambda pair: swap\_and\_clone(zero, pair), free)

def \_generate\_solution\_path(self, path):

if self.\_parent == None:

return path

else:

path.append(self)

return self.\_parent.\_generate\_solution\_path(path)

def solve(self, h):

"""Performs A\* search for goal state.

h(puzzle) - heuristic function, returns an integer

"""

def is\_solved(puzzle):

return puzzle.adj\_matrix == \_goal\_state

openl = [self]

closedl = []

move\_count = 0

while len(openl) > 0:

x = openl.pop(0)

move\_count += 1

if (is\_solved(x)):

if len(closedl) > 0:

return x.\_generate\_solution\_path([]), move\_count

else:

return [x]

succ = x.\_generate\_moves()

idx\_open = idx\_closed = -1

for move in succ:

# have we already seen this node?

idx\_open = index(move, openl)

idx\_closed = index(move, closedl)

hval = h(move)

fval = hval + move.\_depth

if idx\_closed == -1 and idx\_open == -1:

move.\_hval = hval

openl.append(move)

elif idx\_open > -1:

copy = openl[idx\_open]

if fval < copy.\_hval + copy.\_depth:

# copy move's values over existing

copy.\_hval = hval

copy.\_parent = move.\_parent

copy.\_depth = move.\_depth

elif idx\_closed > -1:

copy = closedl[idx\_closed]

if fval < copy.\_hval + copy.\_depth:

move.\_hval = hval

closedl.remove(copy)

openl.append(move)

closedl.append(x)

openl = sorted(openl, key=lambda p: p.\_hval + p.\_depth)

# if finished state not found, return failure

return [], 0

def shuffle(self, step\_count):

for i in range(step\_count):

row, col = self.find(0)

free = self.\_get\_legal\_moves()

target = random.choice(free)

self.swap((row, col), target)

row, col = target

def find(self, value):

"""returns the row, col coordinates of the specified value

in the graph"""

if value < 0 or value > 8:

raise Exception("value out of range")

for row in range(3):

for col in range(3):

if self.adj\_matrix[row][col] == value:

return row, col

def peek(self, row, col):

"""returns the value at the specified row and column"""

return self.adj\_matrix[row][col]

def poke(self, row, col, value):

"""sets the value at the specified row and column"""

self.adj\_matrix[row][col] = value

def swap(self, pos\_a, pos\_b):

"""swaps values at the specified coordinates"""

temp = self.peek(\*pos\_a)

self.poke(pos\_a[0], pos\_a[1], self.peek(\*pos\_b))

self.poke(pos\_b[0], pos\_b[1], temp)

def heur(puzzle, item\_total\_calc, total\_calc):

"""

Heuristic template that provides the current and target position for each number and the

total function.

Parameters:

puzzle - the puzzle

item\_total\_calc - takes 4 parameters: current row, target row, current col, target col.

Returns int.

total\_calc - takes 1 parameter, the sum of item\_total\_calc over all entries, and returns int.

This is the value of the heuristic function

"""

t = 0

for row in range(3):

for col in range(3):

val = puzzle.peek(row, col) - 1

target\_col = val % 3

target\_row = val / 3

# account for 0 as blank

if target\_row < 0:

target\_row = 2

t += item\_total\_calc(row, target\_row, col, target\_col)

return total\_calc(t)

#some heuristic functions, the best being the standard manhattan distance in this case, as it comes

#closest to maximizing the estimated distance while still being admissible.

def h\_manhattan(puzzle):

return heur(puzzle,

lambda r, tr, c, tc: abs(tr - r) + abs(tc - c),

lambda t : t)

def h\_manhattan\_lsq(puzzle):

return heur(puzzle,

lambda r, tr, c, tc: (abs(tr - r) + abs(tc - c))\*\*2,

lambda t: math.sqrt(t))

def h\_linear(puzzle):

return heur(puzzle,

lambda r, tr, c, tc: math.sqrt(math.sqrt((tr - r)\*\*2 + (tc - c)\*\*2)),

lambda t: t)

def h\_linear\_lsq(puzzle):

return heur(puzzle,

lambda r, tr, c, tc: (tr - r)\*\*2 + (tc - c)\*\*2,

lambda t: math.sqrt(t))

def h\_default(puzzle):

return 0

def main():

p = EightPuzzle()

p.shuffle(20)

print p

path, count = p.solve(h\_manhattan)

path.reverse()

for i in path:

print i

print "Solved with Manhattan distance exploring", count, "states"

path, count = p.solve(h\_manhattan\_lsq)

print "Solved with Manhattan least squares exploring", count, "states"

path, count = p.solve(h\_linear)

print "Solved with linear distance exploring", count, "states"

path, count = p.solve(h\_linear\_lsq)

print "Solved with linear least squares exploring", count, "states"

# path, count = p.solve(heur\_default)

# print "Solved with BFS-equivalent in", count, "moves"

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Output:**



# Exp no 3: Implementation of constraint satisfaction problems(Cryptarithmetic Problem)

# Link:

# <https://www.tutorialandexample.com/cryptarithmetic-problem/>

# Source Code:

// CPP program for solving cryptographic puzzles

#include <bits/stdc++.h>

using namespace std;

// vector stores 1 corresponding to index

// number which is already assigned

// to any char, otherwise stores 0

vector<int> use(10);

// structure to store char and its corresponding integer

struct node

{

char c;

int v;

};

// function check for correct solution

int check(node\* nodeArr, const int count, string s1,

string s2, string s3)

{

int val1 = 0, val2 = 0, val3 = 0, m = 1, j, i;

// calculate number corresponding to first string

for (i = s1.length() - 1; i >= 0; i--)

{

char ch = s1[i];

for (j = 0; j < count; j++)

if (nodeArr[j].c == ch)

break;

val1 += m \* nodeArr[j].v;

m \*= 10;

}

m = 1;

// calculate number corresponding to second string

for (i = s2.length() - 1; i >= 0; i--)

{

char ch = s2[i];

for (j = 0; j < count; j++)

if (nodeArr[j].c == ch)

break;

val2 += m \* nodeArr[j].v;

m \*= 10;

}

m = 1;

// calculate number corresponding to third string

for (i = s3.length() - 1; i >= 0; i--)

{

char ch = s3[i];

for (j = 0; j < count; j++)

if (nodeArr[j].c == ch)

break;

val3 += m \* nodeArr[j].v;

m \*= 10;

}

// sum of first two number equal to third return true

if (val3 == (val1 + val2))

return 1;

// else return false

return 0;

}

// Recursive function to check solution for all permutations

bool permutation(const int count, node\* nodeArr, int n,

string s1, string s2, string s3)

{

// Base case

if (n == count - 1)

{

// check for all numbers not used yet

for (int i = 0; i < 10; i++)

{

// if not used

if (use[i] == 0)

{

// assign char at index n integer i

nodeArr[n].v = i;

// if solution found

if (check(nodeArr, count, s1, s2, s3) == 1)

{

cout << "\nSolution found: ";

for (int j = 0; j < count; j++)

cout << " " << nodeArr[j].c << " = "

<< nodeArr[j].v;

return true;

}

}

}

return false;

}

for (int i = 0; i < 10; i++)

{

// if ith integer not used yet

if (use[i] == 0)

{

// assign char at index n integer i

nodeArr[n].v = i;

// mark it as not available for other char

use[i] = 1;

// call recursive function

if (permutation(count, nodeArr, n + 1, s1, s2, s3))

return true;

// backtrack for all other possible solutions

use[i] = 0;

}

}

return false;

}

bool solveCryptographic(string s1, string s2,

string s3)

{

// count to store number of unique char

int count = 0;

// Length of all three strings

int l1 = s1.length();

int l2 = s2.length();

int l3 = s3.length();

// vector to store frequency of each char

vector<int> freq(26);

for (int i = 0; i < l1; i++)

++freq[s1[i] - 'A'];

for (int i = 0; i < l2; i++)

++freq[s2[i] - 'A'];

for (int i = 0; i < l3; i++)

++freq[s3[i] - 'A'];

// count number of unique char

for (int i = 0; i < 26; i++)

if (freq[i] > 0)

count++;

// solution not possible for count greater than 10

if (count > 10)

{

cout << "Invalid strings";

return 0;

}

// array of nodes

node nodeArr[count];

// store all unique char in nodeArr

for (int i = 0, j = 0; i < 26; i++)

{

if (freq[i] > 0)

{

nodeArr[j].c = char(i + 'A');

j++;

}

}

return permutation(count, nodeArr, 0, s1, s2, s3);

}

// Driver function

int main()

{

string s1 = "SEND";

string s2 = "MORE";

string s3 = "MONEY";

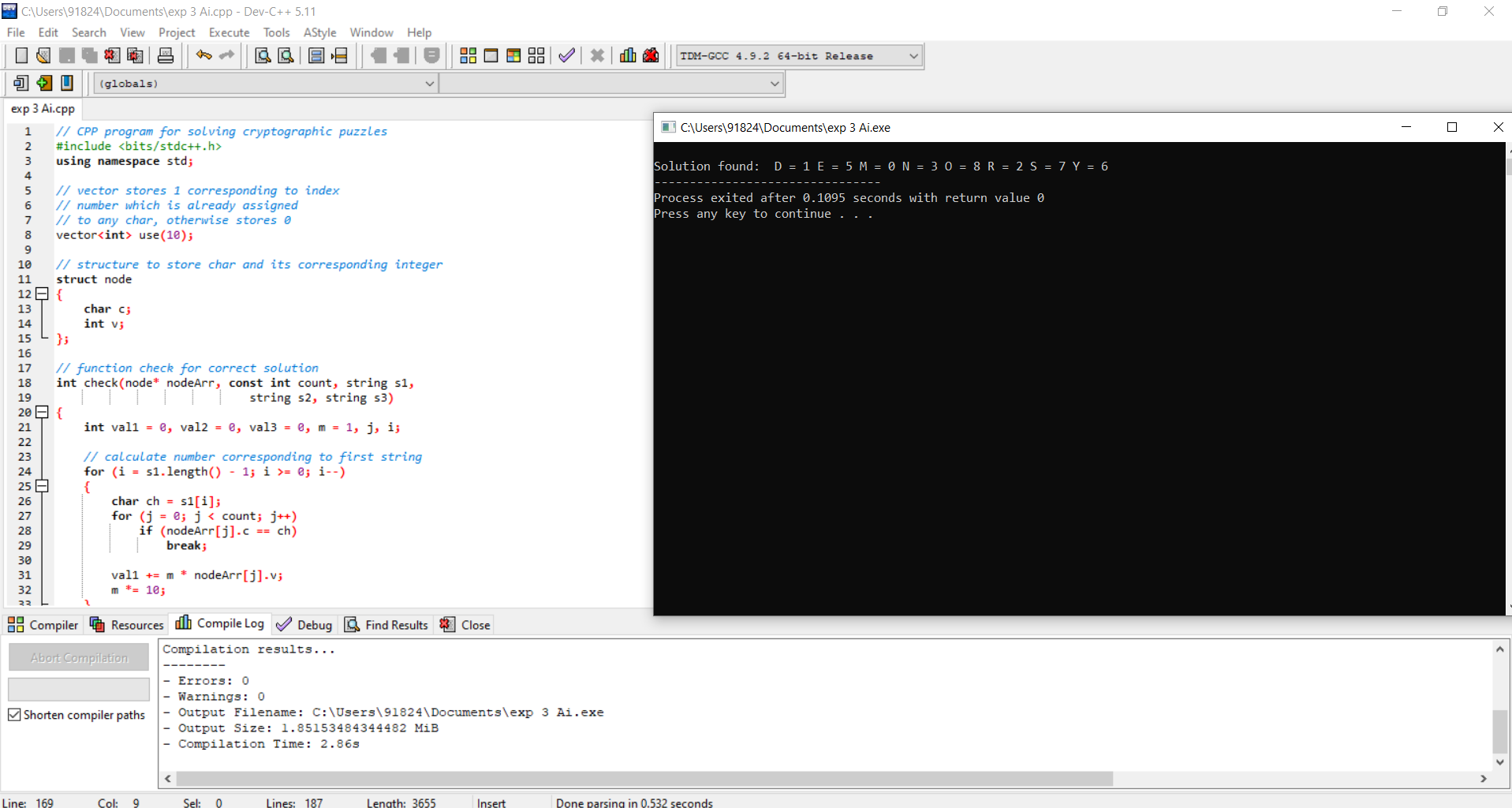
if (solveCryptographic(s1, s2, s3) == false)

cout << "No solution";

return 0;

}

**Output:**



# Exp no 4: Implementation and Analysis of DFS and BFS for an application

# Link for DFS:

# <https://www.programiz.com/dsa/graph-dfs>

**Source Code for DFS:**

# DFS algorithm in Python

# DFS algorithm

def dfs(graph, start, visited=None):

if visited is None:

visited = set()

visited.add(start)

print(start)

for next in graph[start] - visited:

dfs(graph, next, visited)

return visited

graph = {'0': set(['1', '2']),

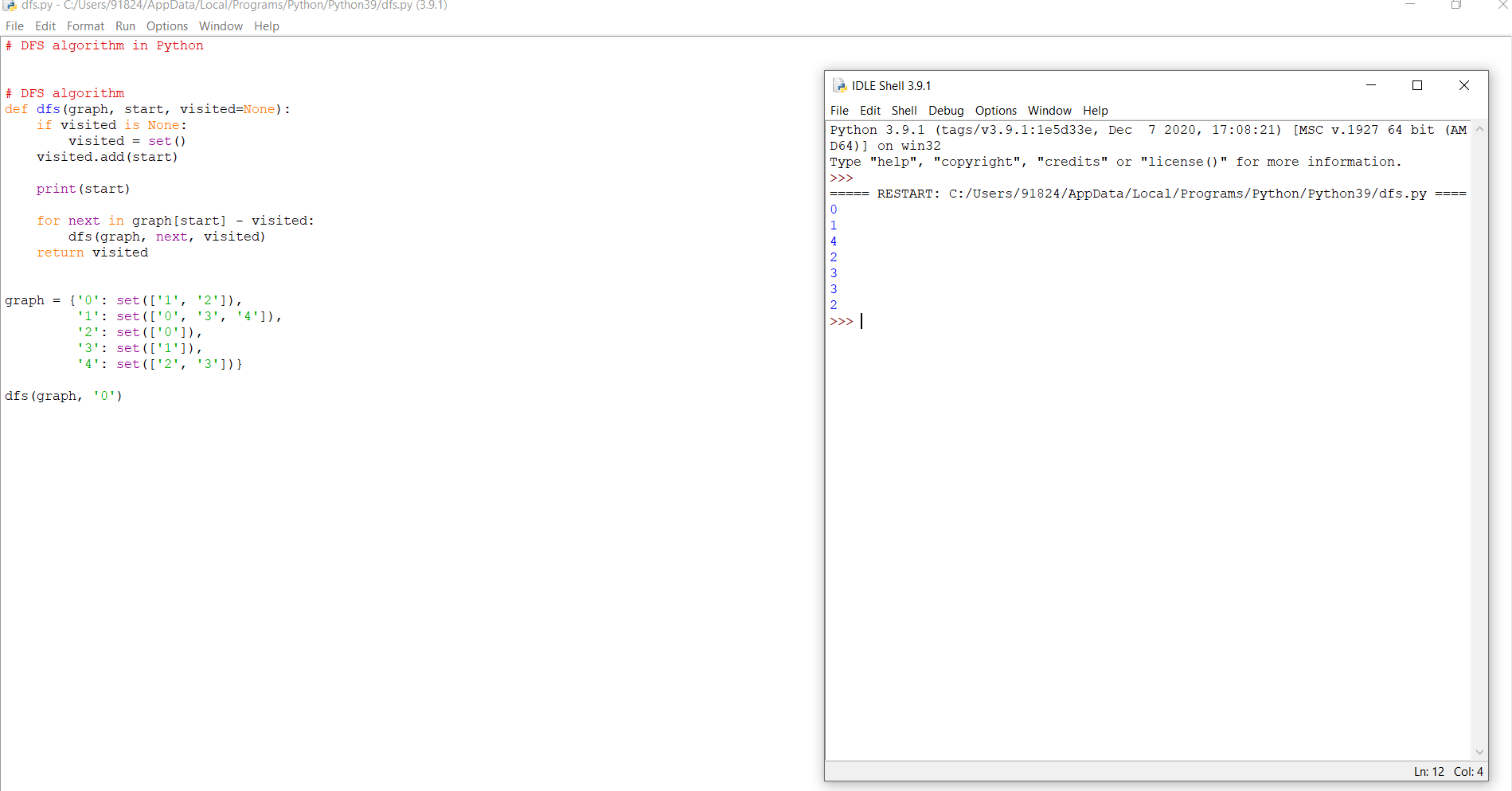
'1': set(['0', '3', '4']),

'2': set(['0']),

'3': set(['1']),

'4': set(['2', '3'])}

dfs(graph, '0')

**Output for DFS:**

# Link for BFS:

# <https://www.programiz.com/dsa/graph-bfs>

**Source Code for BFS:**

# BFS algorithm in Python

import collections

# BFS algorithm

def bfs(graph, root):

visited, queue = set(), collections.deque([root])

visited.add(root)

while queue:

# Dequeue a vertex from queue

vertex = queue.popleft()

print(str(vertex) + " ", end="")

# If not visited, mark it as visited, and

# enqueue it

for neighbour in graph[vertex]:

if neighbour not in visited:

visited.add(neighbour)

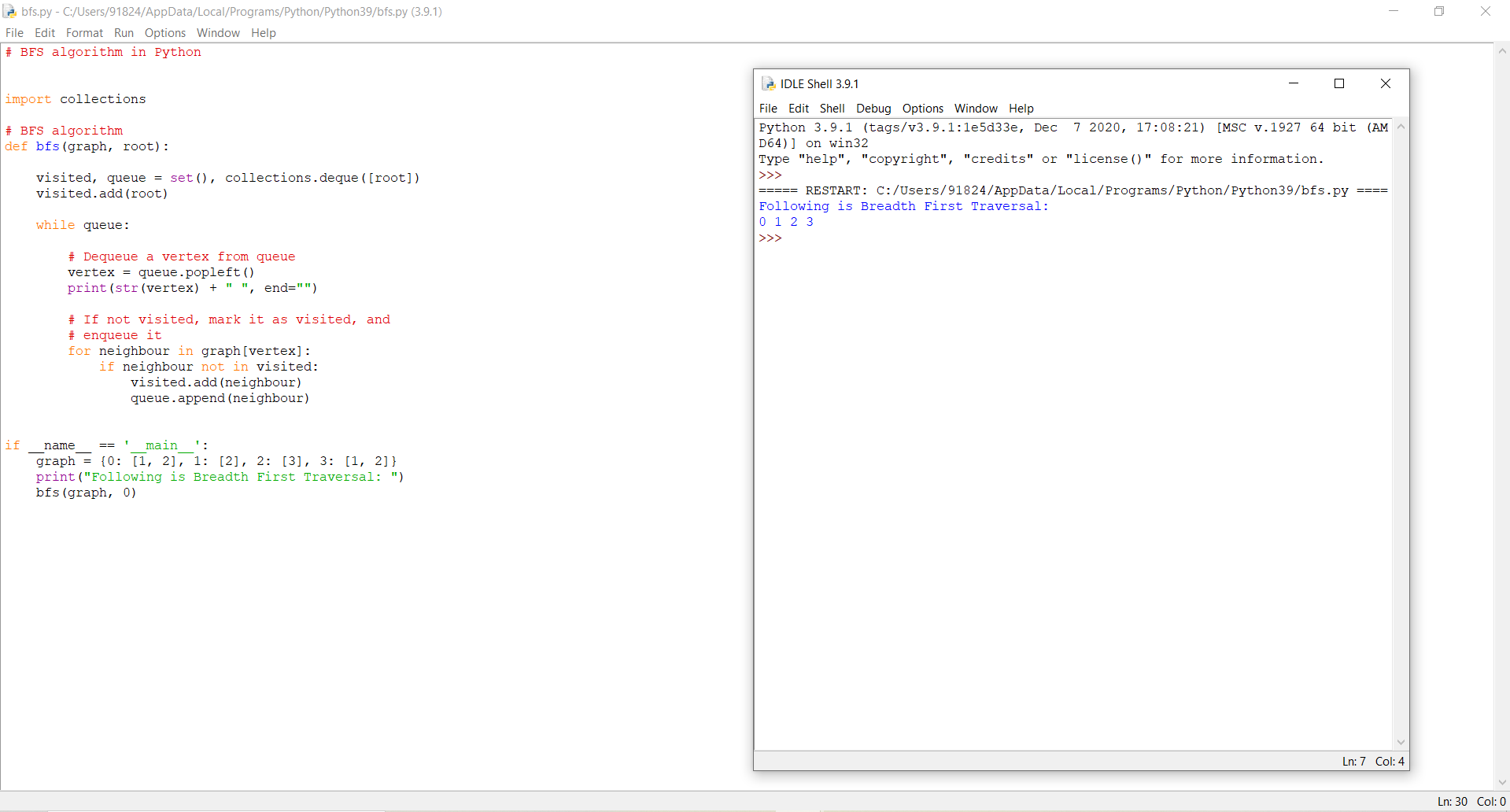
queue.append(neighbour)

if \_\_name\_\_ == '\_\_main\_\_':

graph = {0: [1, 2], 1: [2], 2: [3], 3: [1, 2]}

print("Following is Breadth First Traversal: ")

bfs(graph, 0)

**Output for BFS:**

# Exp no 5: Developing Best first search and A\* Algorithm for real world problem

# Link:

# <https://medium.com/@nicholas.w.swift/easy-a-star-pathfinding-7e6689c7f7b2>

# Source Code:

# class Node():

# """A node class for A\* Pathfinding"""

# def \_\_init\_\_(self, parent=None, position=None):

# self.parent = parent

# self.position = position

# self.g = 0

# self.h = 0

# self.f = 0

# def \_\_eq\_\_(self, other):

# return self.position == other.position

# def astar(maze, start, end):

# """Returns a list of tuples as a path from the given start to the given end in the given maze"""

# # Create start and end node

# start\_node = Node(None, start)

# start\_node.g = start\_node.h = start\_node.f = 0

# end\_node = Node(None, end)

# end\_node.g = end\_node.h = end\_node.f = 0

# # Initialize both open and closed list

# open\_list = []

# closed\_list = []

# # Add the start node

# open\_list.append(start\_node)

# # Loop until you find the end

# while len(open\_list) > 0:

# # Get the current node

# current\_node = open\_list[0]

# current\_index = 0

# for index, item in enumerate(open\_list):

# if item.f < current\_node.f:

# current\_node = item

# current\_index = index

# # Pop current off open list, add to closed list

# open\_list.pop(current\_index)

# closed\_list.append(current\_node)

# # Found the goal

# if current\_node == end\_node:

# path = []

# current = current\_node

# while current is not None:

# path.append(current.position)

# current = current.parent

# return path[::-1] # Return reversed path

# # Generate children

# children = []

# for new\_position in [(0, -1), (0, 1), (-1, 0), (1, 0), (-1, -1), (-1, 1), (1, -1), (1, 1)]: # Adjacent squares

# # Get node position

# node\_position = (current\_node.position[0] + new\_position[0], current\_node.position[1] + new\_position[1])

# # Make sure within range

# if node\_position[0] > (len(maze) - 1) or node\_position[0] < 0 or node\_position[1] > (len(maze[len(maze)-1]) -1) or node\_position[1] < 0:

# continue

# # Make sure walkable terrain

# if maze[node\_position[0]][node\_position[1]] != 0:

# continue

# # Create new node

# new\_node = Node(current\_node, node\_position)

# # Append

# children.append(new\_node)

# # Loop through children

# for child in children:

# # Child is on the closed list

# for closed\_child in closed\_list:

# if child == closed\_child:

# continue

# # Create the f, g, and h values

# child.g = current\_node.g + 1

# child.h = ((child.position[0] - end\_node.position[0]) \*\* 2) + ((child.position[1] - end\_node.position[1]) \*\* 2)

# child.f = child.g + child.h

# # Child is already in the open list

# for open\_node in open\_list:

# if child == open\_node and child.g > open\_node.g:

# continue

# # Add the child to the open list

# open\_list.append(child)

# def main():

# maze = [[0, 0, 0, 0, 1, 0, 0, 0, 0, 0],

# [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],

# [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],

# [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],

# [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],

# [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],

# [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],

# [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],

# [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],

# [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]

# start = (0, 0)

# end = (7, 6)

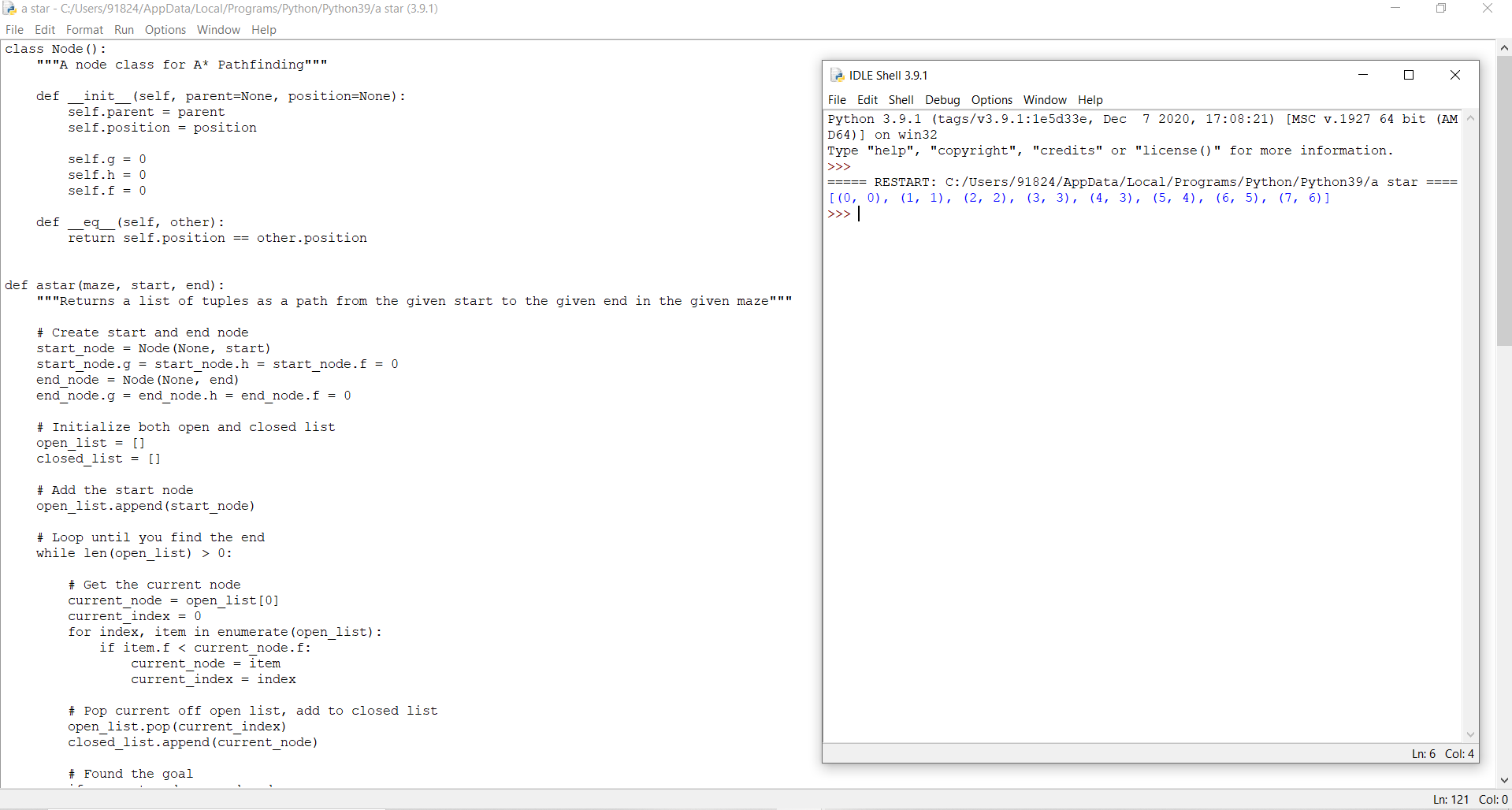
# path = astar(maze, start, end)

# print(path)

# if \_\_name\_\_ == '\_\_main\_\_':

# main()

# Output:



**Exp no 6:Implementation of minimax algorithm for an application**

**Link:**

<https://www.annytab.com/minimax-algorithm-in-python/>

**Source Code:**

# Import libraries

import sys

import random

# This class represent a tic tac to game

class TicTacToeGame:

# Create a new game

def \_\_init\_\_(self, rows:int, columns:int, goal:int, max\_depth:int=4):

# Create the game state

self.state = []

self.tiles = {}

self.inverted\_tiles = {}

tile = 0

for y in range(rows):

row = []

for x in range(columns):

row += '.'

tile += 1

self.tiles[tile] = (y, x)

self.inverted\_tiles[(y, x)] = tile

self.state.append(row)

# Set the number of noughts and crosses in a row that is needed to win the game

self.goal = goal

# Create vectors

self.vectors = [(1,0), (0,1), (1,1), (-1,1)]

# Set lengths

self.rows = rows

self.columns = columns

self.max\_row\_index = rows - 1

self.max\_columns\_index = columns - 1

self.max\_depth = max\_depth

# Heuristics for cutoff

self.winning\_positions = []

self.get\_winning\_positions()

# Set the starting player at random

#self.player = 'O'

self.player = random.choice(['X', 'O'])

# Get winning positions

def get\_winning\_positions(self):

# Loop the board

for y in range(self.rows):

for x in range(self.columns):

# Loop vectors

for vector in self.vectors:

# Get the start position

sy, sx = (y, x)

# Get vector deltas

dy, dx = vector

# Create a counter

counter = 0

# Loop until we are outside the board

positions = []

while True:

# Add the position

positions.append(self.inverted\_tiles.get((sy, sx)))

# Check if we have a winning position

if (len(positions) == self.goal):

# Add winning positions

self.winning\_positions.append(positions)

# Break out from the loop

break

# Update the position

sy += dy

sx += dx

# Check if the loop should terminate

if(sy < 0 or abs(sy) > self.max\_row\_index or sx < 0 or abs(sx) > self.max\_columns\_index):

break

# Play the game

def play(self):

# Variables

result = None

# Create an infinite loop

print('Starting board')

while True:

# Draw the state

self.print\_state()

# Get a move from a player

if (self.player == 'X'): # AI

# Print AI move

print('Player X moving (AI) ...')

# Get the best move

max, py, px, depth = self.max(-sys.maxsize, sys.maxsize)

# Get a heuristic move at cutoff

print('Depth: {0}'.format(depth))

if(depth > self.max\_depth):

py, px = self.get\_best\_move()

# Make a move

self.state[py][px] = 'X'

# Check if the game has ended, break out from the loop in that case

result = self.game\_ended()

if(result != None):

break

# Change turn

self.player = 'O'

elif (self.player == 'O'): # Human player

# Print turn

print('Player O moving (Human) ...')

# Get a recommended move

min, py, px, depth = self.min(-sys.maxsize, sys.maxsize)

# Get a heuristic move at cutoff

print('Depth: {0}'.format(depth))

if(depth > self.max\_depth):

py, px = self.get\_best\_move()

# Print a recommendation

print('Recommendation: {0}'.format(self.inverted\_tiles.get((py, px))))

# Get input

number = int(input('Make a move (tile number): '))

tile = self.tiles.get(number)

# Check if the move is legal

if(tile != None):

# Make a move

py, px = tile

self.state[py][px] = 'O'

# Check if the game has ended, break out from the loop in that case

result = self.game\_ended()

if(result != None):

break

# Change turn

self.player = 'X'

else:

print('Move is not legal, try again.')

# Print result

self.print\_state()

print('Winner is player: {0}'.format(result))

# An evaluation function to get the best move based on heuristics

def get\_best\_move(self):

# Create an heuristic dictionary

heuristics = {}

# Get all empty cells

empty\_cells = []

for y in range(self.rows):

for x in range(self.columns):

if (self.state[y][x] == '.'):

empty\_cells.append((y, x))

# Loop empty positions

for empty in empty\_cells:

# Get numbered position

number = self.inverted\_tiles.get(empty)

# Loop winning positions

for win in self.winning\_positions:

# Check if number is in a winning position

if(number in win):

# Calculate the number of X:s and O:s in the winning position

player\_x = 0

player\_o = 0

start\_score = 1

for box in win:

# Get the position

y, x = self.tiles[box]

# Count X:s and O:s

if(self.state[y][x] == 'X'):

player\_x += start\_score if self.player == 'X' else start\_score \* 2

start\_score \*= 10

elif (self.state[y][x] == 'O'):

player\_o += start\_score if self.player == 'O' else start\_score \* 2

start\_score \*= 10

# Save heuristic

if(player\_x == 0 or player\_o == 0):

# Calculate a score

score = max(player\_x, player\_o) + start\_score

# Update the score

if(heuristics.get(number) != None):

heuristics[number] += score

else:

heuristics[number] = score

# Get the best move from the heuristic dictionary

best\_move = random.choice(empty\_cells)

best\_count = -sys.maxsize

for key, value in heuristics.items():

if(value > best\_count):

best\_move = self.tiles.get(key)

best\_count = value

# Return the best move

return best\_move

# Check if the game has ended

def game\_ended(self) -> str:

# Check if a player has won

result = self.player\_has\_won()

if(result != None):

return result

# Check if the board is full

for y in range(self.rows):

for x in range(self.columns):

if (self.state[y][x] == '.'):

return None

# Return a tie

return 'It is a tie!'

# Check if a player has won

def player\_has\_won(self) -> str:

# Loop the board

for y in range(self.rows):

for x in range(self.columns):

# Loop vectors

for vector in self.vectors:

# Get the start position

sy, sx = (y, x)

# Get vector deltas

dy, dx = vector

# Create counters

steps = 0

player\_x = 0

player\_o = 0

# Loop until we are outside the board or have moved the number of steps in the goal

while steps < self.goal:

# Add steps

steps += 1

# Check if a player has a piece in the tile

if(self.state[sy][sx] == 'X'):

player\_x += 1

elif(self.state[sy][sx] == 'O'):

player\_o += 1

# Update the position

sy += dy

sx += dx

# Check if the loop should terminate

if(sy < 0 or abs(sy) > self.max\_row\_index or sx < 0 or abs(sx) > self.max\_columns\_index):

break

# Check if we have a winner

if(player\_x >= self.goal):

return 'X'

elif(player\_o >= self.goal):

return 'O'

# Return None if no winner is found

return None

# Get a min value (O)

def min(self, alpha:int=-sys.maxsize, beta:int=sys.maxsize, depth:int=0):

# Variables

min\_value = sys.maxsize

by = None

bx = None

# Check if the game has ended

result = self.game\_ended()

if(result != None):

if result == 'X':

return 1, 0, 0, depth

elif result == 'O':

return -1, 0, 0, depth

elif result == 'It is a tie!':

return 0, 0, 0, depth

elif(depth > self.max\_depth):

return 0, 0, 0, depth

# Loop the board

for y in range(self.rows):

for x in range(self.columns):

# Check if the tile is empty

if (self.state[y][x] == '.'):

# Make a move

self.state[y][x] = 'O'

# Get max value

max, max\_y, max\_x, depth = self.max(alpha, beta, depth + 1)

# Set min value to max value if it is lower than curren min value

if (max < min\_value):

min\_value = max

by = y

bx = x

# Reset the tile

self.state[y][x] = '.'

# Do an alpha test

if (min\_value <= alpha):

return min\_value, bx, by, depth

# Do a beta test

if (min\_value < beta):

beta = min\_value

# Return min value

return min\_value, by, bx, depth

# Get max value (X)

def max(self, alpha:int=-sys.maxsize, beta:int=sys.maxsize, depth:int=0):

# Variables

max\_value = -sys.maxsize

by = None

bx = None

# Check if the game has ended

result = self.game\_ended()

if(result != None):

if result == 'X':

return 1, 0, 0, depth

elif result == 'O':

return -1, 0, 0, depth

elif result == 'It is a tie!':

return 0, 0, 0, depth

elif(depth > self.max\_depth):

return 0, 0, 0, depth

# Loop the board

for y in range(self.rows):

for x in range(self.columns):

# Check if the current tile is empty

if (self.state[y][x] == '.'):

# Add a piece to the board

self.state[y][x] = 'X'

# Set max value to min value if min value is greater than current max value

min, min\_y, min\_x, depth = self.min(alpha, beta, depth + 1)

# Adjust the max value

if (min > max\_value):

max\_value = min

by = y

bx = x

# Reset the tile

self.state[y][x] = '.'

# Do a beta test

if (max\_value >= beta):

return max\_value, bx, by, depth

# Do an alpha test

if (max\_value > alpha):

alpha = max\_value

# Return max value

return max\_value, by, bx, depth

# Print the current game state

def print\_state(self):

for y in range(self.rows):

print('| ', end='')

for x in range(self.columns):

if (self.state[y][x] != '.'):

print(' {0} | '.format(self.state[y][x]), end='')

else:

digit = str(self.inverted\_tiles.get((y,x))) if len(str(self.inverted\_tiles.get((y,x)))) > 1 else ' ' + str(self.inverted\_tiles.get((y,x)))

print('{0} | '.format(digit), end='')

print()

print()

# The main entry point for this module

def main():

# Create a game

#game = TicTacToeGame(7, 6, 4, 1000)

game = TicTacToeGame(3, 3, 3, 1000)

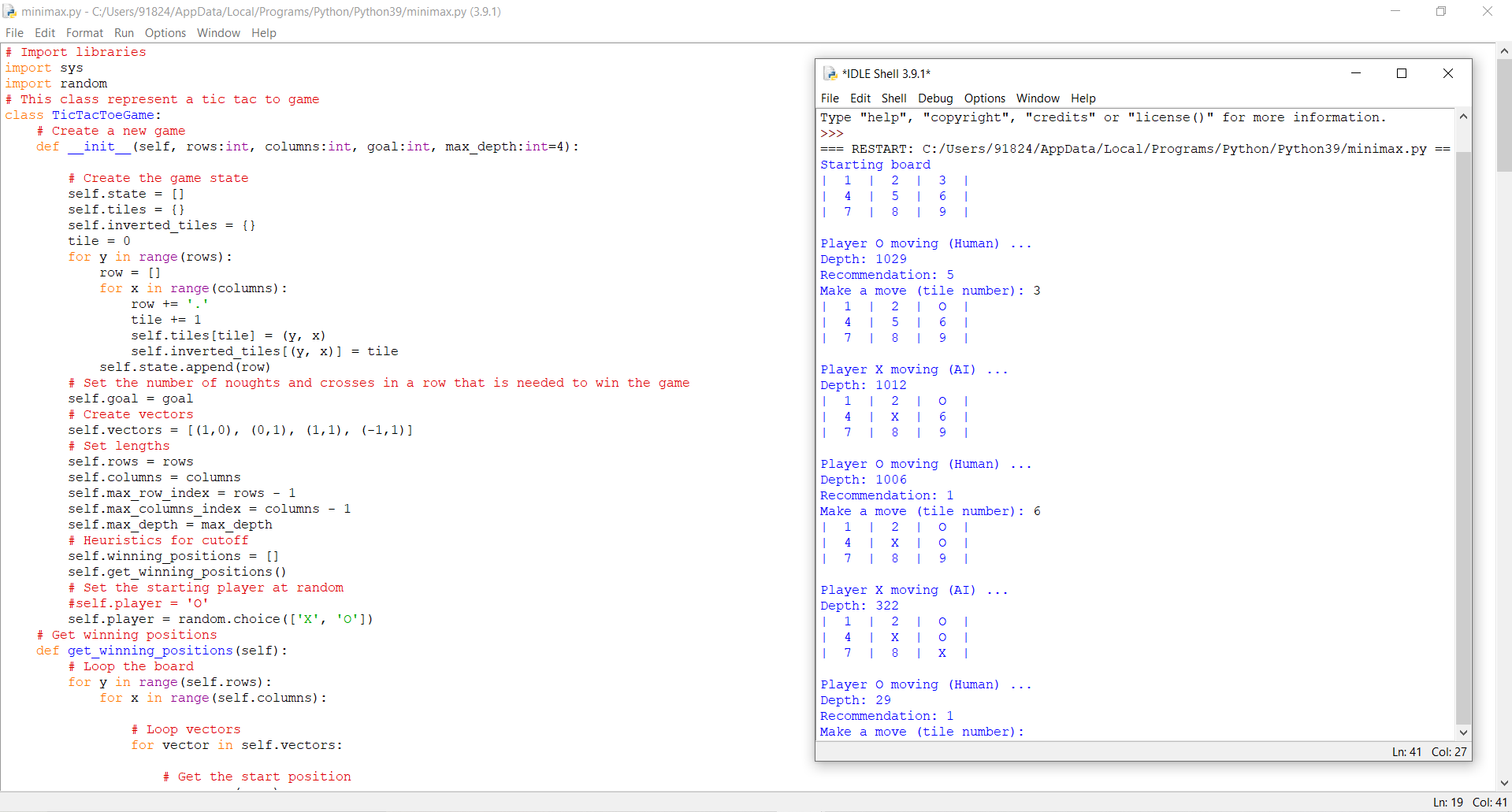
# Play the game

game.play()

# Tell python to run main method

if \_\_name\_\_ == "\_\_main\_\_": main()

**Output:**



**Exp no 7:Implementation of Unification and resolution.**

**Link:**

**Source Code:**

def get\_index\_comma(string):

"""

Return index of commas in string

"""

index\_list = list()

# Count open parentheses

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):

"""

Check if expression is variable

"""

for i in expr:

if i == '(':

return False

return True

def process\_expression(expr):

"""

input: - expression:

'Q(a, g(x, b), f(y))'

return: - predicate symbol:

Q

- list of arguments

['a', 'g(x, b)', 'f(y)']

"""

# Remove space in expression

expr = expr.replace(' ', '')

# Find the first index == '('

index = None

for i in range(len(expr)):

if expr[i] == '(':

index = i

break

# Return predicate symbol and remove predicate symbol in expression

predicate\_symbol = expr[:index]

expr = expr.replace(predicate\_symbol, '')

# Remove '(' in the first index and ')' in the last index

expr = expr[1:len(expr) - 1]

# List of arguments

arg\_list = list()

# Split string with commas, return list of arguments

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):

"""

input: expression:

'Q(a, g(x, b), f(y))'

return: full list of arguments:

['a', 'x', 'b', 'y']

"""

\_, 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):

"""

Check if var occurs in expr

"""

arg\_list = get\_arg\_list(expr)

if var in arg\_list:

return True

return False

def unify(expr1, expr2):

"""

Unification Algorithm

Step 1: If Ψ1 or Ψ2 is a variable or constant, then:

a, If Ψ1 or Ψ2 are identical, then return NULL.

b, Else if Ψ1 is a variable:

- then if Ψ1 occurs in Ψ2, then return False

- Else return (Ψ2 / Ψ1)

c, Else if Ψ2 is a variable:

- then if Ψ2 occurs in Ψ1, then return False

- Else return (Ψ1 / Ψ2)

d, Else return False

Step 2: If the initial Predicate symbol in Ψ1 and Ψ2 are not same, then return False.

Step 3: IF Ψ1 and Ψ2 have a different number of arguments, then return False.

Step 4: Create Substitution list.

Step 5: For i=1 to the number of elements in Ψ1.

a, Call Unify function with the ith element of Ψ1 and ith element of Ψ2, and put the result into S.

b, If S = False then returns False

c, If S ≠ Null then append to Substitution list

Step 6: Return Substitution list.

"""

# Step 1:

if is\_variable(expr1) and is\_variable(expr2):

if expr1 == expr2:

return 'Null'

else:

return False

elif is\_variable(expr1) and not is\_variable(expr2):

if check\_occurs(expr1, expr2):

return False

else:

tmp = str(expr2) + '/' + str(expr1)

return tmp

elif not is\_variable(expr1) and is\_variable(expr2):

if check\_occurs(expr2, expr1):

return False

else:

tmp = str(expr1) + '/' + str(expr2)

return tmp

else:

predicate\_symbol\_1, arg\_list\_1 = process\_expression(expr1)

predicate\_symbol\_2, arg\_list\_2 = process\_expression(expr2)

# Step 2

if predicate\_symbol\_1 != predicate\_symbol\_2:

return False

# Step 3

elif len(arg\_list\_1) != len(arg\_list\_2):

return False

else:

# Step 4: Create substitution list

sub\_list = list()

# Step 5:

for i in range(len(arg\_list\_1)):

tmp = unify(arg\_list\_1[i], arg\_list\_2[i])

if not tmp:

return False

elif tmp == 'Null':

pass

else:

if type(tmp) == list:

for j in tmp:

sub\_list.append(j)

else:

sub\_list.append(tmp)

# Step 6

return sub\_list

if \_\_name\_\_ == '\_\_main\_\_':

# Data 1

f1 = 'p(b(A), X, f(g(Z)))'

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

# Data 2

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

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

# Data 3

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

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

result = unify(f1, f2)

if not result:

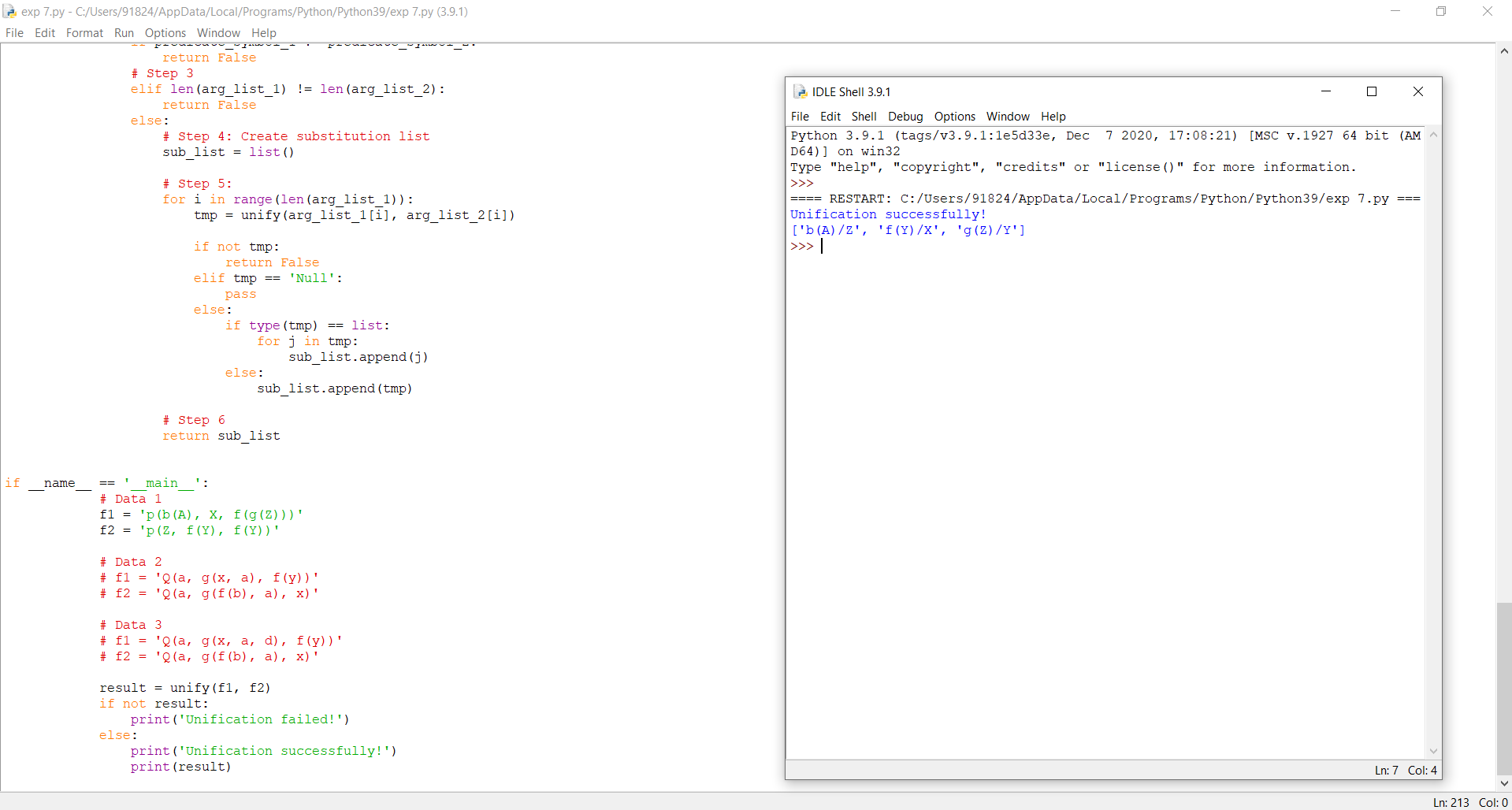
print('Unification failed!')

else:

print('Unification successfully!')

print(result)

**Output:**



**Exp no 8: Implementation of Knowledge Represntation schemes**

**Link:**

<https://towardsdatascience.com/knowledge-representation-and-reasoning-with-answer-set-programming-376e3113a421>

**Source Code:**

#include <stdio.h>

#define SIZE 9

//sudoku problem

int matrix[9][9] = {

{5,3,0,0,7,0,0,0,0},

{6,0,0,1,9,5,0,0,0},

{0,9,8,0,0,0,0,6,0},

{8,0,0,0,6,0,0,0,3},

{4,0,0,8,0,3,0,0,1},

{7,0,0,0,2,0,0,0,6},

{0,6,0,0,0,0,2,8,0},

{0,0,0,4,1,9,0,0,5},

{0,0,0,0,8,0,0,7,9}

};

//function to print sudoku

void print\_sudoku()

{

int i,j;

for(i=0;i<SIZE;i++)

{

for(j=0;j<SIZE;j++)

{

printf("%d\t",matrix[i][j]);

}

printf("\n\n");

}

}

//function to check if all cells are assigned or not

//if there is any unassigned cell

//then this function will change the values of

//row and col accordingly

int number\_unassigned(int \*row, int \*col)

{

int num\_unassign = 0;

int i,j;

for(i=0;i<SIZE;i++)

{

for(j=0;j<SIZE;j++)

{

//cell is unassigned

if(matrix[i][j] == 0)

{

//changing the values of row and col

\*row = i;

\*col = j;

//there is one or more unassigned cells

num\_unassign = 1;

return num\_unassign;

}

}

}

return num\_unassign;

}

//function to check if we can put a

//value in a paticular cell or not

int is\_safe(int n, int r, int c)

{

int i,j;

//checking in row

for(i=0;i<SIZE;i++)

{

//there is a cell with same value

if(matrix[r][i] == n)

return 0;

}

//checking column

for(i=0;i<SIZE;i++)

{

//there is a cell with the value equal to i

if(matrix[i][c] == n)

return 0;

}

//checking sub matrix

int row\_start = (r/3)\*3;

int col\_start = (c/3)\*3;

for(i=row\_start;i<row\_start+3;i++)

{

for(j=col\_start;j<col\_start+3;j++)

{

if(matrix[i][j]==n)

return 0;

}

}

return 1;

}

//function to solve sudoku

//using backtracking

int solve\_sudoku()

{

int row;

int col;

//if all cells are assigned then the sudoku is already solved

//pass by reference because number\_unassigned will change the values of row and col

if(number\_unassigned(&row, &col) == 0)

return 1;

int n,i;

//number between 1 to 9

for(i=1;i<=SIZE;i++)

{

//if we can assign i to the cell or not

//the cell is matrix[row][col]

if(is\_safe(i, row, col))

{

matrix[row][col] = i;

//backtracking

if(solve\_sudoku())

return 1;

//if we can't proceed with this solution

//reassign the cell

matrix[row][col]=0;

}

}

return 0;

}

int main()

{

if (solve\_sudoku())

print\_sudoku();

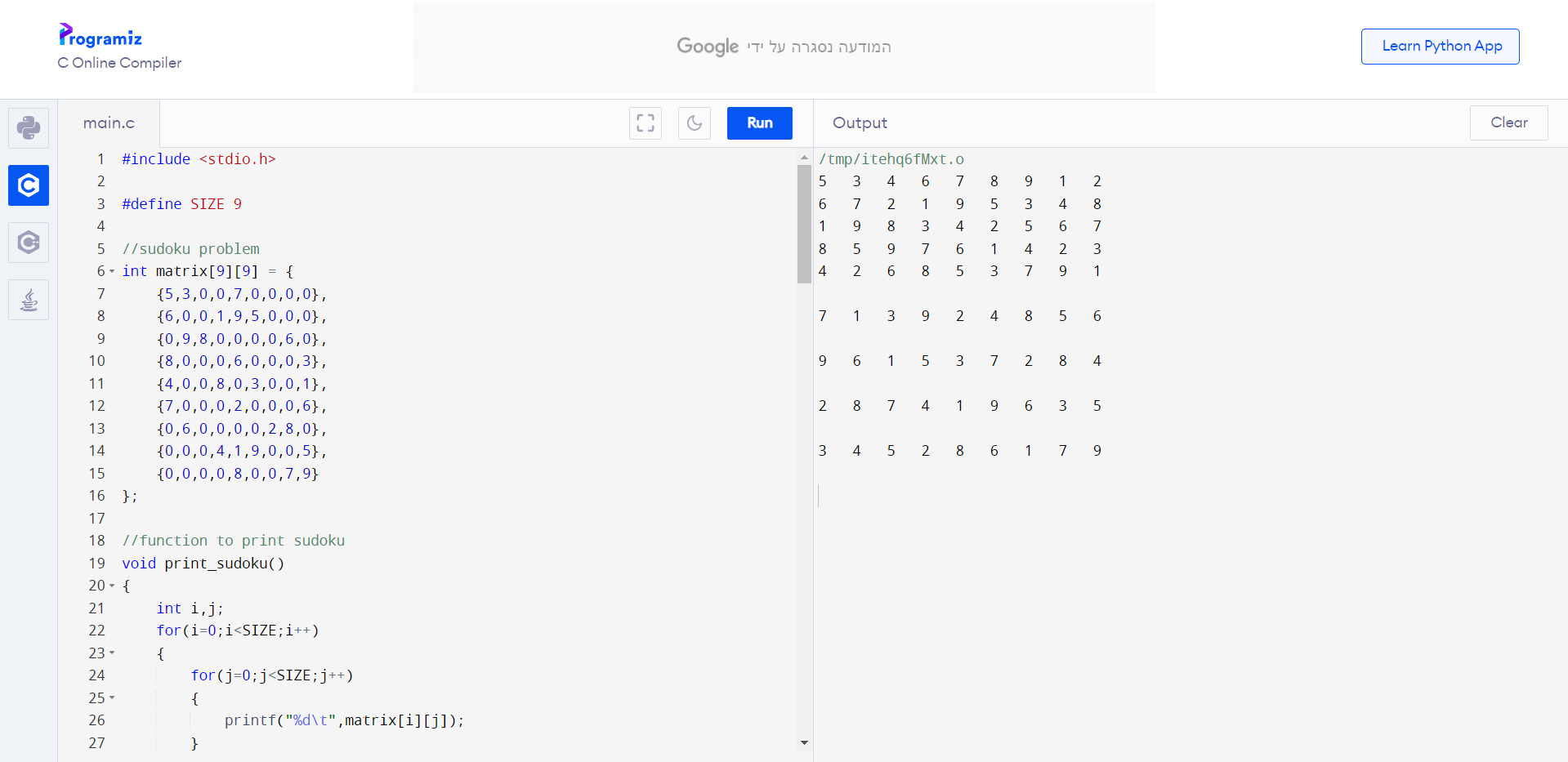
else

printf("No solution\n");

return 0;

}

**Output:**



**Exp no 9:Implementation of Uncertain Methods for an application.**

**Link:**

<https://towardsdatascience.com/my-deep-learning-model-says-sorry-i-dont-know-the-answer-that-s-absolutely-ok-50ffa562cb0b>

**Source Code:**

# prediction

y\_pred\_without\_dropout = model\_without\_dropout.predict(x\_test)

y\_pred\_with\_dropout = model\_with\_dropout.predict(x\_test)

# plotting

fig, ax = plt.subplots(1,1,figsize=(10,5))

ax.scatter(x\_train, y\_train, s=10, label='train data')

ax.plot(x\_test, x\_test, ls='--', label='test data', color='green')

ax.plot(x\_test, y\_pred\_without\_dropout, label='predicted ANN - R2 {:.2f}'.format(r2\_score(x\_test, y\_pred\_without\_dropout)), color='red')

ax.plot(x\_test, y\_pred\_with\_dropout, label='predicted ANN Dropout - R2 {:.2f}'.format(r2\_score(x\_test, y\_pred\_with\_dropout)), color='black')

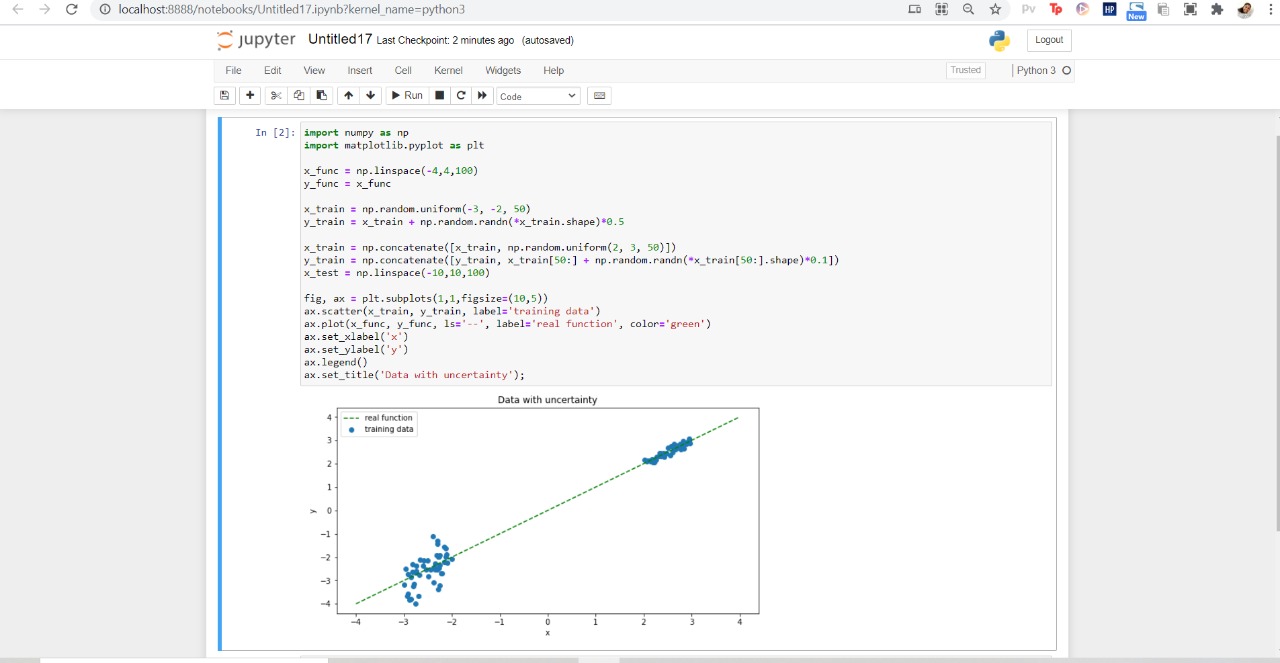
ax.set\_xlabel('x')

ax.set\_ylabel('y')

ax.legend()

ax.set\_title('test data');

**Output:**

****

**Exp no 10 :Implementation of block world problem**

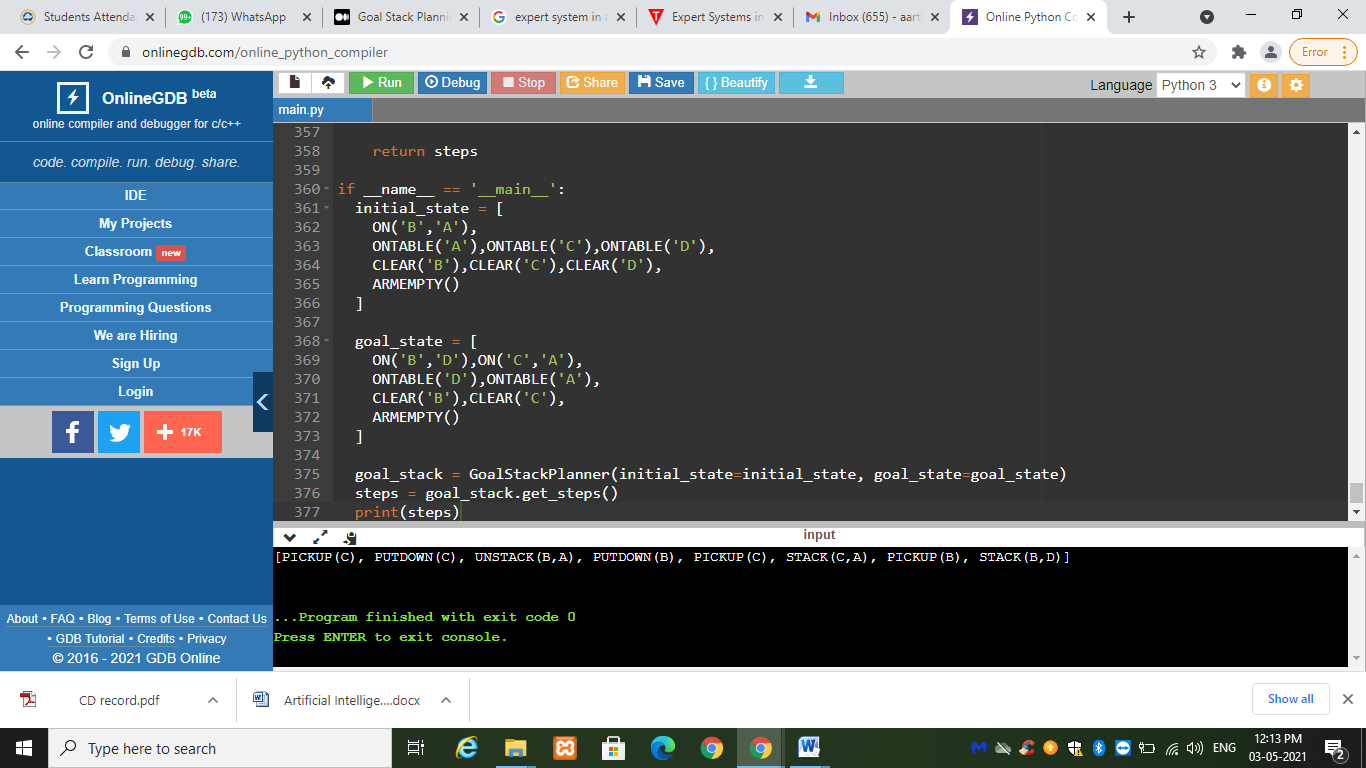
**Link:**

<https://apoorvdixit619.medium.com/goal-stack-planning-for-blocks-world-problem-41779d090f29>

**Source code:**

|  |
| --- |
| #Base Classes |
|  |  |
|  | #PREDICATE - ON, ONTABLE, CLEAR, HOLDING, ARMEMPTY |
|  | class PREDICATE: |
|  | def \_\_str\_\_(self): |
|  | pass |
|  | def \_\_repr\_\_(self): |
|  | pass |
|  | def \_\_eq\_\_(self, other) : |
|  | pass |
|  | def \_\_hash\_\_(self): |
|  | pass |
|  | def get\_action(self, world\_state): |
|  | pass |
|  |  |
|  |  |
|  | #OPERATIONS - Stack, Unstack, Pickup, Putdown |
|  | class Operation: |
|  | def \_\_str\_\_(self): |
|  | pass |
|  | def \_\_repr\_\_(self): |
|  | pass |
|  | def \_\_eq\_\_(self, other) : |
|  | pass |
|  | def precondition(self): |
|  | pass |
|  | def delete(self): |
|  | pass |
|  | def add(self): |
|  | pass |
|  |  |
|  |  |
|  | class ON(PREDICATE): |
|  |  |
|  | def \_\_init\_\_(self, X, Y): |
|  | self.X = X |
|  | self.Y = Y |
|  |  |
|  | def \_\_str\_\_(self): |
|  | return "ON({X},{Y})".format(X=self.X,Y=self.Y) |
|  |  |
|  | def \_\_repr\_\_(self): |
|  | return self.\_\_str\_\_() |
|  |  |
|  | def \_\_eq\_\_(self, other) : |
|  | return self.\_\_dict\_\_ == other.\_\_dict\_\_ and self.\_\_class\_\_ == other.\_\_class\_\_ |
|  |  |
|  | def \_\_hash\_\_(self): |
|  | return hash(str(self)) |
|  |  |
|  | def get\_action(self, world\_state): |
|  | return StackOp(self.X,self.Y) |
|  |  |
|  |  |
|  | class ONTABLE(PREDICATE): |
|  |  |
|  | def \_\_init\_\_(self, X): |
|  | self.X = X |
|  |  |
|  | def \_\_str\_\_(self): |
|  | return "ONTABLE({X})".format(X=self.X) |
|  |  |
|  | def \_\_repr\_\_(self): |
|  | return self.\_\_str\_\_() |
|  |  |
|  | def \_\_eq\_\_(self, other) : |
|  | return self.\_\_dict\_\_ == other.\_\_dict\_\_ and self.\_\_class\_\_ == other.\_\_class\_\_ |
|  |  |
|  | def \_\_hash\_\_(self): |
|  | return hash(str(self)) |
|  |  |
|  | def get\_action(self, world\_state): |
|  | return PutdownOp(self.X) |
|  |  |
|  |  |
|  | class CLEAR(PREDICATE): |
|  |  |
|  | def \_\_init\_\_(self, X): |
|  | self.X = X |
|  |  |
|  | def \_\_str\_\_(self): |
|  | return "CLEAR({X})".format(X=self.X) |
|  | self.X = X |
|  |  |
|  | def \_\_repr\_\_(self): |
|  | return self.\_\_str\_\_() |
|  |  |
|  | def \_\_eq\_\_(self, other) : |
|  | return self.\_\_dict\_\_ == other.\_\_dict\_\_ and self.\_\_class\_\_ == other.\_\_class\_\_ |
|  |  |
|  | def \_\_hash\_\_(self): |
|  | return hash(str(self)) |
|  |  |
|  | def get\_action(self, world\_state): |
|  | for predicate in world\_state: |
|  | #If Block is on another block, unstack |
|  | if isinstance(predicate,ON) and predicate.Y==self.X: |
|  | return UnstackOp(predicate.X, predicate.Y) |
|  | return None |
|  |  |
|  |  |
|  | class HOLDING(PREDICATE): |
|  |  |
|  | def \_\_init\_\_(self, X): |
|  | self.X = X |
|  |  |
|  | def \_\_str\_\_(self): |
|  | return "HOLDING({X})".format(X=self.X) |
|  |  |
|  | def \_\_repr\_\_(self): |
|  | return self.\_\_str\_\_() |
|  |  |
|  | def \_\_eq\_\_(self, other) : |
|  | return self.\_\_dict\_\_ == other.\_\_dict\_\_ and self.\_\_class\_\_ == other.\_\_class\_\_ |
|  |  |
|  | def \_\_hash\_\_(self): |
|  | return hash(str(self)) |
|  |  |
|  | def get\_action(self, world\_state): |
|  | X = self.X |
|  | #If block is on table, pick up |
|  | if ONTABLE(X) in world\_state: |
|  | return PickupOp(X) |
|  | #If block is on another block, unstack |
|  | else: |
|  | for predicate in world\_state: |
|  | if isinstance(predicate,ON) and predicate.X==X: |
|  | return UnstackOp(X,predicate.Y) |
|  |  |
|  |  |
|  | class ARMEMPTY(PREDICATE): |
|  |  |
|  | def \_\_init\_\_(self): |
|  | pass |
|  |  |
|  | def \_\_str\_\_(self): |
|  | return "ARMEMPTY" |
|  |  |
|  | def \_\_repr\_\_(self): |
|  | return self.\_\_str\_\_() |
|  |  |
|  | def \_\_eq\_\_(self, other) : |
|  | return self.\_\_dict\_\_ == other.\_\_dict\_\_ and self.\_\_class\_\_ == other.\_\_class\_\_ |
|  |  |
|  | def \_\_hash\_\_(self): |
|  | return hash(str(self)) |
|  |  |
|  | def get\_action(self, world\_state=[]): |
|  | for predicate in world\_state: |
|  | if isinstance(predicate,HOLDING): |
|  | return PutdownOp(predicate.X) |
|  | return None |
|  |  |
|  |  |
|  | class StackOp(Operation): |
|  |  |
|  | def \_\_init\_\_(self, X, Y): |
|  | self.X = X |
|  | self.Y = Y |
|  |  |
|  | def \_\_str\_\_(self): |
|  | return "STACK({X},{Y})".format(X=self.X,Y=self.Y) |
|  |  |
|  | def \_\_repr\_\_(self): |
|  | return self.\_\_str\_\_() |
|  |  |
|  | def \_\_eq\_\_(self, other) : |
|  | return self.\_\_dict\_\_ == other.\_\_dict\_\_ and self.\_\_class\_\_ == other.\_\_class\_\_ |
|  |  |
|  | def precondition(self): |
|  | return [ CLEAR(self.Y) , HOLDING(self.X) ] |
|  |  |
|  | def delete(self): |
|  | return [ CLEAR(self.Y) , HOLDING(self.X) ] |
|  |  |
|  | def add(self): |
|  | return [ ARMEMPTY() , ON(self.X,self.Y) ] |
|  |  |
|  |  |
|  | class UnstackOp(Operation): |
|  |  |
|  | def \_\_init\_\_(self, X, Y): |
|  | self.X = X |
|  | self.Y = Y |
|  |  |
|  | def \_\_str\_\_(self): |
|  | return "UNSTACK({X},{Y})".format(X=self.X,Y=self.Y) |
|  |  |
|  | def \_\_repr\_\_(self): |
|  | return self.\_\_str\_\_() |
|  |  |
|  | def \_\_eq\_\_(self, other) : |
|  | return self.\_\_dict\_\_ == other.\_\_dict\_\_ and self.\_\_class\_\_ == other.\_\_class\_\_ |
|  |  |
|  | def precondition(self): |
|  | return [ ARMEMPTY() , ON(self.X,self.Y) , CLEAR(self.X) ] |
|  |  |
|  | def delete(self): |
|  | return [ ARMEMPTY() , ON(self.X,self.Y) ] |
|  |  |
|  | def add(self): |
|  | return [ CLEAR(self.Y) , HOLDING(self.X) ] |
|  |  |
|  |  |
|  | class PickupOp(Operation): |
|  |  |
|  | def \_\_init\_\_(self, X): |
|  | self.X = X |
|  |  |
|  | def \_\_str\_\_(self): |
|  | return "PICKUP({X})".format(X=self.X) |
|  |  |
|  | def \_\_repr\_\_(self): |
|  | return self.\_\_str\_\_() |
|  |  |
|  | def \_\_eq\_\_(self, other) : |
|  | return self.\_\_dict\_\_ == other.\_\_dict\_\_ and self.\_\_class\_\_ == other.\_\_class\_\_ |
|  |  |
|  | def precondition(self): |
|  | return [ CLEAR(self.X) , ONTABLE(self.X) , ARMEMPTY() ] |
|  |  |
|  | def delete(self): |
|  | return [ ARMEMPTY() , ONTABLE(self.X) ] |
|  |  |
|  | def add(self): |
|  | return [ HOLDING(self.X) ] |
|  |  |
|  |  |
|  | class PutdownOp(Operation): |
|  |  |
|  | def \_\_init\_\_(self, X): |
|  | self.X = X |
|  |  |
|  | def \_\_str\_\_(self): |
|  | return "PUTDOWN({X})".format(X=self.X) |
|  |  |
|  | def \_\_repr\_\_(self): |
|  | return self.\_\_str\_\_() |
|  |  |
|  | def \_\_eq\_\_(self, other) : |
|  | return self.\_\_dict\_\_ == other.\_\_dict\_\_ and self.\_\_class\_\_ == other.\_\_class\_\_ |
|  |  |
|  | def precondition(self): |
|  | return [ HOLDING(self.X) ] |
|  |  |
|  | def delete(self): |
|  | return [ HOLDING(self.X) ] |
|  |  |
|  | def add(self): |
|  | return [ ARMEMPTY() , ONTABLE(self.X) ] |
|  |  |
|  |  |
|  | def isPredicate(obj): |
|  | predicates = [ON, ONTABLE, CLEAR, HOLDING, ARMEMPTY] |
|  | for predicate in predicates: |
|  | if isinstance(obj,predicate): |
|  | return True |
|  | return False |
|  |  |
|  | def isOperation(obj): |
|  | operations = [StackOp, UnstackOp, PickupOp, PutdownOp] |
|  | for operation in operations: |
|  | if isinstance(obj,operation): |
|  | return True |
|  | return False |
|  |  |
|  | def arm\_status(world\_state): |
|  | for predicate in world\_state: |
|  | if isinstance(predicate, HOLDING): |
|  | return predicate |
|  | return ARMEMPTY() |
|  |  |
|  |  |
|  | class GoalStackPlanner: |
|  |  |
|  | def \_\_init\_\_(self, initial\_state, goal\_state): |
|  | self.initial\_state = initial\_state |
|  | self.goal\_state = goal\_state |
|  |  |
|  | def get\_steps(self): |
|  |  |
|  | #Store Steps |
|  | steps = [] |
|  |  |
|  | #Program Stack |
|  | stack = [] |
|  |  |
|  | #World State/Knowledge Base |
|  | world\_state = self.initial\_state.copy() |
|  |  |
|  | #Initially push the goal\_state as compound goal onto the stack |
|  | stack.append(self.goal\_state.copy()) |
|  |  |
|  | #Repeat until the stack is empty |
|  | while len(stack)!=0: |
|  |  |
|  | #Get the top of the stack |
|  | stack\_top = stack[-1] |
|  |  |
|  | #If Stack Top is Compound Goal, push its unsatisfied goals onto stack |
|  | if type(stack\_top) is list: |
|  | compound\_goal = stack.pop() |
|  | for goal in compound\_goal: |
|  | if goal not in world\_state: |
|  | stack.append(goal) |
|  |  |
|  | #If Stack Top is an action |
|  | elif isOperation(stack\_top): |
|  |  |
|  | #Peek the operation |
|  | operation = stack[-1] |
|  |  |
|  | all\_preconditions\_satisfied = True |
|  |  |
|  | #Check if any precondition is unsatisfied and push it onto program stack |
|  | for predicate in operation.delete(): |
|  | if predicate not in world\_state: |
|  | all\_preconditions\_satisfied = False |
|  | stack.append(predicate) |
|  |  |
|  | #If all preconditions are satisfied, pop operation from stack and execute it |
|  | if all\_preconditions\_satisfied: |
|  |  |
|  | stack.pop() |
|  | steps.append(operation) |
|  |  |
|  | for predicate in operation.delete(): |
|  | world\_state.remove(predicate) |
|  | for predicate in operation.add(): |
|  | world\_state.append(predicate) |
|  |  |
|  |  |
|  | #If Stack Top is a single satisfied goal |
|  | elif stack\_top in world\_state: |
|  | stack.pop() |
|  |  |
|  | #If Stack Top is a single unsatisfied goal |
|  | else: |
|  | unsatisfied\_goal = stack.pop() |
|  |  |
|  | #Replace Unsatisfied Goal with an action that can complete it |
|  | action = unsatisfied\_goal.get\_action(world\_state) |
|  |  |
|  | stack.append(action) |
|  | #Push Precondition on the stack |
|  | for predicate in action.precondition(): |
|  | if predicate not in world\_state: |
|  | stack.append(predicate) |
|  |  |
|  | return steps |
|  |  |
|  | if \_\_name\_\_ == '\_\_main\_\_': |
|  | initial\_state = [ |
|  | ON('B','A'), |
|  | ONTABLE('A'),ONTABLE('C'),ONTABLE('D'), |
|  | CLEAR('B'),CLEAR('C'),CLEAR('D'), |
|  | ARMEMPTY() |
|  | ] |
|  |  |
|  | goal\_state = [ |
|  | ON('B','D'),ON('C','A'), |
|  | ONTABLE('D'),ONTABLE('A'), |
|  | CLEAR('B'),CLEAR('C'), |
|  | ARMEMPTY() |
|  | ] |
|  |  |
|  | goal\_stack = GoalStackPlanner(initial\_state=initial\_state, goal\_state=goal\_state) |
|  | steps = goal\_stack.get\_steps() |
|  | print(steps) |

**Output:**



**Exp no 11: Implementation of Learning algorithms.**

**Link:**

<https://towardsdatascience.com/linear-regression-on-boston-housing-dataset-f409b7e4a155>

**Source code:**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

import seaborn as sns

%matplotlib inline

from sklearn.datasets import load\_boston

boston\_dataset = load\_boston()

boston = pd.DataFrame(boston\_dataset.data, columns=boston\_dataset.feature\_names)

boston.head()

boston['MEDV'] = boston\_dataset.target

boston.isnull().sum()

sns.set(rc={'figure.figsize':(11.7,8.27)})

sns.distplot(boston['MEDV'], bins=30)

plt.show()

correlation\_matrix = boston.corr().round(2)

# annot = True to print the values inside the square

sns.heatmap(data=correlation\_matrix, annot=True)

plt.figure(figsize=(20, 5))

features = ['LSTAT', 'RM']

target = boston['MEDV']

for i, col in enumerate(features):

plt.subplot(1, len(features) , i+1)

x = boston[col]

y = target

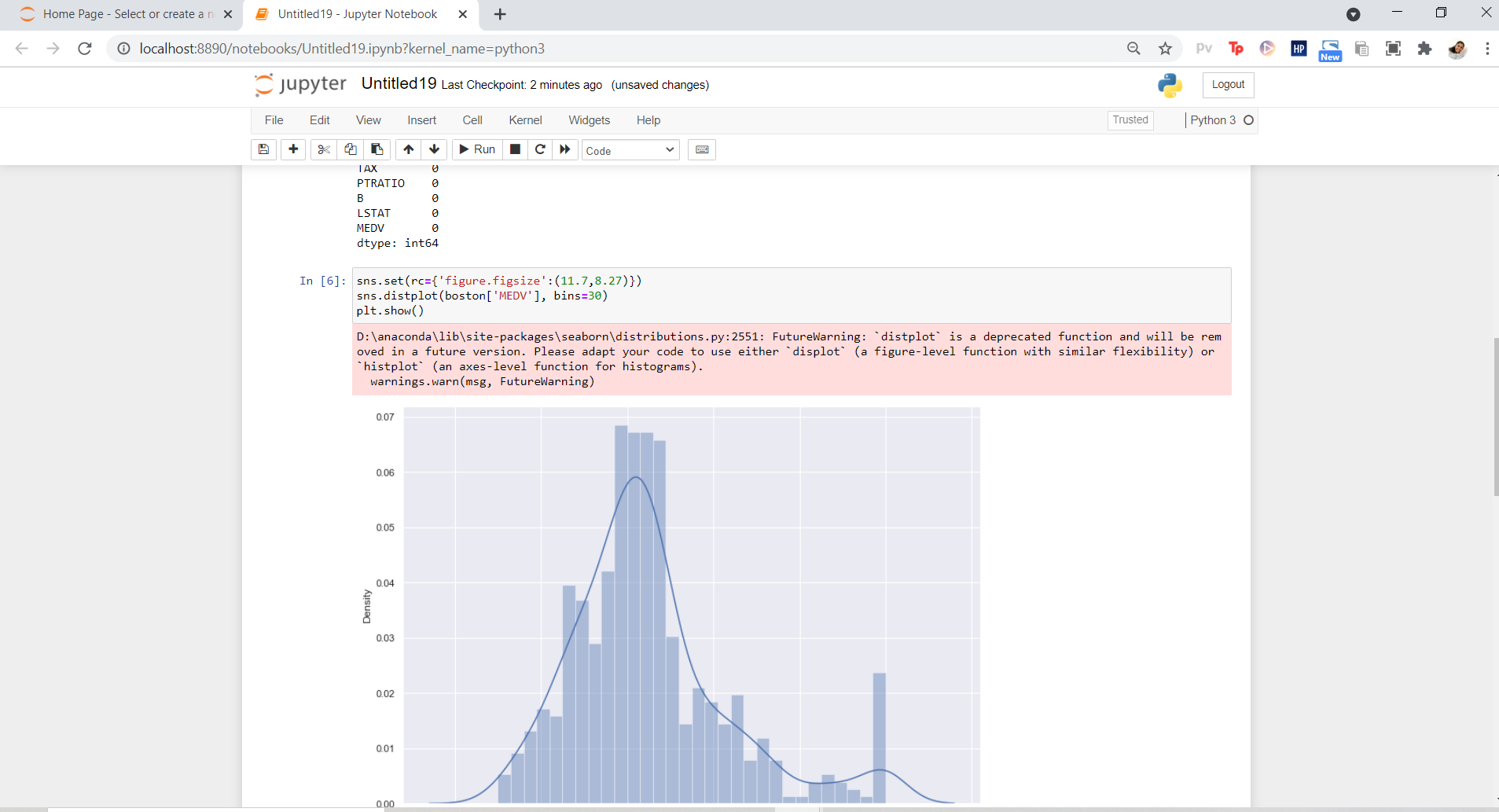
plt.scatter(x, y, marker='o')

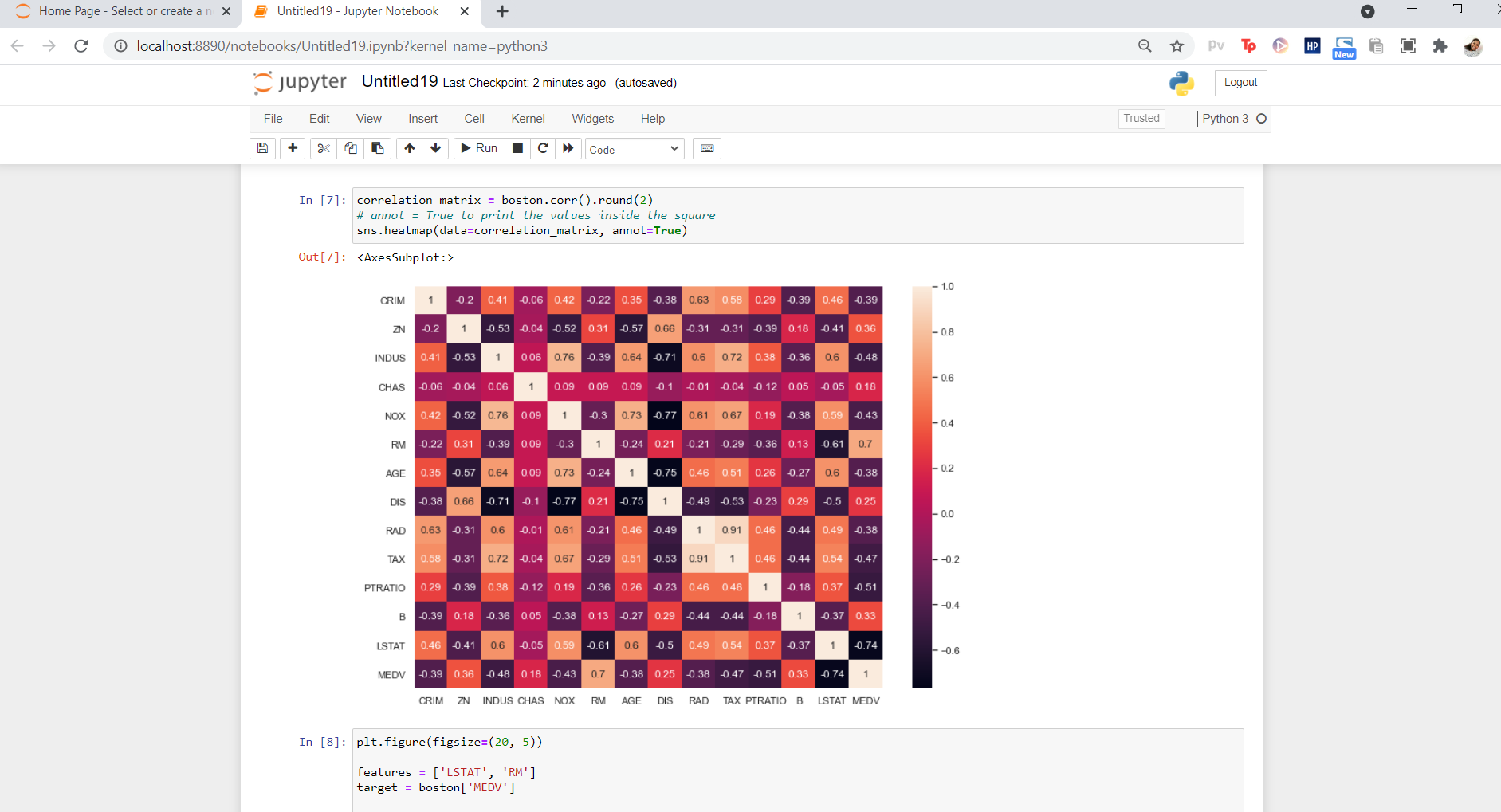
plt.title(col)

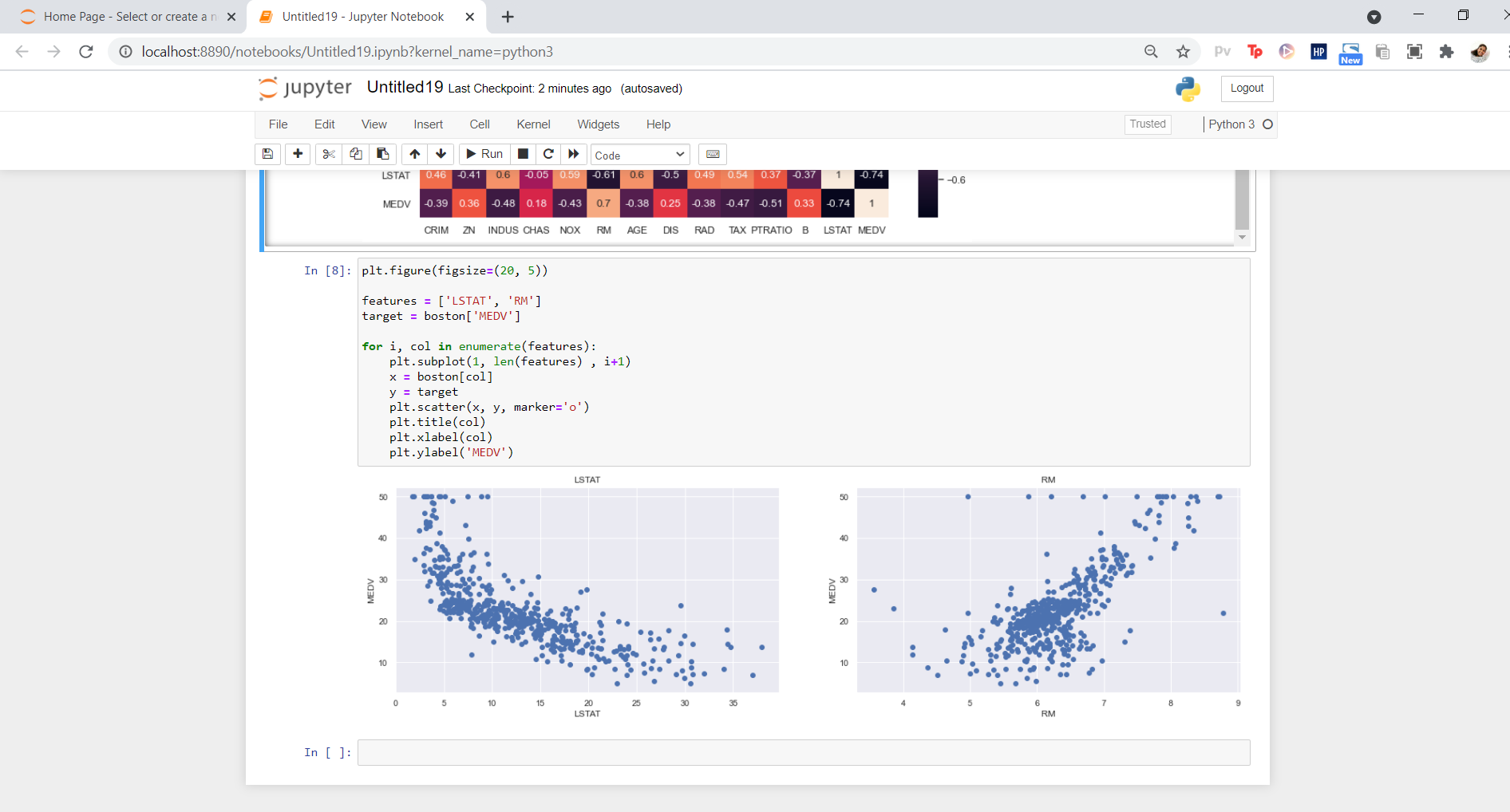
plt.xlabel(col)

plt.ylabel('MEDV')

**Output:**







**Exp no 12: Development of ensemble model.**

**Link:**

<https://www.datacamp.com/community/tutorials/adaboost-classifier-python>

**Source Code:**

# Load libraries

from sklearn.ensemble import AdaBoostClassifier

from sklearn import datasets

# Import train\_test\_split function

from sklearn.model\_selection import train\_test\_split

#Import scikit-learn metrics module for accuracy calculation

from sklearn import metrics

# Load data

iris = datasets.load\_iris()

X = iris.data

y = iris.target

# Split dataset into training set and test set

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3) # 70% training and 30% test

# Create adaboost classifer object

abc = AdaBoostClassifier(n\_estimators=50,

learning\_rate=1)

# Train Adaboost Classifer

model = abc.fit(X\_train, y\_train)

#Predict the response for test dataset

y\_pred = model.predict(X\_test)

# Model Accuracy, how often is the classifier correct?

print("Accuracy:",metrics.accuracy\_score(y\_test, y\_pred))

# Load libraries

from sklearn.ensemble import AdaBoostClassifier

# Import Support Vector Classifier

from sklearn.svm import SVC

#Import scikit-learn metrics module for accuracy calculation

from sklearn import metrics

svc=SVC(probability=True, kernel='linear')

# Create adaboost classifer object

abc =AdaBoostClassifier(n\_estimators=50, base\_estimator=svc,learning\_rate=1)

# Train Adaboost Classifer

model = abc.fit(X\_train, y\_train)

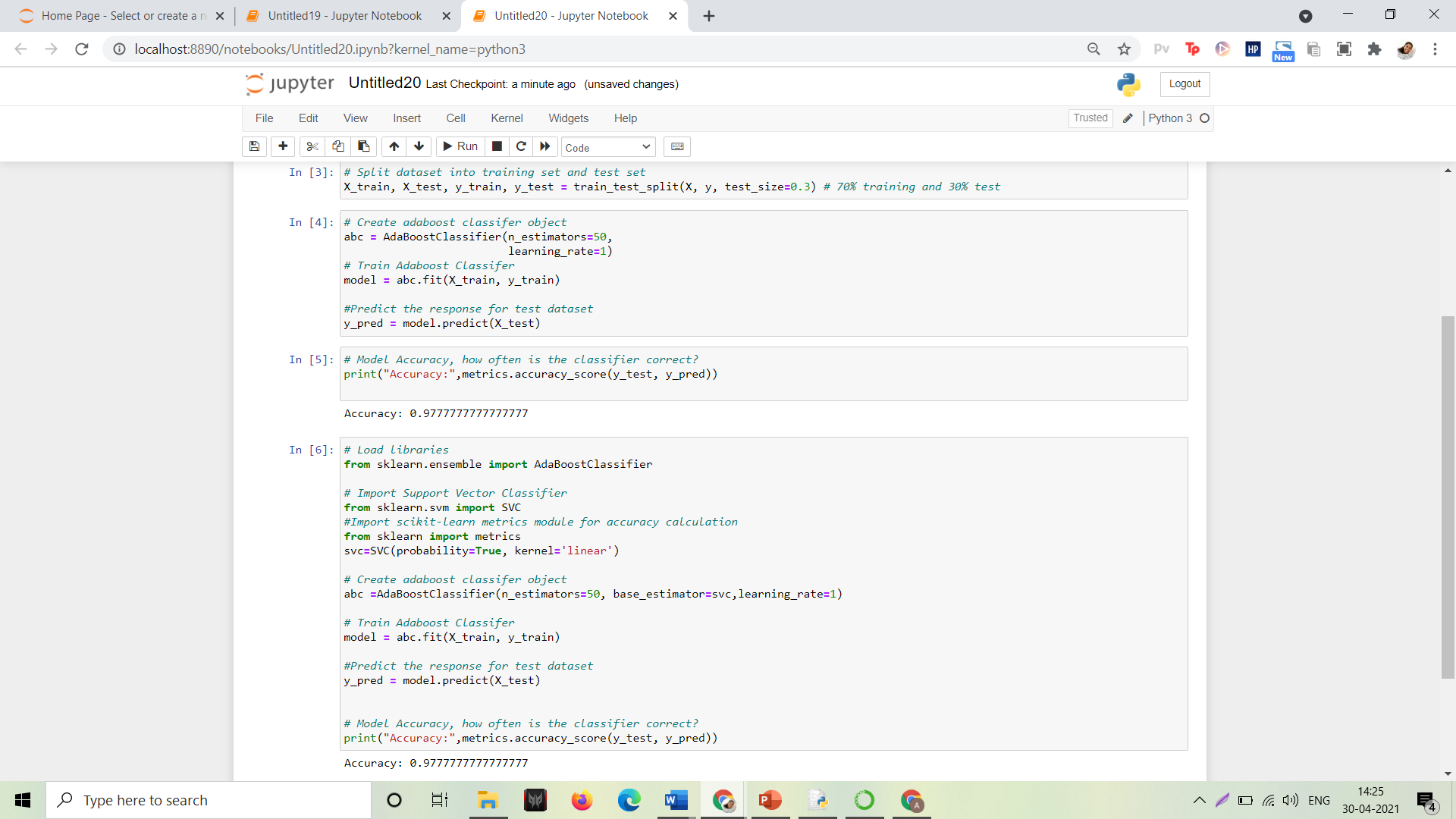
#Predict the response for test dataset

y\_pred = model.predict(X\_test)

# Model Accuracy, how often is the classifier correct?

print("Accuracy:",metrics.accuracy\_score(y\_test, y\_pred))

**Output:**



**Exp no 13: Natural language processing-Levels of NLP - Text Preprocessing**

**Link:**

[Text Preprocessing in Python | Set - 1 - GeeksforGeeks](https://www.geeksforgeeks.org/text-preprocessing-in-python-set-1/)

[Hands-On Lab On Text Preprocessing in NLP Using Python | by @pramodAIML | Predict | Medium](https://medium.com/predict/hands-on-lab-on-text-preprocessing-in-nlp-using-python-ea419a00396e)

# import the necessary libraries

import nltk

import string

import re

### **Text Lowercase:**

def text\_lowercase(text):

return text.lower()

input\_str = "Hey, did you know that the summer break is coming? Amazing right !! It's only 5 more days !!"

text\_lowercase (input\_str)

**OUTPUT:**

Input: “Hey, did you know that the summer break is coming? Amazing right!! It’s only 5 more days!!”

Output: “hey, did you know that the summer break is coming? amazing right!! it’s only 5 more days!!”

**2. Remove numbers:**

# Remove numbers

def remove\_numbers(text):

result = re.sub(r'\d+', '', text)

return result

input\_str = "There are 3 balls in this bag, and 12 in the other one."

remove\_numbers(input\_str)

**Input**: “There are 3 balls in this bag, and 12 in the other one.”

**Output**: ‘There are balls in this bag, and in the other one.’

1. **Convert numbers into words**

# import the inflect library

import inflect

p = inflect.engine()

# convert number into words

def convert\_number(text):

# split string into list of words

temp\_str = text.split()

# initialise empty list

new\_string = []

for word in temp\_str:

# if word is a digit, convert the digit

# to numbers and append into the new\_string list

if word.isdigit():

temp = p.number\_to\_words(word)

new\_string.append(temp)

# append the word as it is

else:

new\_string.append(word)

# join the words of new\_string to form a string

temp\_str = ' '.join(new\_string)

return temp\_str

input\_str = 'There are 3 balls in this bag, and 12 in the other one.'

convert\_number(input\_str)

**Input**: “There are 3 balls in this bag, and 12 in the other one.”

**Output**: “There are three balls in this bag, and twelve in the other one.”

1. **Remove punctuation:**

# remove punctuation

def remove\_punctuation(text):

translator = str.maketrans('', '', string.punctuation)

return text.translate(translator)

input\_str = "Hey, did you know that the summer break is coming? Amazing right !! It's only 5 more days !!"

remove\_punctuation(input\_str)

**Input**: “Hey, did you know that the summer break is coming? Amazing right!! It’s only 5 more days!!”

**Output**: “Hey did you know that the summer break is coming Amazing right Its only 5 more days”

1. **Remove whitespaces:**

# remove whitespace from text

def remove\_whitespace(text):

return " ".join(text.split())

input\_str = " we don't need the given questions"

remove\_whitespace(input\_str)

**Input:** " we don't need the given questions"

**Output**: "we don't need the given questions"

1. **Remove default stopwords:**

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

# remove stopwords function

def remove\_stopwords(text):

stop\_words = set(stopwords.words("english"))

word\_tokens = word\_tokenize(text)

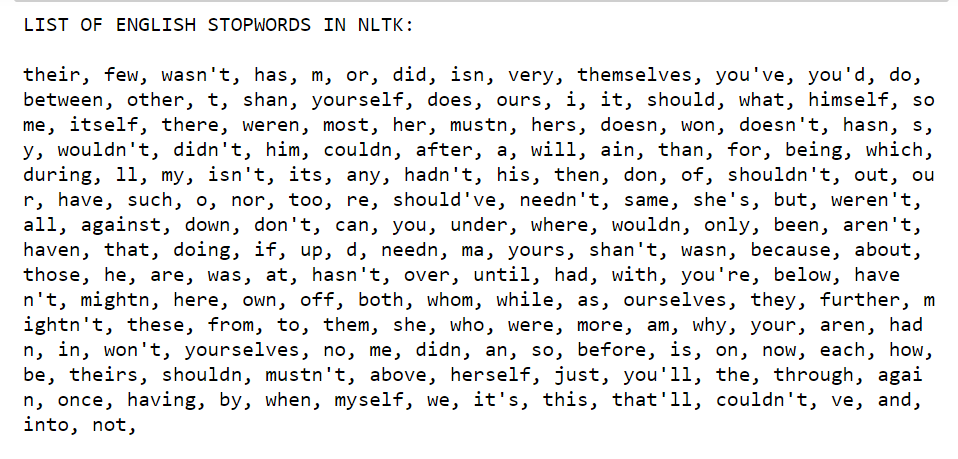
filtered\_text = [word for word in word\_tokens if word not in stop\_words]

return filtered\_text

example\_text = "This is a sample sentence and we are going to remove the stopwords from this."

remove\_stopwords(example\_text)

**The NLTK library has a set of stopwords and we can use these to remove stopwords from our text and return a list of word tokens.**



**Input:** “This is a sample sentence and we are going to remove the stopwords from this”

**Output:** [‘This’, ‘sample’, ‘sentence’, ‘going’, ‘remove’, ‘stopwords’]

1. **Stemming:**

Stemming is the process of getting the root form of a word. Stem or root is the part to which inflectional affixes (-ed, -ize, -de, -s, etc.) are added. The stem of a word is created by removing the prefix or suffix of a word. So, stemming a word may not result in actual words.

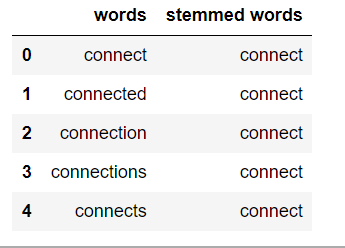
**Example:**

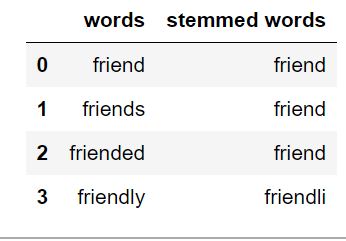
books ---> book

looked ---> look

denied ---> deni

flies ---> fli





from nltk.stem.porter import PorterStemmer

from nltk.tokenize import word\_tokenize

stemmer = PorterStemmer()

# stem words in the list of tokenized words

def stem\_words(text):

word\_tokens = word\_tokenize(text)

stems = [stemmer.stem(word) for word in word\_tokens]

return stems

text = 'data science uses scientific methods algorithms and many types of processes'

stem\_words(text)

**Input:** ‘data science uses scientific methods algorithms and many types of processes’

**Output:** [‘data’, ‘scienc’, ‘use’, ‘scientif’, ‘method’, ‘algorithm’, ‘and’, ‘mani’, ‘type’, ‘of’, ‘process’]

1. **Lemmatization:**

Lemmatization also converts a word to its root form. The only difference is that lemmatization ensures that the root word belongs to the language.

from nltk.stem import WordNetLemmatizer

from nltk.tokenize import word\_tokenize

lemmatizer = WordNetLemmatizer()

# lemmatize string

def lemmatize\_word(text):

word\_tokens = word\_tokenize(text)

# provide context i.e. part-of-speech

lemmas = [lemmatizer.lemmatize(word, pos ='v') for word in word\_tokens]

return lemmas

text = 'data science uses scientific methods algorithms and many types of processes'

lemmatize\_word(text)

**Input:** ‘data science uses scientific methods algorithms and many types of processes’

**Output**: [‘data’, ‘science’, ‘use’, ‘scientific’, ‘methods’, ‘algorithms’, ‘and’, ‘many’, ‘type’, ‘of’, ‘process’]

Experiment No 14 **Implementation of NLP programs**

Date :

|  |  |
| --- | --- |
| **Aim** | To discriminate between ham/spam messages automatically using UCI  datasets. |
| **Software Requirements** | Using Google Colab |

|  |  |
| --- | --- |
| **Code** (Implementation of Linear Regression algorithm to predict students score using the given dataset) | |
| 1. import pandas as pd 2. import numpy as np 3. import string 4. import seaborn as sns 5. import matplotlib.pyplot as plt 6. from nltk.corpus import stopwords 7. from sklearn.feature\_extraction.text import CountVectorizer 8. from sklearn.feature\_extraction.text import TfidfTransformer 9. from sklearn.model\_selection import train\_test\_split 10. from sklearn.svm import SVC 11. from collections import Counter 12. from sklearn.metrics import classification\_report,confusion\_matrix 13. from sklearn.model\_selection import GridSearchCV 14. %matplotlib inline 15. # Load data 16. data = pd.read\_excel('data.xlsx') 17. # Rename names columns 18. data.columns = ['label', 'messages'] 19. data["length"] = data["messages"].apply(len) 20. data.sort\_values(by='length', ascending=False).head(10) 21. data.hist(column = 'length', by ='label',figsize=(12,4), bins = 5) 22. def transform\_message(message): 23. message\_not\_punc = [] # Message without punctuation 24 i = 0 24. for punctuation in message: 25. if punctuation not in string.punctuation: 26. message\_not\_punc.append(punctuation) 27. # Join words again to form the string. 28. message\_not\_punc = ''.join(message\_not\_punc) 30 29. # Remove any stopwords for message\_not\_punc, but first we should 30. # to transform this into the list. 31. message\_clean = list(message\_not\_punc.split(" ")) 32. while i <= len(message\_clean): 33. for mess in message\_clean: 34. if mess.lower() in stopwords.words('english'): 35. message\_clean.remove(mess) 38 i =i +1 36. return message\_clean 37. vectorization = CountVectorizer(analyzer = transform\_message ) 38. X = vectorization.fit(data['messages']) 42   43 X\_transform = X.transform([data['messages']]) |  |

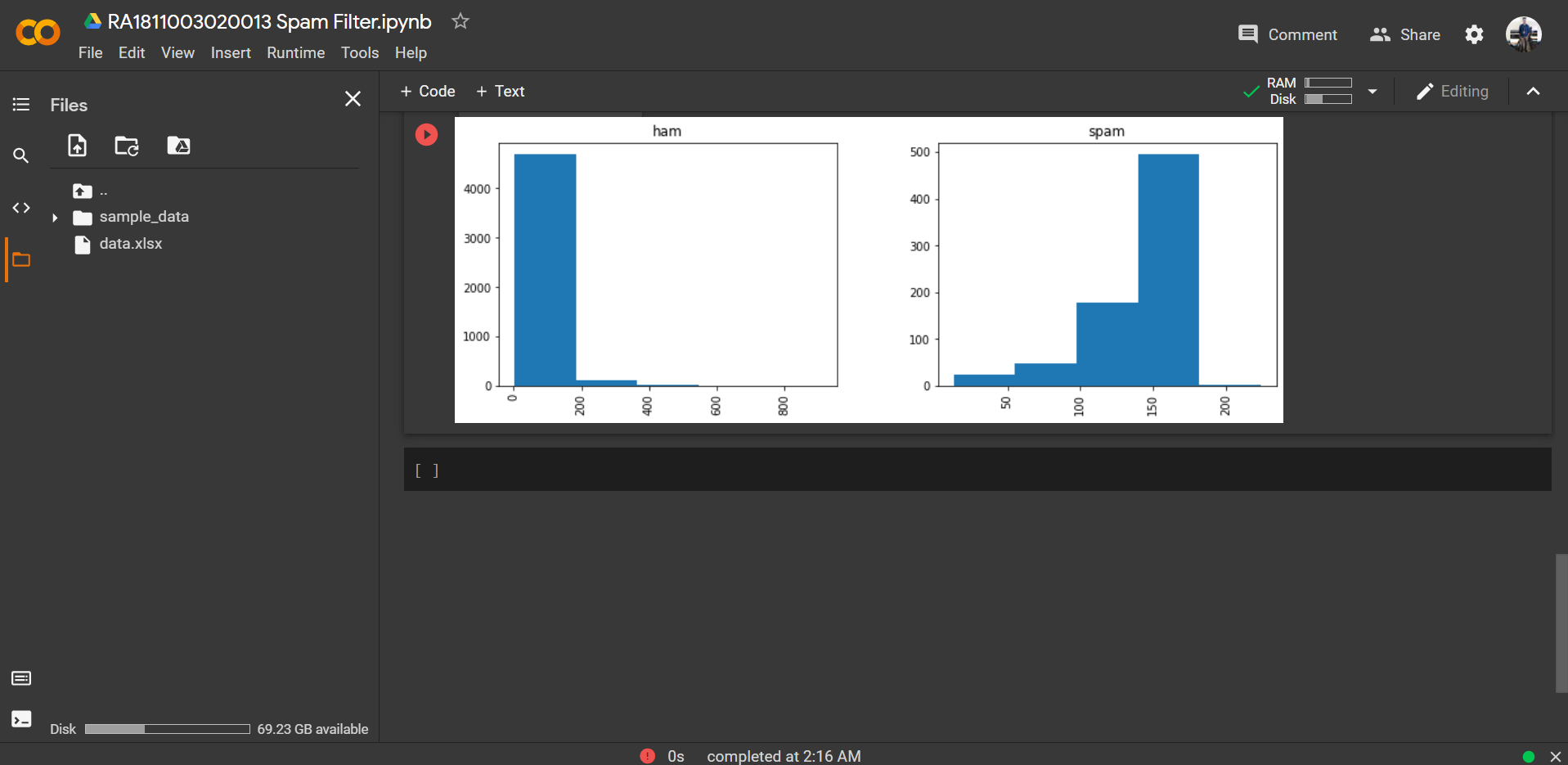
Page No : 45

18CSC305J AI LAB

|  |  |  |
| --- | --- | --- |
| 44 | |  |
| 45 | # TF-IDF |  |
| 46 | tfidf\_transformer = TfidfTransformer().fit(X\_transform) |
| 47 | X\_tfidf = tfidf\_transformer.transform(X\_transform) |
| 48 | print(X\_tfidf.shape) |
| 49 | # Classification Model |
| 50 |  |
| 51 | X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_tfidf, data['messages'], test\_size=0.30, |
|  | random\_state = 50) |
| 52 | clf = SVC(kernel='linear').fit(X\_train, y\_train) |
| 53 | # Test model |
| 54 |  |
| 55 | predictions = clf.predict(X\_test) |
| 56 | print('predicted', predictions) |
| 57 | # Is our model reliable? |
| 58 |  |
| 59 | print (classification\_report(y\_test, predictions)) |
| **60** | print(confusion\_matrix(y\_test,predictions)) |

Page No : 46

# O utput of the above program

18CSC305J AI LAB

# Applying deep learning method for Automatic Handwriting recognition

Experiment No 15

Date :

|  |  |
| --- | --- |
| **Aim** | Applying deep learning method for Automatic Handwriting recognition |
| **Software Requirements** | Using Google Colab |

|  |  |  |
| --- | --- | --- |
| **Code** (Implementation of Linear Regression algorithm to predict students score using the given dataset) | | |
| 1 | import argparse |  |
| 2 | import json |
| 3 |  |
| 4 | import cv2 |
| 5 | import editdistance |
| 6 | from path import Path |
| 7 |  |
| 8 | from DataLoaderIAM import DataLoaderIAM, Batch |
| 9 | from Model import Model, DecoderType |
| 10 | from SamplePreprocessor import preprocess |
| 11 |  |
| 12 |  |
| 13 | class FilePaths: |
| 14 | "filenames and paths to data" |
| 15 | fnCharList = '../model/charList.txt' |
| 16 | fnSummary = '../model/summary.json' |
| 17 | fnInfer = '../data/test.png' |
| 18 | fnCorpus = '../data/corpus.txt' |
| 19 |  |
| 20 |  |
| 21 | def write\_summary(charErrorRates, wordAccuracies): |
| 22 | with open(FilePaths.fnSummary, 'w') as f: |
| 23 | json.dump({'charErrorRates': charErrorRates, 'wordAccuracies': wordAccuracies}, f) |
| 24 |  |
| 25 |  |
| 26 | def train(model, loader): |
| 27 | "train NN" |
| 28 | epoch = 0 # number of training epochs since start |
| 29 | summaryCharErrorRates = [] |
| 30 | summaryWordAccuracies = [] |
| 31 | bestCharErrorRate = float('inf') # best valdiation character error rate |
| 32 | noImprovementSince = 0 # number of epochs no improvement of character error rate occured |
| 33 | earlyStopping = 25 # stop training after this number of epochs without improvement |
| 34 | while True: |
| 35 | epoch += 1 |
| 36 | print('Epoch:', epoch) |
| 37 |  |
| 38 | # train |
| 39 | print('Train NN') |
| 40 | loader.trainSet() |
| 41 | while loader.hasNext(): |
| 42 | iterInfo = loader.getIteratorInfo() |
| 43 | batch = loader.getNext() |
| 44 | loss = model.trainBatch(batch) |

|  |  |
| --- | --- |
| 45 print(f'Epoch: {epoch} Batch: {iterInfo[0]}/{iterInfo[1]} Loss: {loss}') |  |
| 46   1. # validate 2. charErrorRate, wordAccuracy = validate(model, loader) 49 3. # write summary 4. summaryCharErrorRates.append(charErrorRate) 5. summaryWordAccuracies.append(wordAccuracy) 6. write\_summary(summaryCharErrorRates, summaryWordAccuracies) 54 7. # if best validation accuracy so far, save model parameters 8. if charErrorRate < bestCharErrorRate: 9. print('Character error rate improved, save model') 10. bestCharErrorRate = charErrorRate 11. noImprovementSince = 0 12. model.save() 13. else: 14. print(f'Character error rate not improved, best so far: {charErrorRate \* 100.0}%') 15. noImprovementSince += 1 64 16. # stop training if no more improvement in the last x epochs 17. if noImprovementSince >= earlyStopping: 18. print(f'No more improvement since {earlyStopping} epochs. Training stopped.') 19. break   69  70   1. def validate(model, loader): 2. "validate NN" 3. print('Validate NN') 4. loader.validationSet() 5. numCharErr = 0 6. numCharTotal = 0 7. numWordOK = 0 8. numWordTotal = 0 9. while loader.hasNext(): 10. iterInfo = loader.getIteratorInfo() 11. print(f'Batch: {iterInfo[0]} / {iterInfo[1]}') 12. batch = loader.getNext() 13. (recognized, \_) = model.inferBatch(batch) 84 14. print('Ground truth -> Recognized') 15. for i in range(len(recognized)): 16. numWordOK += 1 if batch.gtTexts[i] == recognized[i] else 0 17. numWordTotal += 1 18. dist = editdistance.eval(recognized[i], batch.gtTexts[i]) 19. numCharErr += dist 20. numCharTotal += len(batch.gtTexts[i]) 21. print('[OK]' if dist == 0 else '[ERR:%d]' % dist, '"' + batch.gtTexts[i] + '"', '->', 22. '"' + recognized[i] + '"') 94 23. # print validation result 24. charErrorRate = numCharErr / numCharTotal 25. wordAccuracy = numWordOK / numWordTotal 26. print(f'Character error rate: {charErrorRate \* 100.0}%. Word accuracy: {wordAccuracy \* 100.0}%.') 27. return charErrorRate, wordAccuracy 100   101   1. def infer(model, fnImg): 2. "recognize text in image provided by file path" |  |

|  |  |
| --- | --- |
| 104 img = preprocess(cv2.imread(fnImg, cv2.IMREAD\_GRAYSCALE), Model.imgSize) |  |
| 1. batch = Batch(None, [img]) 2. (recognized, probability) = model.inferBatch(batch, True) 3. print(f'Recognized: "{recognized[0]}"') 4. print(f'Probability: {probability[0]}') 109   110   1. def main(): 2. "main function" 3. parser = argparse.ArgumentParser() 4. parser.add\_argument('--train', help='train the NN', action='store\_true') 5. parser.add\_argument('--validate', help='validate the NN', action='store\_true') 6. parser.add\_argument('--decoder', choices=['bestpath', 'beamsearch', 'wordbeamsearch'], default='bestpath', 7. help='CTC decoder') 8. parser.add\_argument('--batch\_size', help='batch size', type=int, default=100) 9. parser.add\_argument('--data\_dir', help='directory containing IAM dataset', type=Path, required=False) 10. parser.add\_argument('--fast', help='use lmdb to load images', action='store\_true') 11. parser.add\_argument('--dump', help='dump output of NN to CSV file(s)', action='store\_true') 12. args = parser.parse\_args() 123 13. # set chosen CTC decoder 14. if args.decoder == 'bestpath': 15. decoderType = DecoderType.BestPath 16. elif args.decoder == 'beamsearch': 17. decoderType = DecoderType.BeamSearch 18. elif args.decoder == 'wordbeamsearch': 19. decoderType = DecoderType.WordBeamSearch 131 20. # train or validate on IAM dataset 21. if args.train or args.validate: 22. # load training data, create TF model 23. loader = DataLoaderIAM(args.data\_dir, args.batch\_size, Model.imgSize, Model.maxTextLen, args.fast) 136 24. # save characters of model for inference mode 25. open(FilePaths.fnCharList, 'w').write(str().join(loader.charList)) 139 26. # save words contained in dataset into file 27. open(FilePaths.fnCorpus, 'w').write(str(' ').join(loader.trainWords + loader.validationWords)) 142 28. # execute training or validation 29. if args.train: 30. model = Model(loader.charList, decoderType) 31. train(model, loader) 32. elif args.validate: 33. model = Model(loader.charList, decoderType, mustRestore=True) 34. validate(model, loader) 150 35. # infer text on test image 36. else: 37. model = Model(open(FilePaths.fnCharList).read(), decoderType, mustRestore=True, dump=args.dump) 38. infer(model, FilePaths.fnInfer) 155   156   1. if name == ' main ': 2. main() |  |

# O utput of the above progr