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 603203, KANCHEEPURAM DISTRICT

SCHOOL OF COMPUTING

DEPARTMENT OF DATA SCIENCE AND BUSINESS SYSTEMS

Course Code: 18CSE305J

Course Name: Artificial Intelligence

LAB REPORT

Name: Deeksha Rawat
Registration Number: RA1911027010005
Section: N1
Branch: CSE – Big Data Analytics

TABLE OF CONTENT

Date:04/01/22 **Ex No:1**	**Title of the Lab** **TOY PROBLEMS**	**Name: Deeksha Rawat** **Reg Number:RA1911027010005** **Section: N1** **Lab Batch: 1** **Day Order: 2**

Q1) Camel and Bananas Puzzle

AIM: To find the solution of a Toy Problem known as Camel and Bananas Puzzle

Description of the Concept or Problem given:

A person has 3000 bananas and a camel. The person wants to transport the maximum number of bananas to a destination which is 1000 KMs away, using only the camel as a mode of transportation. The camel cannot carry more than 1000 bananas at a time and eats a banana every km it travels. What is the maximum number of bananas that can be transferred to the destination using only camel ?

Manual Solution:

Let’s see what we can infer from the question:

- We have a total of 3000 bananas.
- The destination is 1000KMs
- Only 1 mode of transport.
- Camel can carry a maximum of 1000 banana at a time.
- Camel eats a banana every km it travels.

With all these points, we can say that person won’t be able to transfer any banana to the destination as the camel is going to eat all the banana on its way to the destination.

But the trick here is to have intermediate drop points, then, the camel can make several short trips in between.

Also, we try to maintain the number of bananas at each point to be multiple of 1000.

Let’s have 2 drop points in between the source and destination.

With 3000 bananas at the source. 2000 at a first intermediate point and 1000 at 2nd intermediate point.

- To go from source to IP1 point camel has to take a total of 5 trips 3 forward and 2 backward. Since we have 3000 bananas to transport.
- The same way from IP1 to IP2 camel has to take a total of 3 trips, 2 forward and 1 backward. Since we have 2000 bananas to transport.
- At last from IP2 to a destination only 1 forward move.

Let’s see the total number of bananas consumed at every point.

- From the source to IP1 its 5x bananas, as the distance between the source and IP1 is x km and the camel had 5 trips.
- From IP1 to IP2 its 3y bananas, as the distance between IP1 and IP2 is y km and the camel had 3 trips.
- From IP2 to destination its z bananas.

We now try to calculate the distance between the points:

1. 3000 – 5x = 2000 so we get x = 200
2. 2000-3y = 1000 so we get y = 333.33 but here the distance is also the number of bananas and it cannot be fraction so we take y =333 and at IP2 we have the number of bananas equal 1001, so its 2000-3y = 1001
3. So the remaining distance to the market is 1000 -x-y =z i.e 1000-200-333 => z =467.

Now, there are 1001 bananas at IP2.

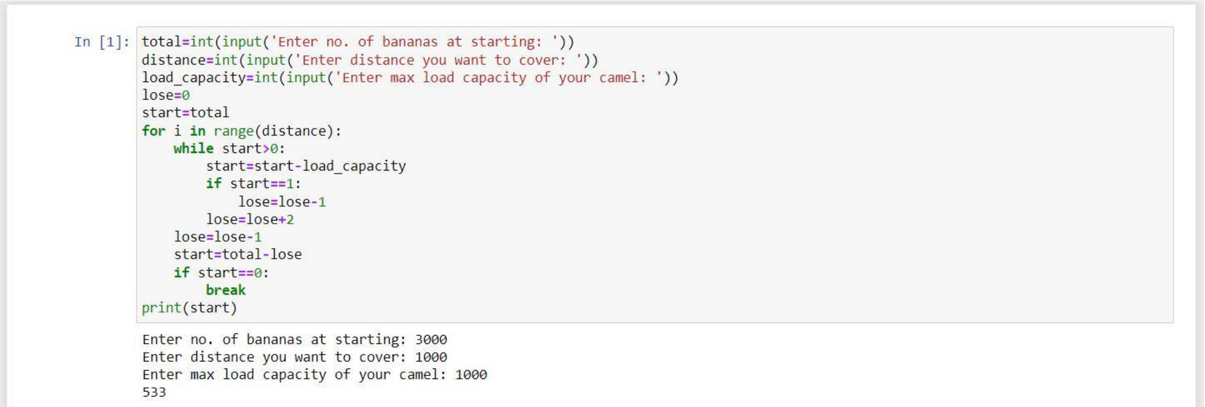
So from IP2 to the destination point camel eats 467 bananas. The remaining bananas are 1001-467=534.

So the maximum number of bananas that can be transferred is 534.

Program Implementation [Coding]

```
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-lose
    if start==0:
        break
print(start)
```

Screenshots of the Outputs



```
In [1]: 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-lose
            if start==0:
                break
        print(start)
```

```
Enter no. of bananas at starting: 3000
Enter distance you want to cover: 1000
Enter max load capacity of your camel: 1000
533
```

Result: The solution for camels and bananas problem is found.

Q2. 3 Water Jug Problem

AIM: To find the solution of a Toy Problem known as 3 Water jug Problem

Description of the Concept or Problem given:

You have three jugs. They can hold 12 liters, 8 liters, and 5 liters of water respectively. 12-liter jug is full of water, but you need to split it in half using only the jugs on hand. How can you split up the water to give away exactly 6 liter, and keep 6 liter?

Manual Solution

Fill 8 liter jug with water from 12 liter jug and pour it into 5 liter Jug. (Now, 8 liter jug contains 3 liters water.)

Empty 5 liter jug pouring water back to 12 liter jug back and fill 3 liter water from 8 liter jug into it. At this point – 8 liter jag is empty. 5 liter jug contains 3 liters water.

Now, fill 8 liter jug with water from 12 liter jug again.

Pour it into 5 liter jug. Since 5 liter jug already contains 3 liter, so, it will accommodate more 2 liters water from 8 liter jug. At this point – 8 liter jag contains 6 liters water and 5 liter jug contains 5 liters water.

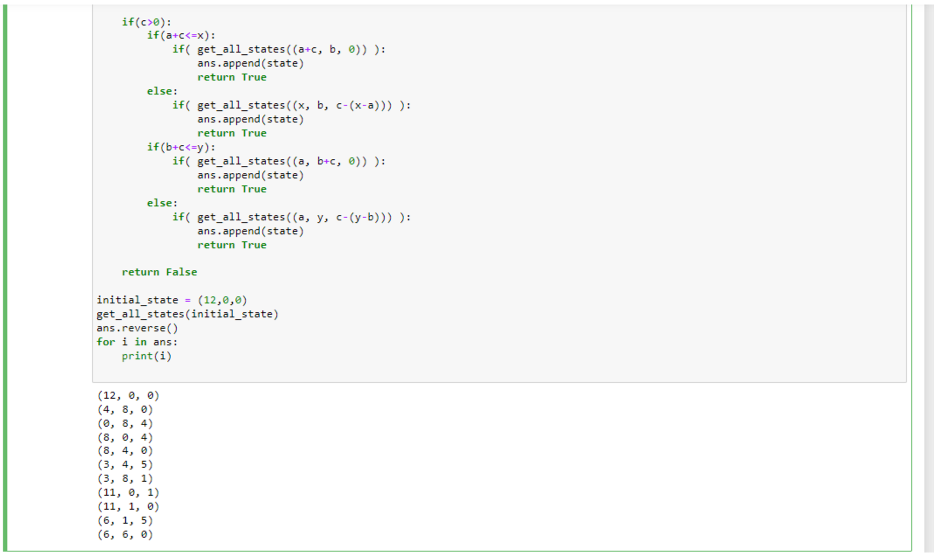
Empty 5 liter jug pouring water back to 12 liter jug back. At this point – 8 liter jug contains 6 liters and 12 liter Jug contains 6 liters water.

Program Implementation [Coding]

```
capacity = (12,8,5)
x = capacity[0]
y = capacity[1]
```

```
z = capacity[2]
memory = {}
ans = []

def get_all_states(state):
    a = state[0]
    b = state[1]
    c = state[2]

    if(a==6 and b==6):
        ans.append(state)
        return True
    if((a,b,c) in memory):
        return False

    memory[(a,b,c)] = 1

    if(a>0):
        if(a+b<=y):
            if( get_all_states((0,a+b,c)) ):
                ans.append(state)
                return True
        else:
            if( get_all_states((a-(y-b), y, c)) ):
                ans.append(state)
                return True
        if(a+c<=z):
            if( get_all_states((0,b,a+c)) ):
                ans.append(state)
                return True
        else:
            if( get_all_states((a-(z-c), b, z)) ):
                ans.append(state)
                return True
    if(b>0):
        if(a+b<=x):
            if( get_all_states((a+b, 0, c)) ):
                ans.append(state)
                return True
        else:
            if( get_all_states((x, b-(x-a), c)) ):
                ans.append(state)
                return True
        if(b+c<=z):
            if( get_all_states((a, 0, b+c)) ):
                ans.append(state)
                return True
        else:
            if( get_all_states((a, b-(z-c), z)) ):
                ans.append(state)
                return True

    if(c>0):
        if(a+c<=x):
            if( get_all_states((a+c, b, 0)) ):
                ans.append(state)
                return True
        else:
            if( get_all_states((x, b, c-(x-a))) ):
                ans.append(state)
                return True
        if(b+c<=y):
            if( get_all_states((a, b+c, 0)) ):
                ans.append(state)
                return True
        else:
            if( get_all_states((a, y, c-(y-b))) ):
                ans.append(state)
                return True

    return False

initial_state = (12,0,0)
get_all_states(initial_state)
ans.reverse()
for i in ans:
    print(i)
```

Screenshots of the Outputs



```
    if(c>0):
        if(a+c<=x):
            if( get_all_states((a+c, b, 0)) ):
                ans.append(state)
                return True
        else:
            if( get_all_states((x, b, c-(x-a))) ):
                ans.append(state)
                return True
        if(b+c<=y):
            if( get_all_states((a, b+c, 0)) ):
                ans.append(state)
                return True
        else:
            if( get_all_states((a, y, c-(y-b))) ):
                ans.append(state)
                return True

    return False

initial_state = (12,0,0)
get_all_states(initial_state)
ans.reverse()
for i in ans:
    print(i)
```

```
(12, 0, 0)
(4, 8, 0)
(0, 8, 4)
(8, 0, 4)
(8, 4, 0)
(3, 4, 5)
(3, 8, 1)
(11, 0, 1)
(11, 1, 0)
(6, 1, 5)
(6, 6, 0)
```

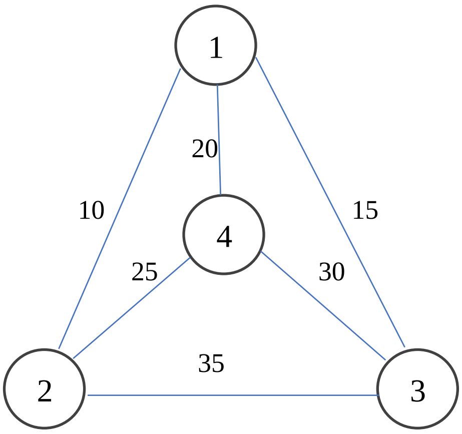
Result: The solution for the water jug problem is found.

Date:16/01/22 **Ex No:2**	**Title of the Lab** **Traveling Salesman and Medical Diagnosis System**	**Name: Deeksha Rawat** **Reg Number:RA1911027010005** **Section: N1** **Lab Batch: 1** **Day Order: 2**

Q1) Traveling Salesman

AIM: To implement Traveling Salesman Problem in Python

Description of the Concept or Problem given:



The goal is to find the shortest possible route that visits each city exactly once and returns to the origin city.

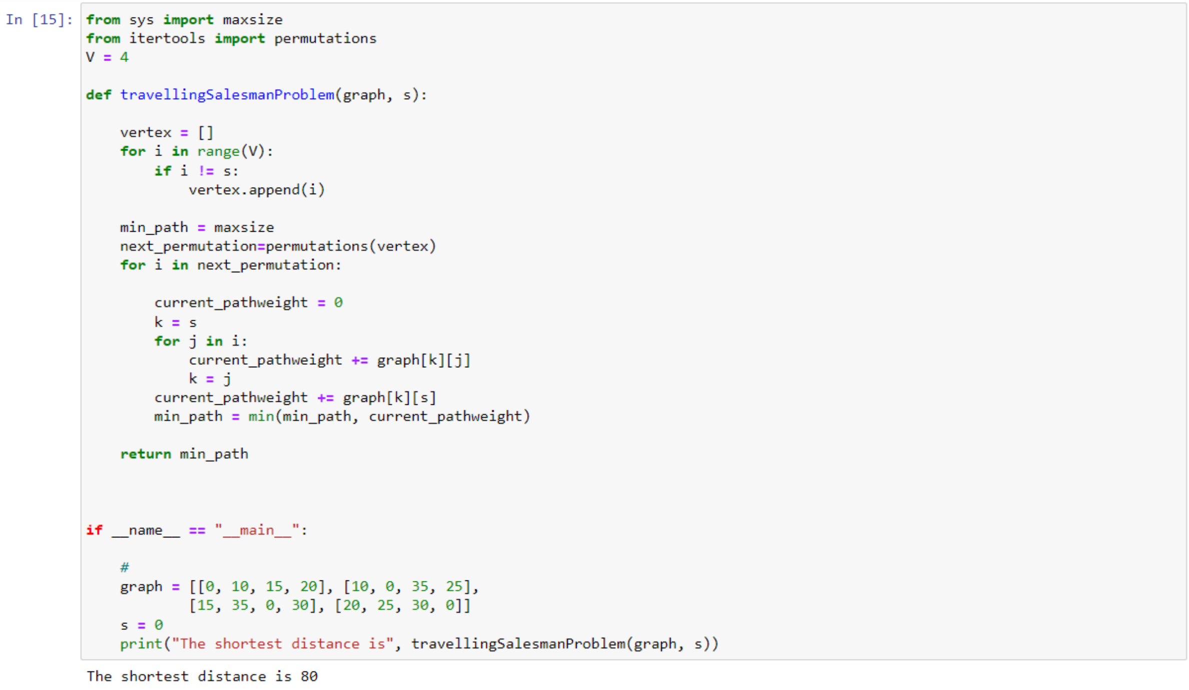
Manual Solution:

From node 1 the shortest distance is 10, so he could go to node 2. From node 2 the shortest distance is 25, so he moves to node 4. From node 4 he cannot go back to node 1 without making a stop at node 3. So, in this case the shortest possible route would be of 10+25+30+15=80m.A TSP tour in the graph is 1-2-4-3-1. The cost of the tour is 10+25+30+15 which is 80.

Program Implementation [Coding]

```
from sys import maxsize

from itertools import permutations

V = 4

def travellingSalesmanProblem(graph, s):
    vertex = []
    for i in range(V):
        if i != s:
            vertex.append(i)

    min_path = maxsize
    next_permutation=permutations(vertex)
    for i in next_permutation:
        current_pathweight = 0
        k = s
        for j in i:
            current_pathweight += graph[k][j]
            k = j
        current_pathweight += graph[k][s]
        min_path = min(min_path, current_pathweight)

    return min_path

if __name__ == "__main__":
    graph = [[0, 10, 15, 20], [10, 0, 35, 25],
            [15, 35, 0, 30], [20, 25, 30, 0]]
    s = 0
    print("The shortest distance is", travellingSalesmanProblem(graph, s))
```

Screenshots of the Outputs



```
In [15]: from sys import maxsize
         from itertools import permutations
         V = 4

         def travellingSalesmanProblem(graph, s):

             vertex = []
             for i in range(V):
                 if i != s:
                     vertex.append(i)

             min_path = maxsize
             next_permutation=permutations(vertex)
             for i in next_permutation:

                 current_pathweight = 0
                 k = s
                 for j in i:
                     current_pathweight += graph[k][j]
                     k = j
                 current_pathweight += graph[k][s]
                 min_path = min(min_path, current_pathweight)

             return min_path


         if __name__ == "__main__":

             #
             graph = [[0, 10, 15, 20], [10, 0, 35, 25],
                     [15, 35, 0, 30], [20, 25, 30, 0]]
             s = 0
             print("The shortest distance is", travellingSalesmanProblem(graph, s))

         The shortest distance is 80
```

Result: The solution for Traveling Salesman Problem is found.

Q2. Medical Diagnosis System

AIM: To implement a Medical Diagnosis System in Python

Description of the Concept or Problem given:

A medical diagnosis system is used to detect how a patient is feeling and whether he has any disease or distress in his body, after going through all the statement answers given

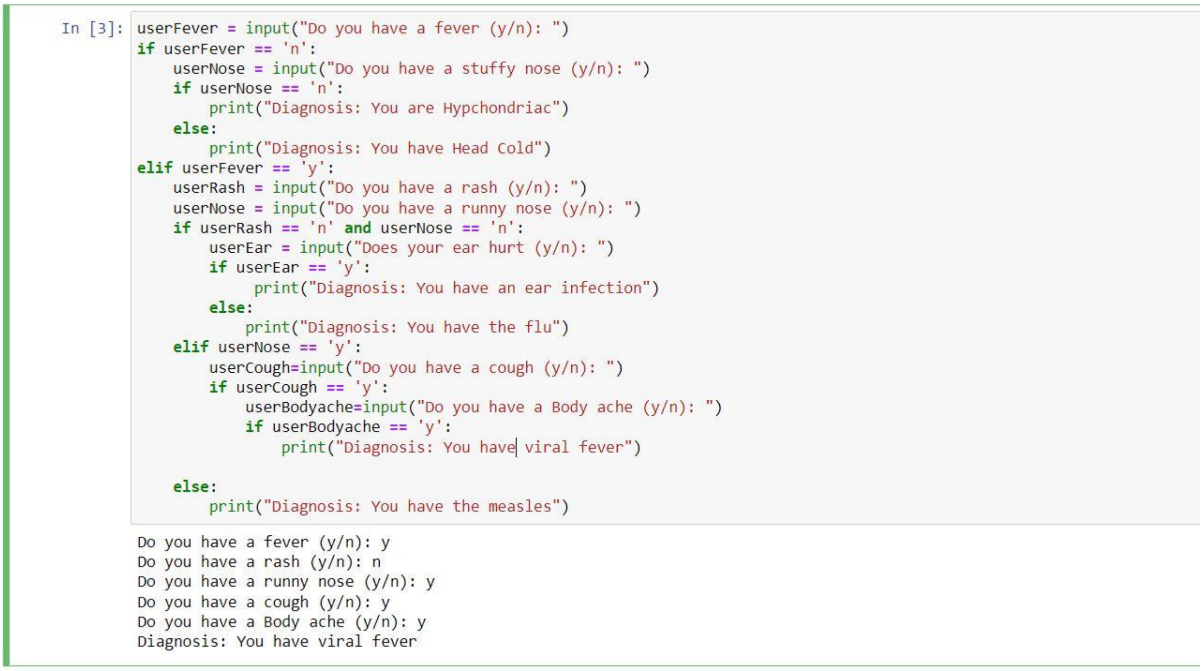
Manual Solution

The medical system asks the patient a series of questions, to which he replies in yes or no and then, the system using the if and elif and else conditions given, deduces the problem.

Program Implementation [Coding]

```
userFever = input("Do you have a fever (y/n): ")
if userFever == 'n':
    userNose = input("Do you have a stuffy nose (y/n): ")
    if userNose == 'n':
        print("Diagnosis: You are Hypchondriac")
    else:
        print("Diagnosis: You have Head Cold")
elif userFever == 'y':
    userRash = input("Do you have a rash (y/n): ")
    userNose = input("Do you have a runny nose (y/n): ")
    if userRash == 'n' and userNose == 'n':
        userEar = input("Does your ear hurt (y/n): ")
        if userEar == 'y':
             print("Diagnosis: You have an ear infection")
        else:
            print("Diagnosis: You have the flu")
    elif userNose == 'y' and userRash=='n':
        userCough=input("Do you have a cough (y/n): ")
        if userCough == 'y':
            userBodyache=input("Do you have a Body ache (y/n): ")
            if userBodyache == 'y':
                print("Diagnosis: You have viral fever")

    elif userRash == 'y'and userNose == 'n' :
        print("Diagnosis: You have the measles")
```

Screenshots of the Outputs



```
In [3]: userFever = input("Do you have a fever (y/n): ")
        if userFever == 'n':
            userNose = input("Do you have a stuffy nose (y/n): ")
            if userNose == 'n':
                print("Diagnosis: You are Hypchondriac")
            else:
                print("Diagnosis: You have Head Cold")
        elif userFever == 'y':
            userRash = input("Do you have a rash (y/n): ")
            userNose = input("Do you have a runny nose (y/n): ")
            if userRash == 'n' and userNose == 'n':
                userEar = input("Does your ear hurt (y/n): ")
                if userEar == 'y':
                     print("Diagnosis: You have an ear infection")
                else:
                    print("Diagnosis: You have the flu")
            elif userNose == 'y':
                userCough=input("Do you have a cough (y/n): ")
                if userCough == 'y':
                    userBodyache=input("Do you have a Body ache (y/n): ")
                    if userBodyache == 'y':
                        print("Diagnosis: You have viral fever")

            else:
                print("Diagnosis: You have the measles")
```

```
Do you have a fever (y/n): y
Do you have a rash (y/n): n
Do you have a runny nose (y/n): y
Do you have a cough (y/n): y
Do you have a Body ache (y/n): y
Diagnosis: You have viral fever
```

Result: The solution for the Medical Diagnosis System problem is found.

Date:27/01/22 **Ex No:3**	**Title of the Lab** **Room Colouring and Latin Square Problem**	**Name: Deeksha Rawat** **Reg Number:RA1911027010005** **Section: N1** **Lab Batch: 1** **Day Order: 2**

Q1) Room Colouring

AIM: To implement Room Coloring Problem in Python

Description of the Concept or Problem given:

Given a graph where each vertex represents a room, colour its vertices such that no two adjacent vertices have the same colour using minimum number of colours and display the result.

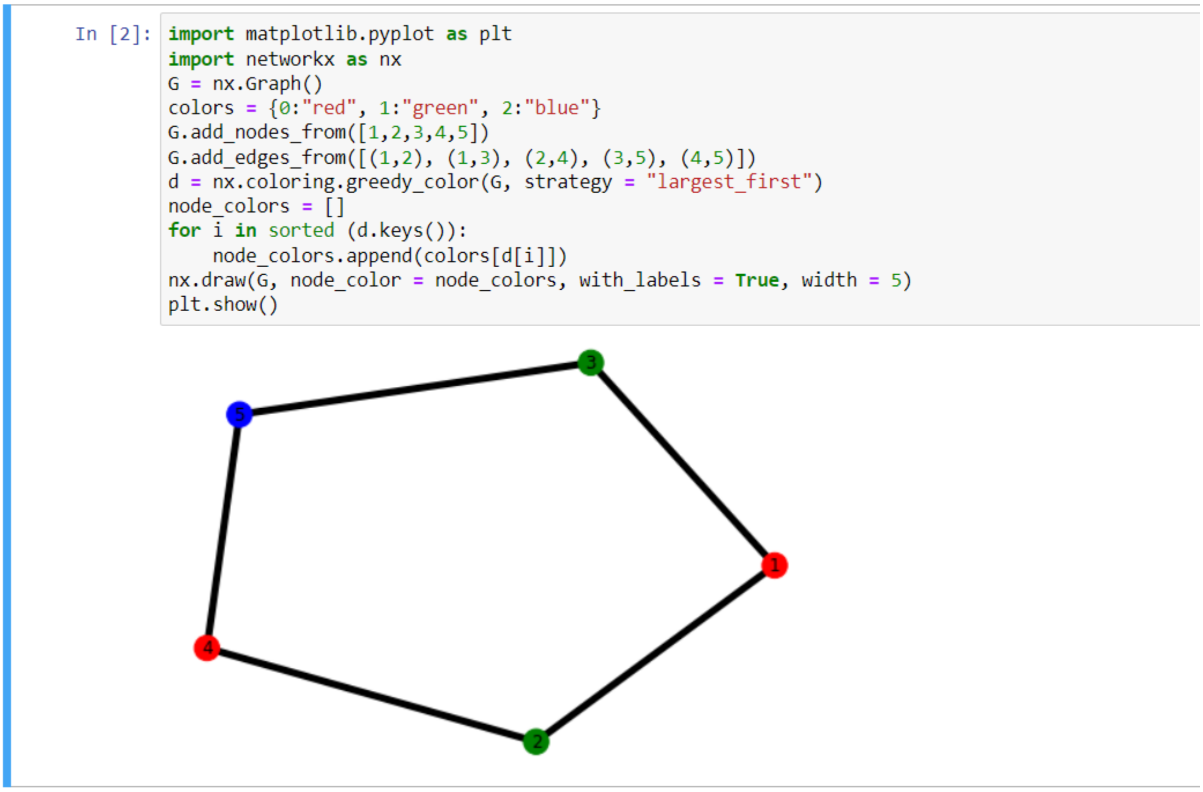
Manual Solution:

1. Color first vertex with first color.
2. Loop for remaining V-1 vertices.:
3. Consider the currently picked vertex and color it with the lowest numbered color that has not been used on any previously colored vertices adjacent to it.
4. If all previously used colors appear on vertices adjacent to v, assign a new color to it.

Program Implementation [Coding]

```
 import matplotlib.pyplot as plt
import networkx as nx
G = nx.Graph()
colors = {0:"red", 1:"green", 2:"blue"}
G.add_nodes_from([1,2,3,4,5])
G.add_edges_from([(1,2), (1,3), (2,4), (3,5), (4,5)])
d = nx.coloring.greedy_color(G, strategy = "largest_first")
node_colors = []
for i in sorted (d.keys()):
    node_colors.append(colors[d[i]])
nx.draw(G, node_color = node_colors, with_labels = True, width = 5)
plt.show()
```

Screenshots of the Outputs



```
In [2]: import matplotlib.pyplot as plt
        import networkx as nx
        G = nx.Graph()
        colors = {0:"red", 1:"green", 2:"blue"}
        G.add_nodes_from([1,2,3,4,5])
        G.add_edges_from([(1,2), (1,3), (2,4), (3,5), (4,5)])
        d = nx.coloring.greedy_color(G, strategy = "largest_first")
        node_colors = []
        for i in sorted (d.keys()):
            node_colors.append(colors[d[i]])
        nx.draw(G, node_color = node_colors, with_labels = True, width = 5)
        plt.show()
```

Result: The solution for Room Colouring Problem is found.

Q2. Latin Square Problem

AIM: To implement a Latin Square Problem in Python

Description of the Concept or Problem given:

A Latin Square is a n x n grid filled by n distinct numbers each appearing exactly once in each row and column. Given an input n, we have to print a n x n matrix consisting of numbers from 1 to n each appearing exactly once in each row and each column.

Manual Solution

1. In the first row, the numbers are stored from 1 to n serially.
2. In the second row, the numbers are shifted to the right by one column. i.e, 1 is stored at 2nd column now and so on.
3. In the third row, the numbers are shifted to the right by two columns. i.e, 1 is stored at 3rd column now and so on.
4. We continue the same way for the remaining rows.

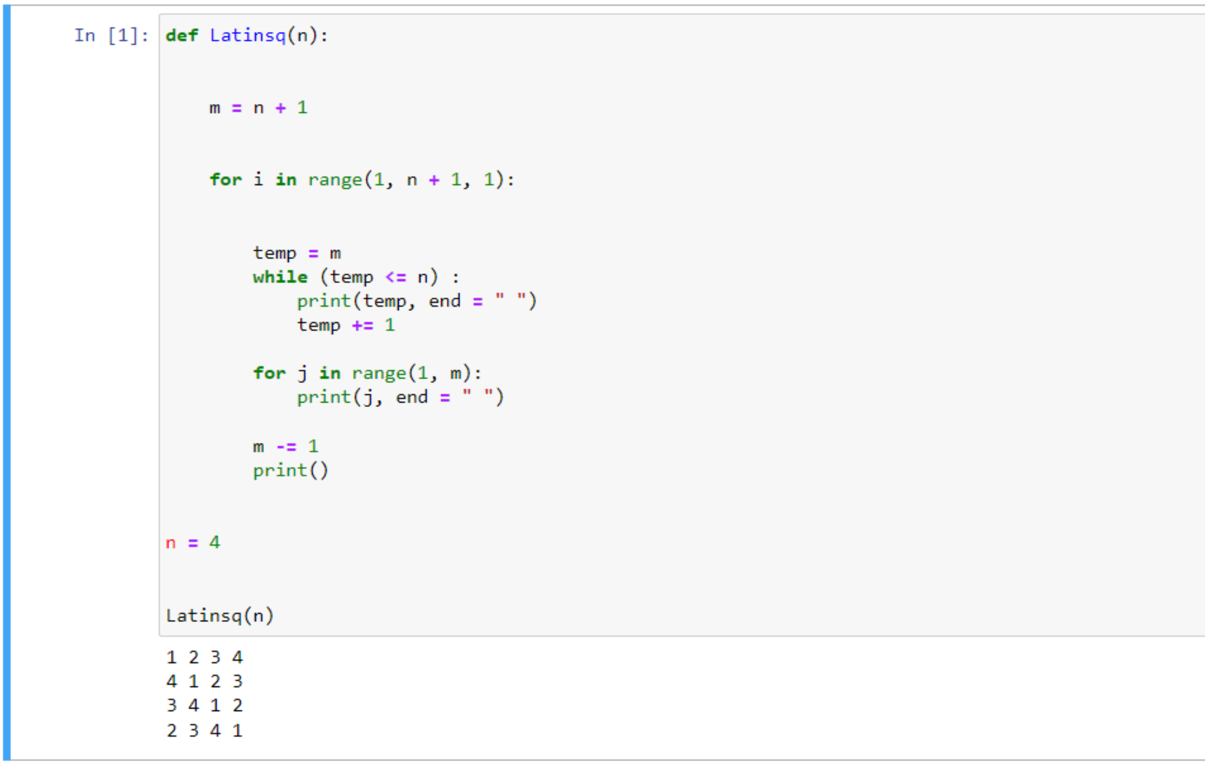
Program Implementation [Coding]

```
def Latinsq(n):
    m = n + 1
    for i in range(1, n + 1, 1):
        temp = m
        while (temp <= n) :
            print(temp, end = " ")
            temp += 1

        for j in range(1, m):
            print(j, end = " ")

        m -= 1
        print()
```

n = 4

Latinsq(n)

Screenshot of the Output



In [1]:

```
def Latinsq(n):


    m = n + 1


    for i in range(1, n + 1, 1):


        temp = m
        while (temp <= n) :
            print(temp, end = " ")
            temp += 1

        for j in range(1, m):
            print(j, end = " ")

        m -= 1
        print()


n = 4


Latinsq(n)
```

```
1 2 3 4
4 1 2 3
3 4 1 2
2 3 4 1
```

Result: The solution for the Latin Square problem is found.

Date:11/02/22 Ex No:4	Title of the Lab BFS implementation in Web crawling and DFS implementation in 6 * 6 Sudoku	Name: Deeksha Rawat Reg Number:RA1911027010005 Section: N1 Lab Batch: 1 Day Order: 2

Q4a) BFS implementation in Web crawling

AIM: To implement BFS implementation in Web crawling in Python

Description of the Concept or Problem given:

Implementation of BFS in web crawling means that we have to extract all the links from a given URL of a page at a level. These links then can be deemed as important or unimportant.

Manual Solution:

1. Breadth First Search Algorithm is the simplest form of crawling algorithm. It starts with the root node and keeps on traversing all the connected nodes to root node. In simple words it start with one link and keeps traversing all the connected links. It does not take into account the relevancy of path while traversing that's why it is also known as blind search algorithm.
2. You can find out particular objective using BFS algorithm. If objective will be found, it reports success and process will be terminated. And if it is not, it will continue traversing.
3. we perform a breadth first search (BFS) traversal considered the formation of a URL page as tree structure. At the first level we have the input URL. At the next level, we have all the URLs inside the input URL and so on.
4. We create a queue and append the input *url* into it. We then pop an *url* and insert all the *url*s inside it into the queue. We do this until all the *url*s at a particular level is not parsed. We repeat the process for the number of times same as the input depth.

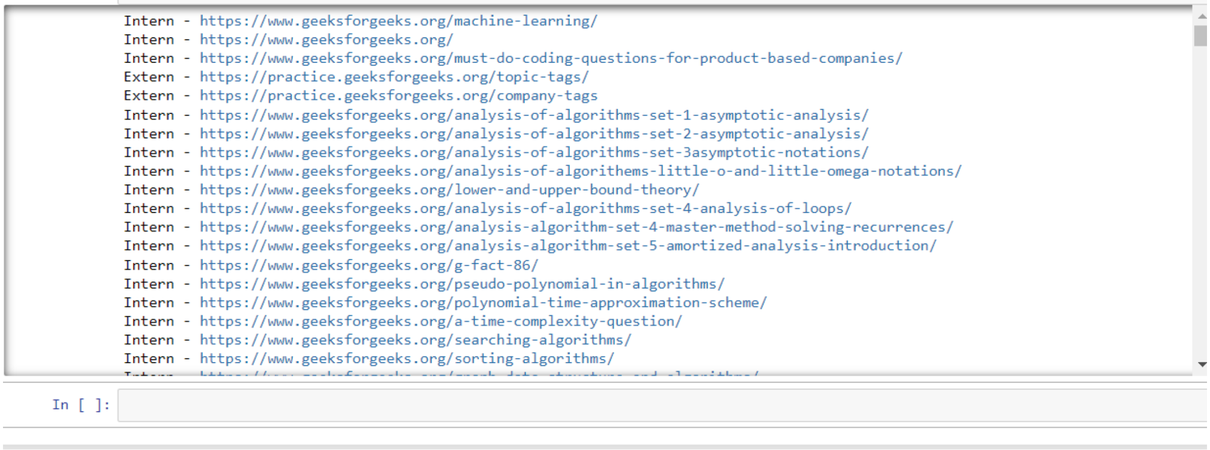
Program Implementation [Coding]

```
from urllib.request import urljoin

from bs4 import BeautifulSoup

import requests

from urllib.request import urlparse
```

```
links_intern = set()
input_url = "https://www.geeksforgeeks.org/machine-learning/"
depth = 1

# Set for storing urls with different domain
links_extern = set()

def level_crawler(input_url):
    temp_urls = set()
    current_url_domain = urlparse(input_url).netloc
    beautiful_soup_object = BeautifulSoup(
        requests.get(input_url).content, "lxml")
    for anchor in beautiful_soup_object.findAll("a"):
        href = anchor.attrs.get("href")
        if(href != "" or href != None):
            href = urljoin(input_url, href)
            href_parsed = urlparse(href)
            href = href_parsed.scheme
            href += "://"
            href += href_parsed.netloc
            href += href_parsed.path
            final_parsed_href = urlparse(href)
            is_valid = bool(final_parsed_href.scheme) and bool(
                final_parsed_href.netloc)
            if is_valid:
                if current_url_domain not in href and href not in links_extern:
                    print("Extern - {}".format(href))
                    links_extern.add(href)
                if current_url_domain in href and href not in links_intern:
                    print("Intern - {}".format(href))
                    links_intern.add(href)
                    temp_urls.add(href)
    return temp_urls
```

```
if(depth == 0):
    print("Intern - {}".format(input_url))


elif(depth == 1):
    level_crawler(input_url)


else:


    queue = []
    queue.append(input_url)
    for j in range(depth):
        for count in range(len(queue)):
            url = queue.pop(0)
            urls = level_crawler(url)
            for i in urls:
                queue.append(i)
```

Screenshots of the Outputs



```
Intern - https://www.geeksforgeeks.org/machine-learning/
Intern - https://www.geeksforgeeks.org/
Intern - https://www.geeksforgeeks.org/must-do-coding-questions-for-product-based-companies/
Extern - https://practice.geeksforgeeks.org/topic-tags/
Extern - https://practice.geeksforgeeks.org/company-tags
Intern - https://www.geeksforgeeks.org/analysis-of-algorithms-set-1-asymptotic-analysis/
Intern - https://www.geeksforgeeks.org/analysis-of-algorithms-set-2-asymptotic-analysis/
Intern - https://www.geeksforgeeks.org/analysis-of-algorithms-set-3asymptotic-notations/
Intern - https://www.geeksforgeeks.org/analysis-of-algorithems-little-o-and-little-omega-notations/
Intern - https://www.geeksforgeeks.org/lower-and-upper-bound-theory/
Intern - https://www.geeksforgeeks.org/analysis-of-algorithms-set-4-analysis-of-loops/
Intern - https://www.geeksforgeeks.org/analysis-algorithm-set-4-master-method-solving-recurrences/
Intern - https://www.geeksforgeeks.org/analysis-algorithm-set-5-amortized-analysis-introduction/
Intern - https://www.geeksforgeeks.org/g-fact-86/
Intern - https://www.geeksforgeeks.org/pseudo-polynomial-in-algorithms/
Intern - https://www.geeksforgeeks.org/polynomial-time-approximation-scheme/
Intern - https://www.geeksforgeeks.org/a-time-complexity-question/
Intern - https://www.geeksforgeeks.org/searching-algorithms/
Intern - https://www.geeksforgeeks.org/sorting-algorithms/
Intern - https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/
```

In [ ]:

Result: The solution for BFS implementation in Web crawling Problem is found.

Q4b). DFS implementation in 6 * 6 Sudoku

AIM: To implement a DFS implementation in 6 * 6 Sudoku Problem in Python

Description of the Concept or Problem given:

Given a partially filled 6×6 2D array ‘grid[6][6]’, the goal is to assign digits (from 1 to 6) to the empty cells so that every row, column, and subgrid of size 2×3 contains exactly one instance of the digits from 1 to 6.

Manual Solution

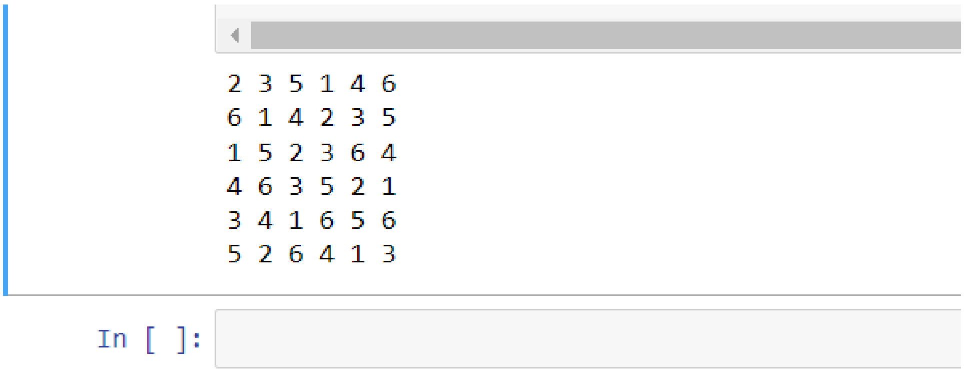
1. Sudoku can be solved by one by one assigning numbers to empty cells. Before assigning a number, check whether it is safe to assign. Check that the same number is not present in the current row, current column and current 2X3 subgrid.
2. After checking for safety, assign the number, and recursively check whether this assignment leads to a solution or not. If the assignment doesn’t lead to a solution, then try the next number for the current empty cell. And if none of the number (1 to 6) leads to a solution, return false and print no solution exists.

Program Implementation [Coding]

```
def print_grid(arr):
    for i in range(6):
        for j in range(6):
            print(str(arr[i][j])+" ",end="")
        print ('')


def empty_location(arr, l):
    for row in range(6):
        for col in range(6):
            if(arr[row][col]== 0):
                l[0]= row
                l[1]= col
                return True
    return False


def used_in_row(arr, row, num):
    for i in range(6):
        if(arr[row][i] == num):
            return True
    return False


def used_in_col(arr, col, num):
    for i in range(6):
        if(arr[i][col] == num):
            return True
    return False


def used_in_box(arr, row, col, num):
    for i in range(2):
        for j in range(3):
            if(arr[i + row][j + col] == num):
                return True
    return False



def check_location(arr, row, col, num):
    return not used_in_row(arr, row, num) and not used_in_col(arr, col, num) and
not used_in_box(arr, row - row % 2,col - col % 3, num)
```

```
def solve_sudoku(arr):
    l =[0, 0]
    if(not empty_location(arr, l)):
        return True
        row = l[0]
    col = l[1]
    for num in range(1, 7):
        if(check_location(arr,
                          row, col, num)):
            arr[row][col]= num

            if(solve_sudoku(arr)):
                return True

            arr[row][col] = 0



    return False

if __name__=="__main__":



    grid =[[0 for x in range(6)]for y in range(6)]
grid =[[0,0,0,1,0,6],
        [6,0,4,0,0,0],
        [1,0,2,0,0,0],
        [0,0,0,5,0,1],
        [0,0,0,6,0,6],
        [5,0,6,0,0,0]]

  if(solve_sudoku(grid)):
     print_grid(grid)
  else:
     print ("No solution exists")
```

Screenshot of the Output



```
2 3 5 1 4 6
6 1 4 2 3 5
1 5 2 3 6 4
4 6 3 5 2 1
3 4 1 6 5 6
5 2 6 4 1 3
```

Result: The solution for the DFS implementation in 6 * 6 problem is found.

Date: 18-02-22 **Ex No:** 5	Implementation of Best First Search and A* - Informed Search Algorithms	**Name:** Deeksha Rawat **Registration Number:** RA1911027010005 **Section:** N1 **Lab Batch:** 1 **Day Order:** 2

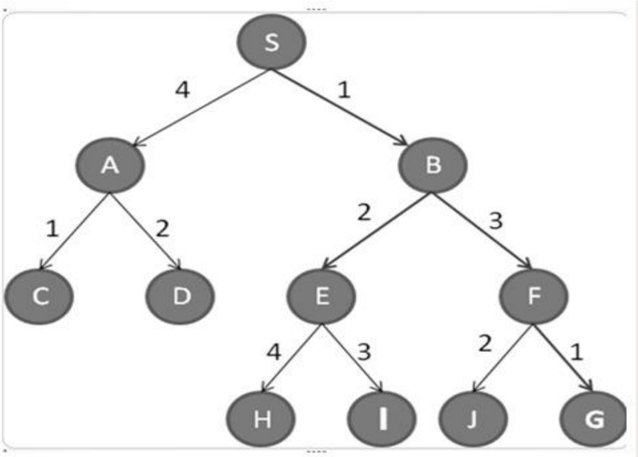
5a)

AIM: To implement Best First search algorithm to find shortest path

Description of the Concept or Problem given:

Implementation of Best First Search specifies that we have to find the shortest path to find the target node in a provided graph

Manual Solution:

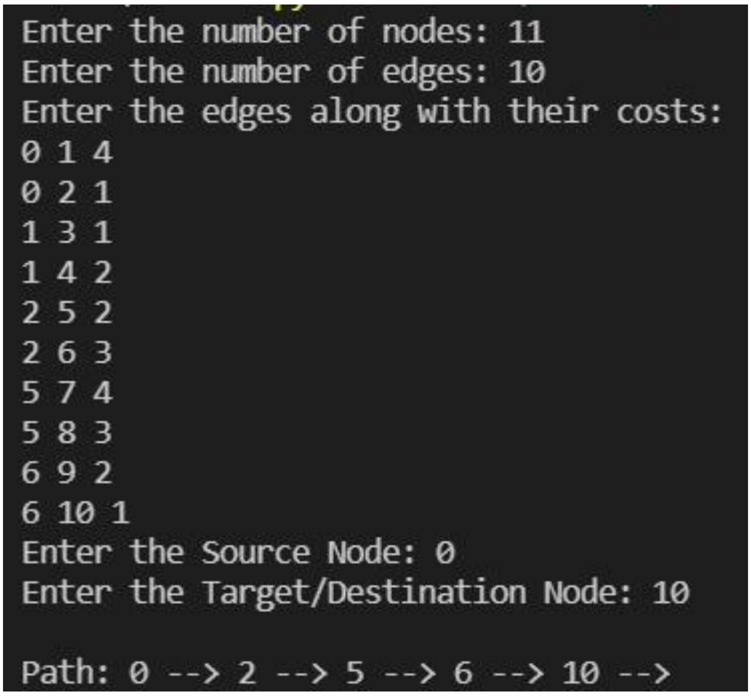


1. Consider the graph, for example we take the source node as S and the target or destination node as G. Best first search uses a priority queue to store the cost of the nodes.
2. It will start checking all the neighbours of S and start with the neighbour which has the least cost.
3. Then itwill keep on adding the neighbours of the visited nodes and remove the visited nodes from the priority queue.
4. If the target node is one of the neighbours of a visited node, it returns the path otherwise processes continue.

Program Implementation:

```
from queue import PriorityQueue
import networkx as nx


def best_first_search(source, target, n):
    visited = [0] * n
    visited[source] = 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))

G = nx.Graph()
v = int(input("Enter the number of nodes: "))
graph = [[] for i in range(v)]
e = int(input("Enter the number of edges: "))
print("Enter the edges along with their costs:")
for i in range(e):
    x, y, z = list(map(int, input().split()))
    addedge(x, y, z)
    G.add_edge(x, y, weight = z)

source = int(input("Enter the Source Node: "))
target = int(input("Enter the Target/Destination Node: "))
print("\nPath: ", end = "")
best_first_search(source, target, v)
```

Screenshots of the Outputs:



```
Enter the number of nodes: 11
Enter the number of edges: 10
Enter the edges along with their costs:
0 1 4
0 2 1
1 3 1
1 4 2
2 5 2
2 6 3
5 7 4
5 8 3
6 9 2
6 10 1
Enter the Source Node: 0
Enter the Target/Destination Node: 10

Path: 0 --> 2 --> 5 --> 6 --> 10 -->
```

Result: The solution for Best First search algorithm to find shortest path found

5b)

AIM: To implement A* algorithm to find the solution of the 8-puzzle problem

Description of the Concept or Problem given: 8-puzzle problem that consists of N tiles where N can 8, 15, 24 and so on. In 8-puzzle, the grid is divided into 3 rows and 3 columns and contains 8 spaces fully filled with integers from 1 to 8 and one empty space where the number tiles can be moved. Start and goal states of the puzzle are also provided. The puzzle can be solved by moving the integer tiles one by one in the empty space and henceforth achieving the goal configuration.

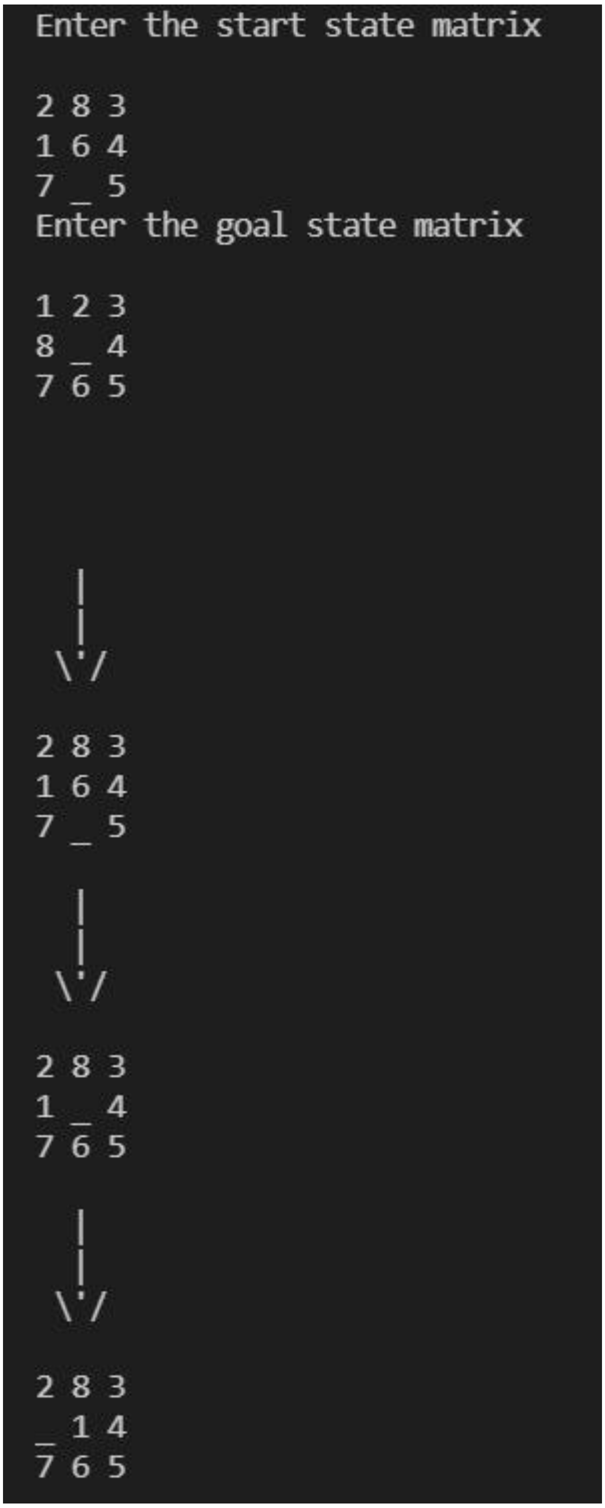
Manual Solution:

1. A heuristic algorithm sacrifices optimality, with precision and accuracy for speed, to solve problems faster and more efficiently.
2. 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 paths transverse gives you the cost of that route.
3. Initially, the Algorithm calculates the cost to all its immediate neighboring 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 :
4. 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

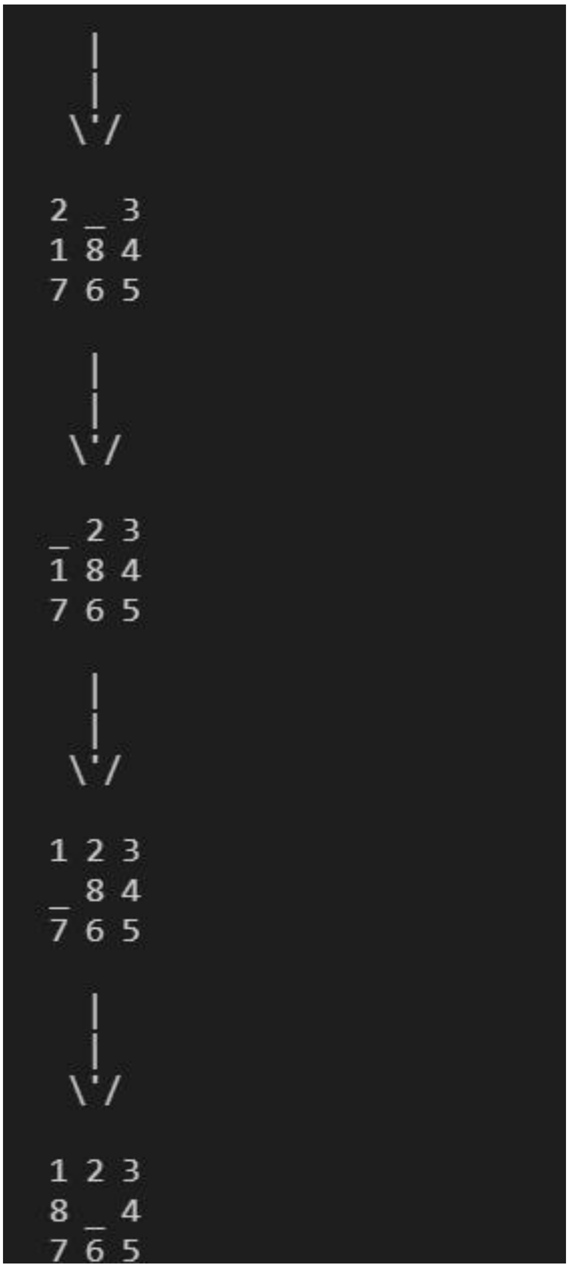
Program Implementation:

```
class Node:
    def _init_(self,data,level,fval):
        self.data = data
        self.level = level
        self.fval = fval

    def generate_child(self):
        x,y = self.find(self.data,'_')
        val_list = [[x,y-1],[x,y+1],[x-1,y],[x+1,y]]
        children = []
        for i in val_list:
            child = self.shuffle(self.data,x,y,i[0],i[1])
            if child is not None:
                child_node = Node(child,self.level+1,0)
                children.append(child_node)
        return children

    def shuffle(self,puz,x1,y1,x2,y2):

        if x2 >= 0 and x2 < len(self.data) and y2 >= 0 and y2 < len(self.data):
            temp_puz = []
            temp_puz = self.copy(puz)
            temp = temp_puz[x2][y2]
            temp_puz[x2][y2] = temp_puz[x1][y1]
            temp_puz[x1][y1] = temp
            return temp_puz
        else:
            return None


    def copy(self,root):
        temp = []
        for i in root:
            t = []
            for j in i:
                t.append(j)
            temp.append(t)
        return temp

    def find(self,puz,x):
        for i in range(0,len(self.data)):
            for j in range(0,len(self.data)):
                if puz[i][j] == x:
                    return i,j


class Puzzle:
    def _init_(self,size):
        self.n = size
        self.open = []
        self.closed = []

    def accept(self):
        puz = []
        for i in range(0,self.n):
            temp = input().split(" ")
            puz.append(temp)
        return puz

    def f(self,start,goal):
        return self.h(start.data,goal)+start.level

    def h(self,start,goal):
        temp = 0
        for i in range(0,self.n):
            for j in range(0,self.n):
                if start[i][j] != goal[i][j] and start[i][j] != '_':
                    temp += 1
        return temp


    def process(self):
        print("Enter the start state matrix \n")
        start = self.accept()
        print("Enter the goal state matrix \n")
        goal = self.accept()

        start = Node(start,0,0)
        start.fval = self.f(start,goal)

        self.open.append(start)
        print("\n\n")
        while True:
            cur = self.open[0]
            print("")
            print("  | ")
            print("  | ")
            print(" \\\'/ \n")
            for i in cur.data:
                for j in i:
                    print(j,end=" ")
                print("")

            if(self.h(cur.data,goal) == 0):
                break
            for i in cur.generate_child():
                i.fval = self.f(i,goal)
                self.open.append(i)
            self.closed.append(cur)
            del self.open[0]

            """ sort the opne list based on f value """
            self.open.sort(key = lambda x:x.fval,reverse=False)


puz = Puzzle(3)
puz.process()
```

Screenshots of the Outputs:



```
Enter the start state matrix

2 8 3
1 6 4
7 _ 5
Enter the goal state matrix

1 2 3
8 _ 4
7 6 5





   |
   |
  \'/

2 8 3
1 6 4
7 _ 5

   |
   |
  \'/

2 8 3
1 _ 4
7 6 5

   |
   |
  \'/

2 8 3
_ 1 4
7 6 5
```



```
     |
     |
    \'/

   2 _ 3
   1 8 4
   7 6 5

     |
     |
    \'/

   _ 2 3
   1 8 4
   7 6 5

     |
     |
    \'/

   1 2 3
   _ 8 4
   7 6 5

     |
     |
    \'/

   1 2 3
   8 _ 4
   7 6 5
```

Result: The solution for A* algorithm to find the solution of the 8-puzzle problem is found

Date: 31-03-22 **Ex No: 6**	**Implementation of Fuzzy Logic**	**Name:** Deeksha Rawat **Registration Number:** RA1911027010005 **Section:** N1 **Lab Batch:** 1 **Day Order:** 2

AIM: To implement Fuzzy Logic

Description of the Concept or Problem given: We have to implement to demonstrate the working of Fuzzy Logic

Manual Solution:

• Define Non Fuzzy Inputs with Fuzzy Sets. The non-fuzzy inputs are numbers from a certain range, and find

• how to represent those non-fuzzy values with fuzzy sets.

• Locate the input, output, and state variables of the plane under consideration.

• 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.

• Obtain the membership function for each fuzzy subset.

• Assign the fuzzy relationships between the inputs or states of fuzzy subsets on one side and output of fuzzy

• subsets on the other side, thereby forming the rule base.

• Carry out the fuzzification process.

• Identify the output contributed from each rule using fuzzy approximate reasoning.

• Combine the fuzzy outputs obtained from each rule.

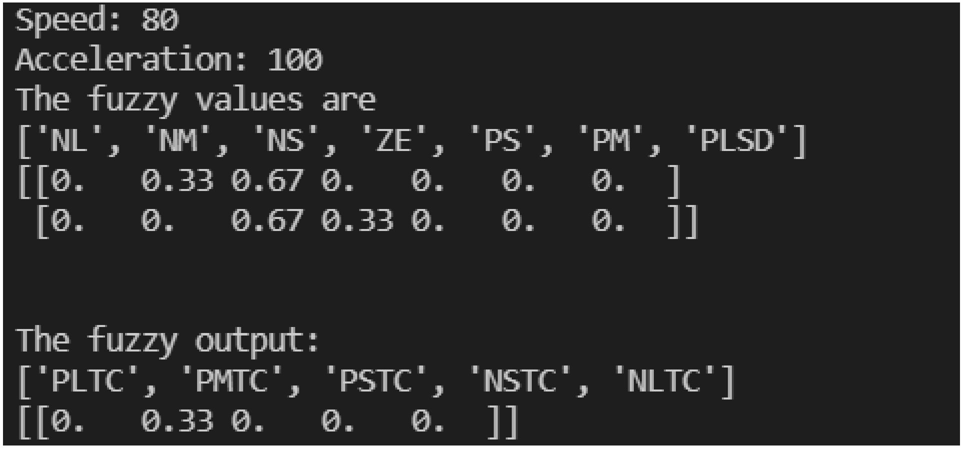
• Finally, apply defuzzification to form a crisp output.

Program Implementation [Coding]:

```
import numpy as np

Speed = int(input('Speed: '))
Acceleration = int(input('Acceleration: '))

def openLeft(x,alpha, beta):
    if x<alpha:
        return 1
    if alpha<x and x<=beta:
        return (beta - x)/(beta - alpha)
    else:
        return 0

def openRight(x,alpha, beta):
    if x<alpha:
        return 0
    if alpha<x and x<=beta:
        return (x-alpha)/(beta - alpha)
    else:
        return 0

def triangular(x,a,b,c):
    return max(min((x-a)/(b-a), (c-x)/(c-b)),0)

def partition(x):
    NL = 0;  NM = 0; NS = 0; ZE = 0; PS = 0; PM = 0; PL = 0

    if x> 0 and x<60:
        NL = openLeft(x,30,60)
    if x> 30 and x<90:
        NM = triangular(x,30,60,90)
    if x> 60 and x<120:
        NS = triangular(x,60,90,120)
    if x> 90 and x<150:
        ZE = triangular(x,90,120,150)
    if x> 120 and x<180:
        PS = triangular(x,120,150,180)
    if x> 150 and x<210:
        PM = triangular(x,120,150,180)
    if x> 180 and x<240:
        PL = openRight(x,180,210)

    return NL,NM,NS,ZE,PS,PM,PL

NLSD,NMSD,NSSD,ZESD,PSSD,PMSD,PLSD = partition(Speed)
NLAC,NMAC,NSAC,ZEAC,PSAC,PMAC,PLAC = partition(Acceleration)

outPut = [[NLSD,NMSD,NSSD,ZESD,PSSD,PMSD,PLSD],
          [NLAC,NMAC,NSAC,ZEAC,PSAC,PMAC,PLAC]]
print("The fuzzy values are")
print(["NL","NM","NS","ZE","PS","PM","PLSD"])
print(np.round(outPut,2))

# Rules implementation
def compare(TC1, TC2):
    TC = 0
    if TC1>TC2 and TC1 !=0 and TC2 !=0:
        TC = TC2
    else:
        TC = TC1

    if TC1 == 0 and TC2 !=0:
        TC = TC2
    if TC2 == 0 and TC1 !=0:
        TC = TC1


    return TC



def
rule(NLSD,NMSD,NSSD,ZESD,PSSD,PMSD,PLSD,NLAC,NMAC,NSAC,ZEAC,PSAC,PMAC,PL
AC):
    PLTC1 = min(NLSD,ZEAC)
    PLTC2 = min(ZESD,NLAC)
    PLTC = compare(PLTC1, PLTC2)


    PMTC1 = min(NMSD,ZEAC)
    PMTC2 = min(ZESD,NMAC)
    PMTC = compare(PMTC1, PMTC2)


    PSTC1 = min(NSSD,PSAC)
    PSTC2 = min(ZESD,NSAC)
    PSTC = compare(PSTC1, PSTC2)
    NSTC = min(PSSD,NSAC)
    NLTC = min(PLSD,ZEAC)


    return PLTC, PMTC, PSTC, NSTC, NLTC


PLTC, PMTC, PSTC, NSTC, NLTC =
rule(NLSD,NMSD,NSSD,ZESD,PSSD,PMSD,PLSD,NLAC,NMAC,NSAC,ZEAC,PSAC,PMAC,PL
AC)


print("\n")


outPutRules = [[PLTC, PMTC, PSTC, NSTC, NLTC ]]
print("The fuzzy output: ")
print(["PLTC", "PMTC", "PSTC", "NSTC", "NLTC"])
print(np.round(outPutRules,2))
```

```
def areaTR(mu, a,b,c):
    x1 = mu*(b-a) + a
    x2 = c - mu*(c-b)
    d1 = (c-a); d2 = x2-x1
    a = (1/2)*mu*(d1 + d2)
    return a

def areaOL(mu, alpha, beta):
    xOL = beta -mu*(beta - alpha)
    return 1/2*mu*(beta+ xOL), beta/2

def areaOR(mu, alpha, beta):
    xOR = (beta - alpha)*mu + alpha
    aOR = (1/2)*mu*((240 - alpha) + (240 -xOR))
    return aOR, (240 - alpha)/2 + alpha

def defuzzyfication(PLTC, PMTC, PSTC, NSTC, NLTC):
    areaPL = 0; areaPM = 0; areaPS = 0; areaNS = 0; areaNL = 0
    cPL = 0; cPM = 0; cPS = 0; cNS = 0; cNL = 0

    if PLTC != 0:

        areaPL, cPL = areaOR(PLTC, 180, 210)

    if PMTC != 0:
        areaPM = areaTR(PMTC, 150, 180, 210)
        cPM = 180

    if PSTC != 0:
        areaPS = areaTR(PSTC, 120, 150, 180)
        cPS = 150

    if NSTC != 0:
        areaNS = areaTR(NSTC, 60, 90, 120)
        cNS = 90

    if NLTC !=0:
        areaNL, cNL = areaOL(NLTC, 30, 60)

    numerator = areaPL*cPL + areaPM*cPM + areaPS*cPS + areaNS*cNS + areaNL*cNL
    denominator = areaPL + areaPM + areaPS + areaNS + areaNL
    if denominator ==0:
        print("Rule does not exist")
        return 0
    else:
        crispOutput = numerator/denominator
        return crispOutput
```

Screenshots of the Outputs:



```
Speed: 80
Acceleration: 100
The fuzzy values are
['NL', 'NM', 'NS', 'ZE', 'PS', 'PM', 'PLSD']
[[0.   0.33 0.67 0.   0.   0.   0.  ]
 [0.   0.   0.67 0.33 0.   0.   0.  ]]


The fuzzy output:
['PLTC', 'PMTC', 'PSTC', 'NSTC', 'NLTC']
[[0.   0.33 0.   0.   0.  ]]
```

Signature of the Student

Deeksha Rawat

Date: 31-03-22 **Ex No: 7**	**Unification and Resolution in Prolog**	**Name:** Deeksha Rawat **Registration Number:** RA1911027010005 **Section:** N1 **Lab Batch:** 1 **Day Order:** 2

AIM: To demonstrate unification and resolution using Prolog

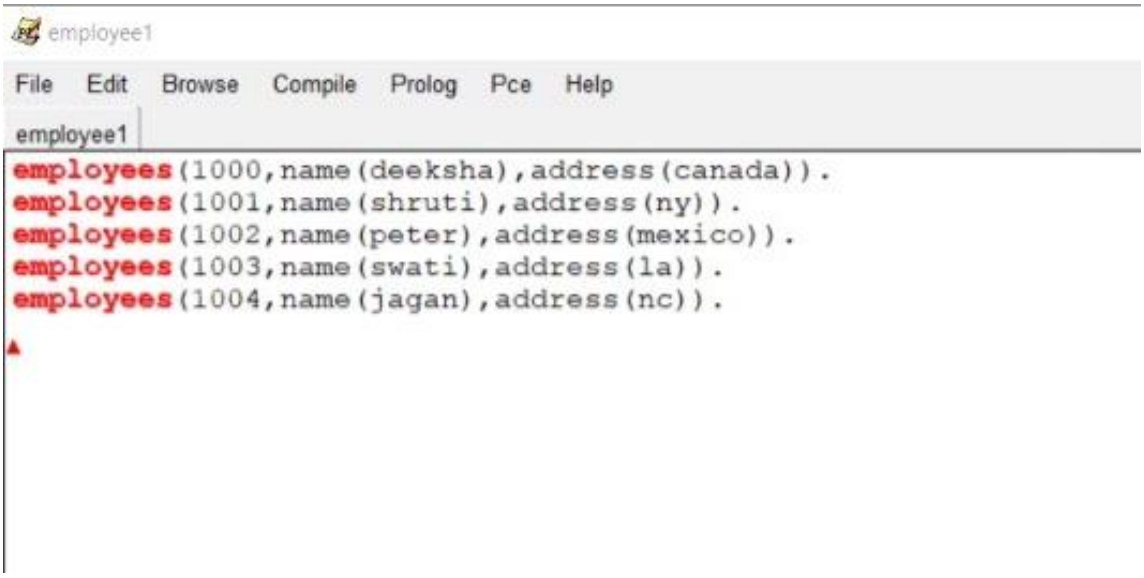
Description of the Concept or Problem given: For Unification, we combine two different logical atomic expressions using a substitution

For Resolution, we provide proofs for theorems by building refutation proofs and obtain a conclusion of the statements

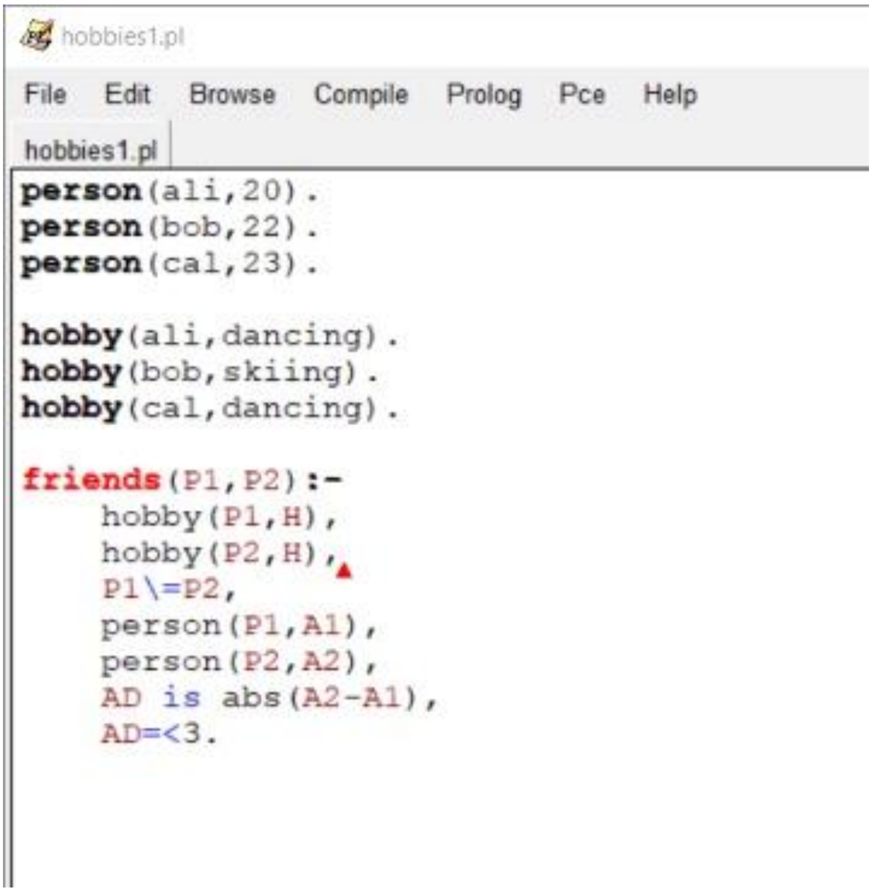
Manual Solution: For both Unification and Resolution, we implement two separate different Knowledgebase. To demonstrate Unification, we make it with Employee details and We write queries to get results

For Resolution, we declare a Knowledgebase to about people with their ages and hobbies separately. Here we also implement a program to check certain conditions and if they are proven by resolution, the program is successfully implemented.

Program Implementation [Coding/Knowledgebase]:

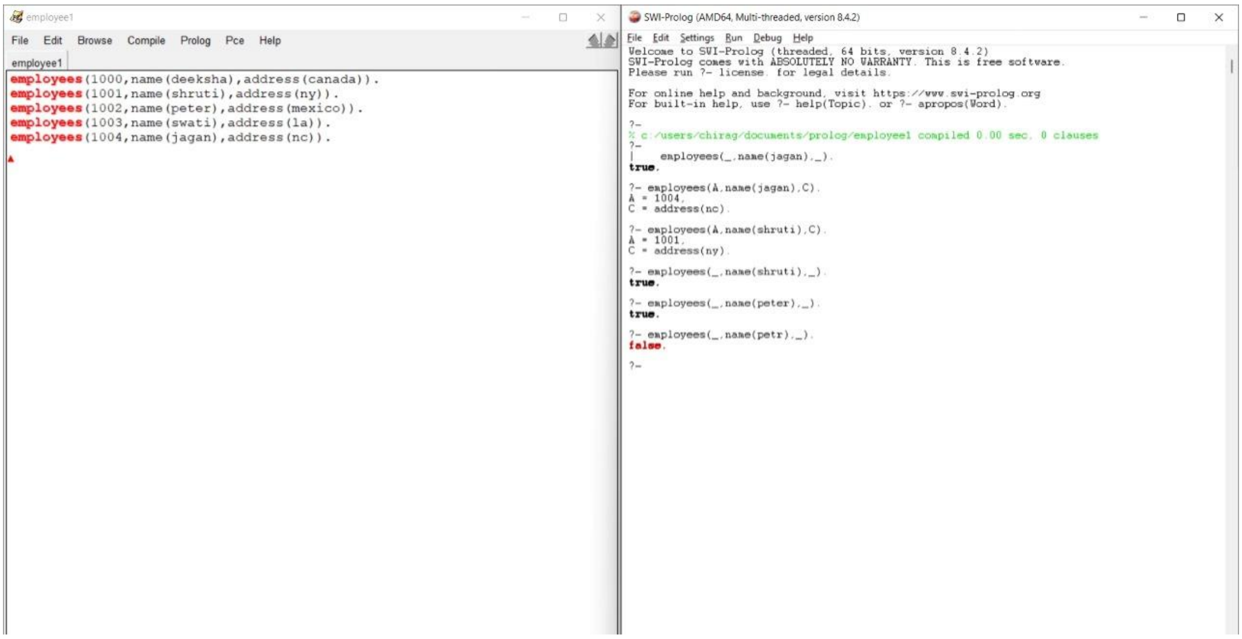


```
employees(1000,name(deeksha),address(canada)).
employees(1001,name(shruti),address(ny)).
employees(1002,name(peter),address(mexico)).
employees(1003,name(swati),address(la)).
employees(1004,name(jagan),address(nc)).
```



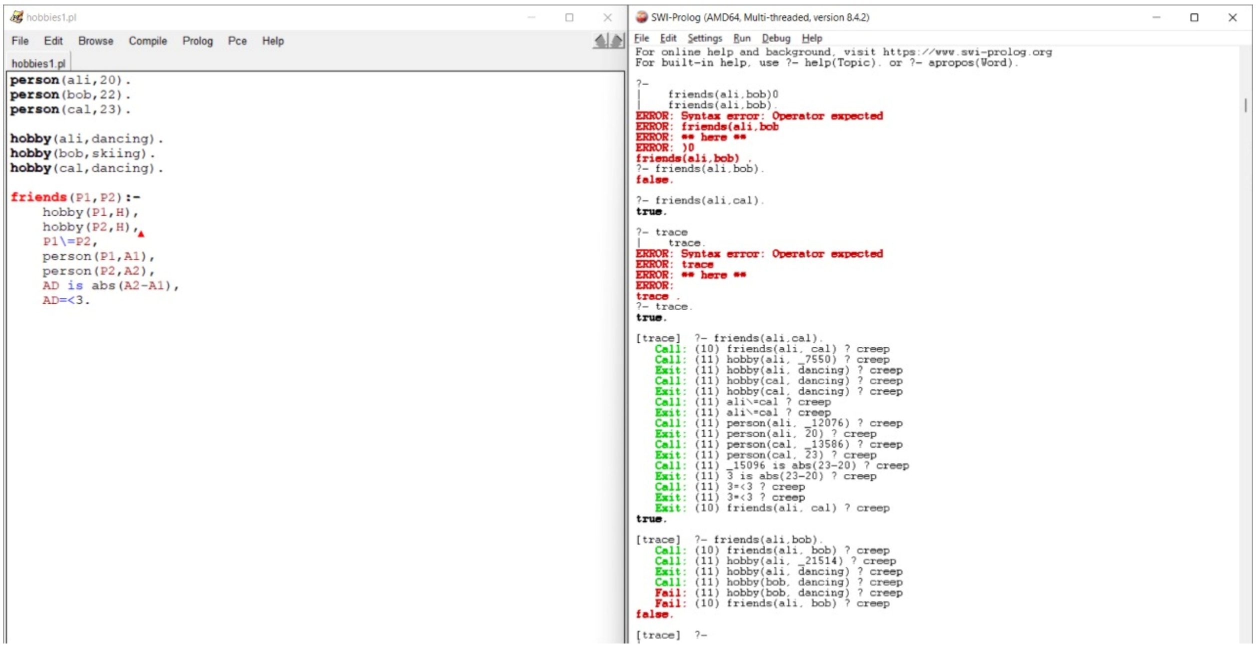
```
person(ali,20).
person(bob,22).
person(cal,23).

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

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

Screenshots of the Outputs:


employee1
File Edit Browse Compile Prolog Pce Help
employee1
SWI-Prolog (AMD64, Multi-threaded, version 8.4.2)
File Edit Settings Run Debug Help

```
employees(1000,name(deeksha),address(canada)).
employees(1001,name(shruti),address(ny)).
employees(1002,name(peter),address(mexico)).
employees(1003,name(swati),address(la)).
employees(1004,name(jagan),address(nc)).
```

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

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

?-
% c:/users/chirag/documents/prolog/employee1 compiled 0.00 sec, 0 clauses
?-
|    employees(_,name(jagan),_).
true.

?- employees(A,name(jagan),C).
A = 1004,
C = address(nc).

?- employees(A,name(shruti),C).
A = 1001,
C = address(ny).

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

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

?- employees(_,name(petr),_).
false.

?-
```


hobbies1.pl
File Edit Browse Compile Prolog Pce Help
hobbies1.pl
SWI-Prolog (AMD64, Multi-threaded, version 8.4.2)
File Edit Settings Run Debug Help

```
person(ali,20).
person(bob,22).
person(cal,23).

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

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

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

?-
|    friends(ali,bob)0
|    friends(ali,bob).
ERROR: Syntax error: Operator expected
ERROR: friends(ali,bob
ERROR: ** here **
ERROR: )0
friends(ali,bob) .
?- friends(ali,bob).
false.

?- friends(ali,cal).
true.

?- trace
|    trace.
ERROR: Syntax error: Operator expected
ERROR: trace
ERROR: ** here **
ERROR:
trace .
?- trace.
true.

[trace]  ?- friends(ali,cal).
   Call: (10) friends(ali, cal) ? creep
   Call: (11) hobby(ali, _7550) ? creep
   Exit: (11) hobby(ali, dancing) ? creep
   Call: (11) hobby(cal, dancing) ? creep
   Exit: (11) hobby(cal, dancing) ? creep
   Call: (11) ali\=cal ? creep
   Exit: (11) ali\=cal ? creep
   Call: (11) person(ali, _12076) ? creep
   Exit: (11) person(ali, 20) ? creep
   Call: (11) person(cal, _13586) ? creep
   Exit: (11) person(cal, 23) ? creep
   Call: (11) _15096 is abs(23-20) ? creep
   Exit: (11) 3 is abs(23-20) ? creep
   Call: (11) 3=<3 ? creep
   Exit: (11) 3=<3 ? creep
   Exit: (10) friends(ali, cal) ? creep
true.

[trace]  ?- friends(ali,bob).
   Call: (10) friends(ali, bob) ? creep
   Call: (11) hobby(ali, _21514) ? creep
   Exit: (11) hobby(ali, dancing) ? creep
   Call: (11) hobby(bob, dancing) ? creep
   Fail: (11) hobby(bob, dancing) ? creep
   Fail: (10) friends(ali, bob) ? creep
false.

[trace]  ?-
```

Signature of the Student

Deeksha Rawat

Date: 04-04-2022 **Ex No: 8**	**Implementation of Machine Learning Algorithms For An Application**	**Name: Deeksha Rawat** **Registration Number: RA1911027010005** **Section: N1** **Lab Batch: 1** **Day Order: 2**

1. Facial Recognition using PCA and SVM

AIM: To implement PCA (unsupervised) and SVM (supervised) learning algorithms on an open source dataset for image recognition and classification.

Description of the Concept or Problem:

PCA, or Principal Component Analysis, is an unsupervised machine learning algorithm which is used in exploratory data analysis to make predictive models. It is most commonly used to incorporate dimensionality reduction by projecting each data point onto only the first few principal components to obtain lower-dimensional data while preserving as much of the data's variation as possible. PCA also assists in feature extraction by combining input variables in a specific way in order to "drop" the least important variables and retain the most important ones.

SVM, or Support Vector Machine, is a supervised machine learning algorithm which is an extension of the support vector classifier that results from enlarging the feature space using kernels. It is mainly used for two-group classification problems. An SVM model can recognize new test data when it is fed a set of labelled training data for each category. The SVM algorithm is a really good algorithm for image classification as SVMs achieve significantly higher search accuracy than traditional query refinement schemes after just three to four rounds of relevance feedback. SVMs are different from other classification algorithms because of the way they choose the decision boundary that maximizes the distance from the nearest data points of all the classes. The decision boundary created by SVMs is called the maximum margin classifier or the maximum margin hyper plane.

Manual Solution:

i. PCA

- We calculate a matrix that summarizes how our variables all relate to one another.
- We then break this matrix down into two separate components: direction and magnitude. We can then understand the "directions" of our data and its "magnitude", or how "important" each direction is.
- We will transform our original data to align with these important directions and find the line of best fit.

- By identifying which “directions” are most important, we compress our data into a smaller space by dropping the “directions” that are the least important, thus reducing the dimensionality.

ii. SVM

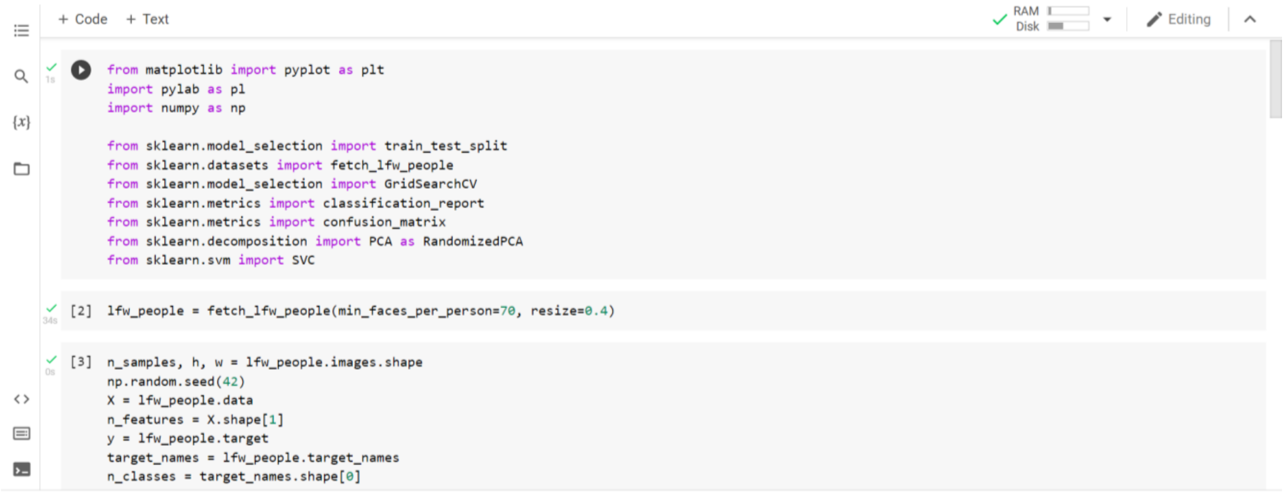
- We import the required packages and examine the raw data.
- We label new data in the correct category based on this model. To see what the decision boundary looks like, we make a custom function to plot it.
- We use the matplotlib library to plot the decision boundary.

Program Implementation [Coding]:

```
from matplotlib import pyplot as plt
import pylab as pl
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.datasets import fetch_lfw_people
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.decomposition import PCA as RandomizedPCA
from sklearn.svm import SVC
lfw_people = fetch_lfw_people(min_faces_per_person=70, resize=0.4)
n_samples, h, w = lfw_people.images.shape
np.random.seed(42)
X = lfw_people.data
n_features = X.shape[1]
y = lfw_people.target
target_names = lfw_people.target_names
n_classes = target_names.shape[0]
print("Total dataset size:")
print("n_samples:", n_samples)
print("n_features:", n_features)
print("n_classes:", n_classes)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2
5, random_state=42)
n_components = 50
pca = RandomizedPCA(n_components=n_components, whiten=True).fit(X_train
)
eigenfaces = pca.components_.reshape((n_components, h, w))
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)
param_grid = {
```

```
         'C': [1e3, 5e3, 1e4, 5e4, 1e5],
          'gamma': [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.1],
          }
clf = GridSearchCV(SVC(kernel='rbf', class_weight='balanced'), param_gr
id)
clf = clf.fit(X_train_pca, y_train)
y_pred = clf.predict(X_test_pca)
print(classification_report(y_test, y_pred, target_names=target_names))
def title(y_pred, y_test, target_names, i):
    pred_name = target_names[y_pred[i]].rsplit(' ', 1)[-1]
    true_name = target_names[y_test[i]].rsplit(' ', 1)[-1]
    return 'predicted: %s\ntrue:      %s' % (pred_name, true_name)
def plot_gallery(images, titles, h, w, n_row=3, n_col=4):

    plt.figure(figsize=(1.8 * n_col, 2.4 * n_row))
    plt.subplots_adjust(bottom=0, left=.01, right=.99, top=.90, hspace=
.35)
    for i in range(n_row * n_col):
        plt.subplot(n_row, n_col, i + 1)
        plt.imshow(images[i].reshape((h, w)), cmap=plt.cm.gray)
        plt.title(titles[i], size=12)
        plt.xticks(())
        plt.yticks(())
prediction_titles = [title(y_pred, y_test, target_names, i)
                         for i in range(y_pred.shape[0])]
plot_gallery(X_test, prediction_titles, h, w)
from sklearn.metrics import accuracy_score
score = accuracy_score(y_test, y_pred)
print(score)
```

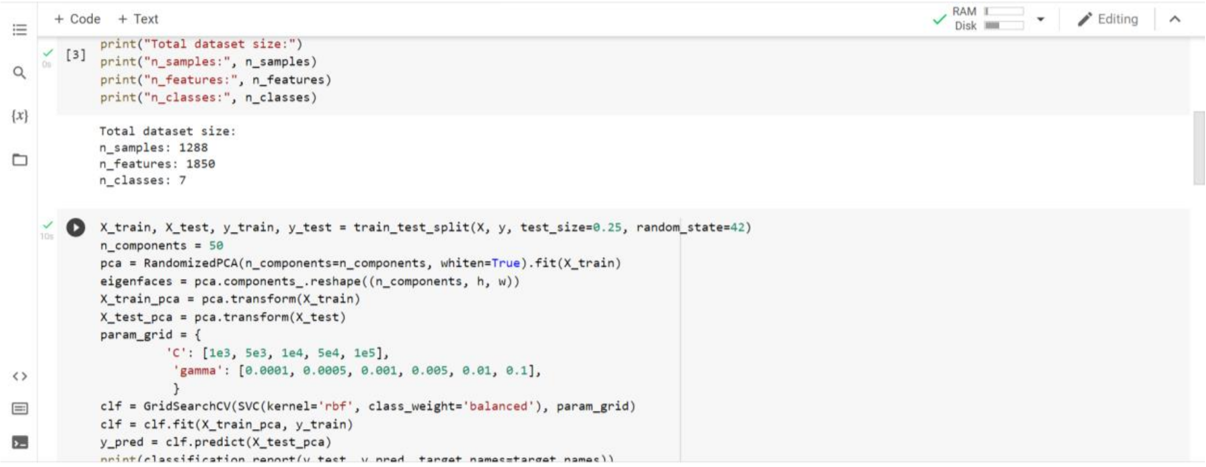
<u>Screenshots of the Outputs:</u>



```
from matplotlib import pyplot as plt
import pylab as pl
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.datasets import fetch_lfw_people
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.decomposition import PCA as RandomizedPCA
from sklearn.svm import SVC
```

```
[2] lfw_people = fetch_lfw_people(min_faces_per_person=70, resize=0.4)
```

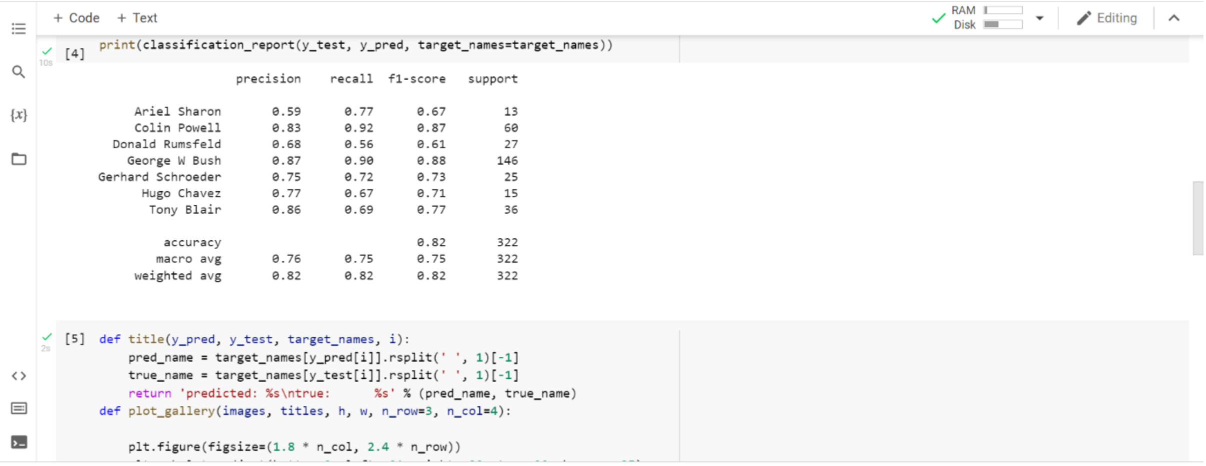
```
[3] n_samples, h, w = lfw_people.images.shape
    np.random.seed(42)
    X = lfw_people.data
    n_features = X.shape[1]
    y = lfw_people.target
    target_names = lfw_people.target_names
    n_classes = target_names.shape[0]
```



```
print("Total dataset size:")
print("n_samples:", n_samples)
print("n_features:", n_features)
print("n_classes:", n_classes)
```

```
Total dataset size:
n_samples: 1288
n_features: 1850
n_classes: 7
```

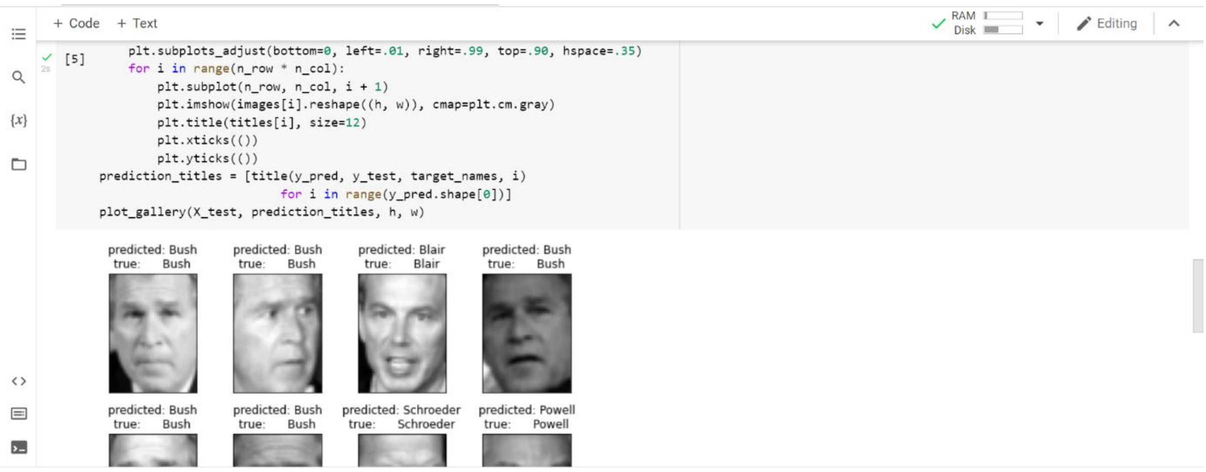
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
n_components = 50
pca = RandomizedPCA(n_components=n_components, whiten=True).fit(X_train)
eigenfaces = pca.components_.reshape((n_components, h, w))
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)
param_grid = {
         'C': [1e3, 5e3, 1e4, 5e4, 1e5],
          'gamma': [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.1],
          }
clf = GridSearchCV(SVC(kernel='rbf', class_weight='balanced'), param_grid)
clf = clf.fit(X_train_pca, y_train)
y_pred = clf.predict(X_test_pca)
print(classification_report(y_test, y_pred, target_names=target_names))
```



```
print(classification_report(y_test, y_pred, target_names=target_names))
```

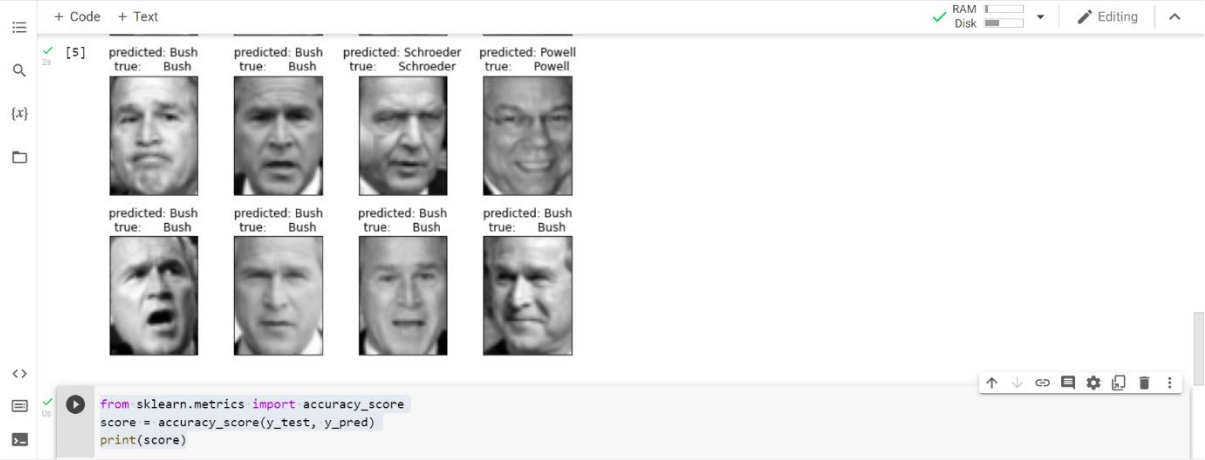
```
                   precision    recall  f1-score   support

     Ariel Sharon       0.59      0.77      0.67        13
     Colin Powell       0.83      0.92      0.87        60
  Donald Rumsfeld       0.68      0.56      0.61        27
    George W Bush       0.87      0.90      0.88       146
Gerhard Schroeder       0.75      0.72      0.73        25
      Hugo Chavez       0.77      0.67      0.71        15
       Tony Blair       0.86      0.69      0.77        36

         accuracy                           0.82       322
        macro avg       0.76      0.75      0.75       322
     weighted avg       0.82      0.82      0.82       322
```

```
def title(y_pred, y_test, target_names, i):
    pred_name = target_names[y_pred[i]].rsplit(' ', 1)[-1]
    true_name = target_names[y_test[i]].rsplit(' ', 1)[-1]
    return 'predicted: %s\ntrue:      %s' % (pred_name, true_name)
def plot_gallery(images, titles, h, w, n_row=3, n_col=4):

    plt.figure(figsize=(1.8 * n_col, 2.4 * n_row))
```

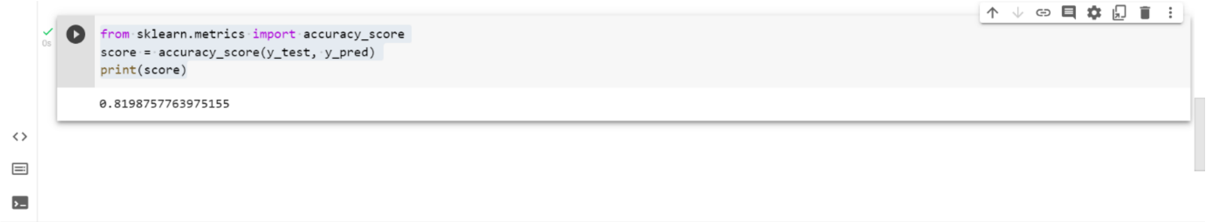


```
    plt.subplots_adjust(bottom=0, left=.01, right=.99, top=.90, hspace=.35)
    for i in range(n_row * n_col):
        plt.subplot(n_row, n_col, i + 1)
        plt.imshow(images[i].reshape((h, w)), cmap=plt.cm.gray)
        plt.title(titles[i], size=12)
        plt.xticks(())
        plt.yticks(())
prediction_titles = [title(y_pred, y_test, target_names, i)
                     for i in range(y_pred.shape[0])]
plot_gallery(X_test, prediction_titles, h, w)
```

```
predicted: Bush      predicted: Bush      predicted: Blair          predicted: Bush
true:    Bush        true:    Bush        true:    Blair            true:    Bush

predicted: Bush      predicted: Bush      predicted: Schroeder      predicted: Powell
true:    Bush        true:    Bush        true:    Schroeder        true:    Powell
```



```
from sklearn.metrics import accuracy_score
score = accuracy_score(y_test, y_pred)
print(score)
```



```
from sklearn.metrics import accuracy_score
score = accuracy_score(y_test, y_pred)
print(score)
```

```
0.8198757763975155
```

RESULT:

Hence, we applied supervised and unsupervised machine learning algorithms (PCA, SVM) on an open source dataset and performed image recognition successfully. SVM is the preferred algorithm as it provides greater accuracy.

Signature of the Student

Deeksha Rawat

RA1911027010005

Date: 04-04-2022 **Ex No: 9**	**Implementation of NLP Algorithms For An Application**	**Name: Deeksha Rawat** **Registration Number: RA1911027010005** **Section: N1** **Lab Batch: 1** **Day Order: 2**

1. Python Autocorrection System using NLP.

AIM: To implement Python Autocorrection System using NLP.

Description of the Concept or Problem:

Natural Language Processing (NLP) is an important branch in the domain of computer science and artificial intelligence, which is an integration of linguistics, computer science and mathematics. The main purpose is to develop an effective computer system, especially software system, which can achieve interaction between human and computer system using natural language. Interaction with computers using natural language is long-term pursuit of human-beings. It is difficult for computers to understand natural language, which is mainly caused by the difficulty of eliminating the ambiguity that exists at all levels of texts or dialogues of natural languages.

One of the features that use Natural Language Processing (NLP) is the Autocorrect function. It is specially programmed to generalize all the correct words in the dictionary and looks for the words that are the most comparable to those words not in the vocabulary. The Autocorrect model is programmed to correct spellings and errors while inputting text and locating the most comparable related words. It is completely based on NLP that compares the words in the vocabulary dictionary and the typed words on the keyboard.If the typed word is found in the dictionary, the autocorrect feature assumes you typed the correct term. If the word does not exist, the tool identifies the most comparable words in our smartphone’s history, as it indicates.

Manual Solution:

1. Identifying Misspelled Word

   A word is misspelled if the text is not found on the vocabulary of the corpus (dictionary), then the autocorrect system flags out for correction. Let us consider an example, how would we get to know the word “drea” is spelled incorrectly or correctly? If a word is spelled correctly then the word will be found in a dictionary and if it is not there then it is probably a misspelled word. Hence, when a word is not found in a dictionary, we will flag it for correction.

2. Find strings that are N-edit-distance away from the misspelled word.

   Editing is an operation performed on the string to change it to another string. The n represents the edit distance like 1, 2, 3, so on, which keeps track of the number of edit operations to be done.
   Hence, the edit distance is the count of the number of operations performed on a word to edit it.
   The following are examples of edits:
   INSERT - a letter should be added.
   DELETE - removes a letter.
   SWAP - swaps two adjacent letters.
   REPLACE - changes one letter to another.

   With these four edits, we are proficient in modifying any string. So the combination of edits allows us to find a list of all possible strings that are n edits to perform.
   For autocorrect, we take n usually between 1 to 3 edits.

3. Filtering of Candidates

   Only correctly spelled words from the created candidate list are considered, so that we can compare them to the words in the corpus to filter out the ones that don't exist. Here we want to consider only correctly spelled real words from our generated candidate list so we can compare the words to a known dictionary (like we did in the first step) and then filter out the words in our generated candidate list that do not appear in the known "dictionary"

4. Calculate Probabilities of Words

   The probabilities of the words are calculated based on the following formula:
   [P(w) = C(w)/V]
   P(w)- the probability of a word w.
   C(w) - number of times (frequency) word appears in the vocabulary dictionary.
   V - the total sum of words in the dictionary.
   When the probabilities are calculated, the actual list of words is grouped by the most likely word from the created candidates. This requires word frequencies that we know and the total number of words in the corpus (also known as dictionary).

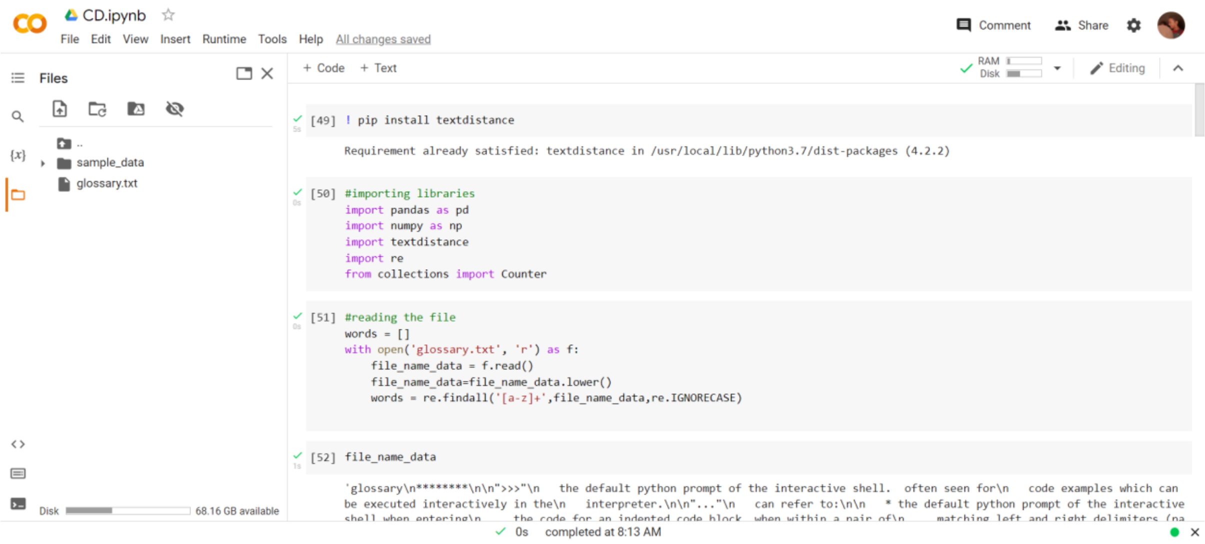
<u>Program Implementation [Coding]:</u>

```
import re

from collections import Counter

import numpy as np
import pandas as pd


def process(fname):
    words = []
    with open(fname) as fl:
        data = fl.read()
    data = data.lower()
    words = re.findall('\w+',data)
    return words


words = process('glossary.txt')
vocab = set(words)


def count(words):
    countd = {}
    countd = Counter(words)
    return countd


count_dict = count(words)


def probabilities(count_dict):
    p = {}
    l = sum(count_dict.values())
    for k in count_dict.keys():
        p[k] = count_dict[k] / l
    return p
probs = probabilities(count_dict)
def delete(word):
    d = []
    s = []
    for i in range(len(word)):
        s.append((word[:i],word[i:]))
    for j,k in s:
        d.append(j+k[1:])
    return d

def switch(word):
    sw = []
    sp = []
    l=len(word)
    for i in range(l):
        sp.append((word[:i],word[i:]))
    sw = [j + k[1] + k[0] + k[2:] for j,k in sp if len(k) >= 2]
    return sw

def replace(word):
    letters = 'abcdefghijklmnopqrstuvwxyz'
    r = []
    s = []
    for i in range(len(word)):
        s.append((word[0:i],word[i:]))
    r = [j + l + (k[1:] if len(k)> 1 else '') for j,k in s if k for l in letters]
    rs=set(r)
    r = sorted(list(rs))
    return r

def insert(word):
    letters = 'abcdefghijklmnopqrstuvwxyz'
    i = []
    s = []
    for j in range(len(word)+1):
        s.append((word[0:j],word[j:]))
    i = [ m + l + n for m,n in s for l in letters]
    return i


def edit1(word):
    edit1s = set()
    edit1s.update(delete(word))
    edit1s.update(replace(word))
    edit1s.update(insert(word))
    edit1s.update(switch(word))
    return edit1s


def edit2(word):
    edit2s = set()
    edit_one = edit1(word)
    for w in edit_one:
        if w:
            edit_two = edit1(w)
            edit2s.update(edit_two)
    return edit2s
def correct(word, probs, vocab, n=2):


    suggestions = []
    n_best = []
    suggestions = list((word in vocab and word) or edit1(word).intersection(vocab) or edit2(wo
rd).intersection(vocab))
    n_best = [[s,probs[s]] for s in list((suggestions))]
    print("Input Word = ", word, "\nSuggestions = ", suggestions[::-1],"Probability=", probs)
    return n_best
```

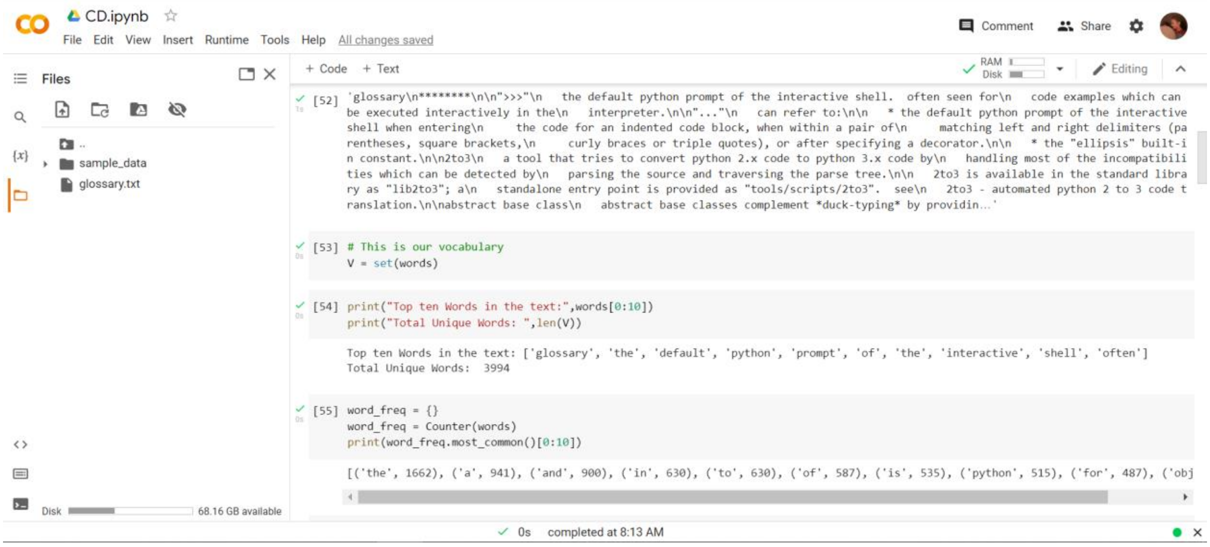
my_word = " chra”

corrections = correct(my_word, probs, vocab, 4)

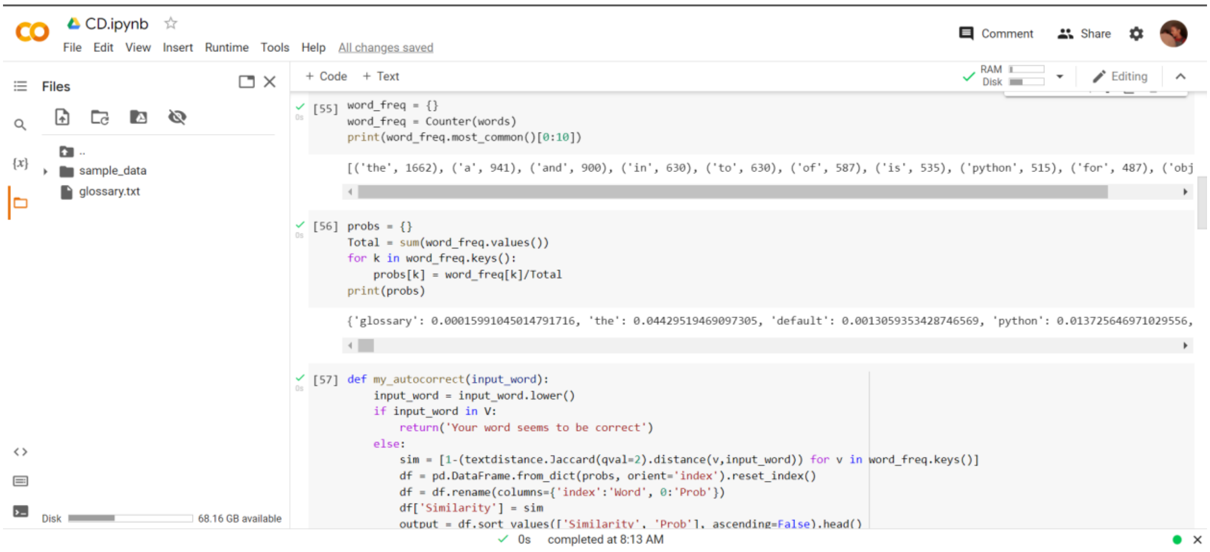
<u>Screenshots of the Outputs:</u>



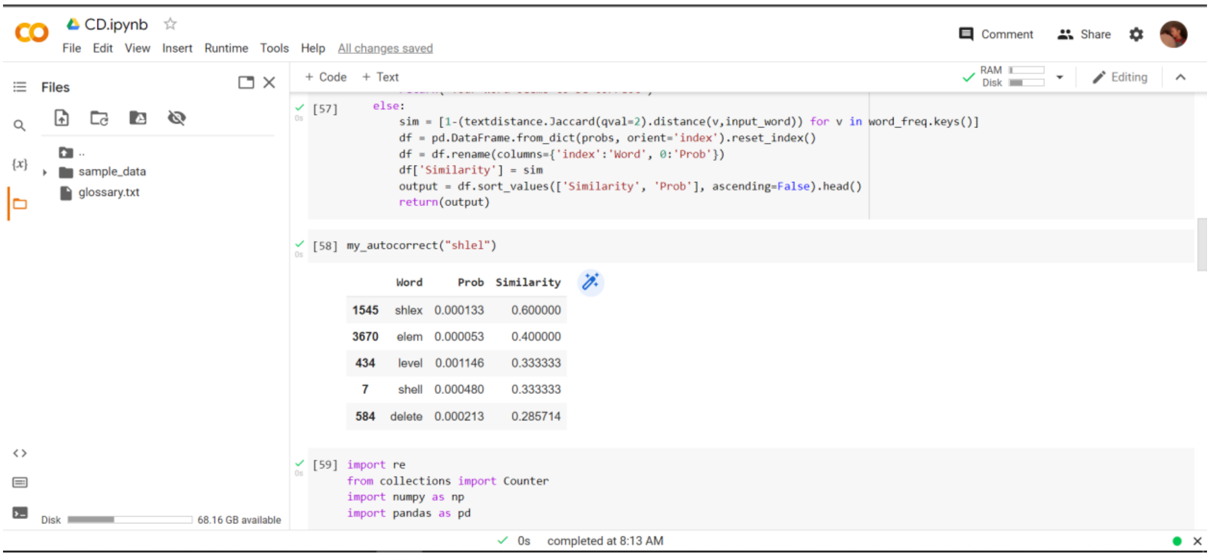
```
[49] ! pip install textdistance

     Requirement already satisfied: textdistance in /usr/local/lib/python3.7/dist-packages (4.2.2)

[50] #importing libraries
     import pandas as pd
     import numpy as np
     import textdistance
     import re
     from collections import Counter

[51] #reading the file
     words = []
     with open('glossary.txt', 'r') as f:
         file_name_data = f.read()
         file_name_data=file_name_data.lower()
         words = re.findall('[a-z]+',file_name_data,re.IGNORECASE)

[52] file_name_data

     'glossary\n********\n\n">>>"\n   the default python prompt of the interactive shell.  often seen for\n   code examples which can
     be executed interactively in the\n   interpreter.\n\n"..."\n   can refer to:\n\n   * the default python prompt of the interactive
     shell when entering\n     the code for an indented code block  when within a pair of\n     matching left and right delimiters (pa
```



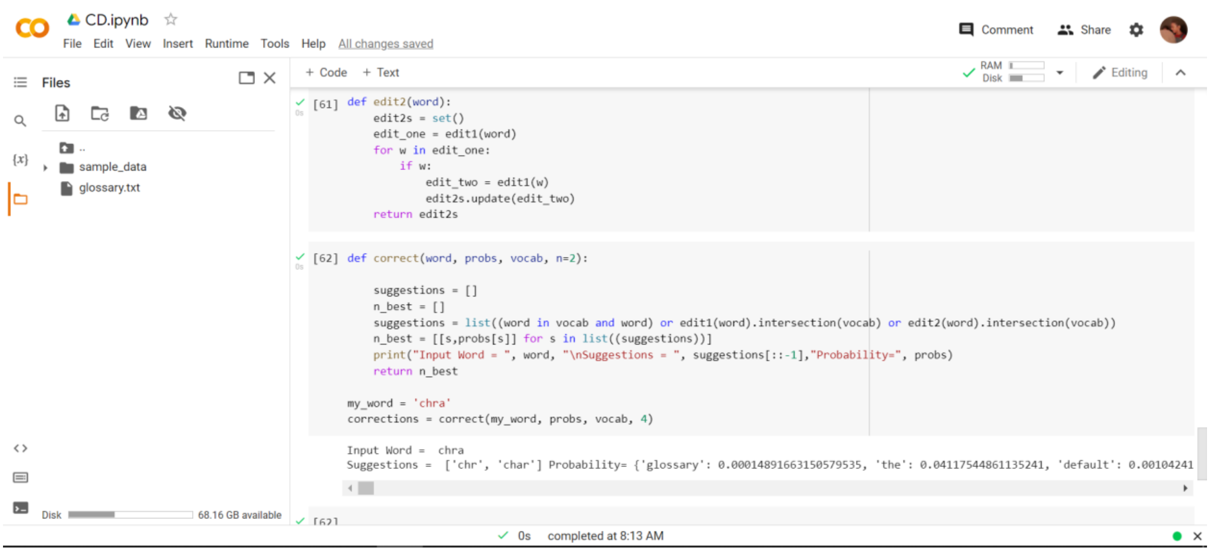
```
[52] 'glossary\n********\n\n">>>"\n   the default python prompt of the interactive shell.  often seen for\n   code examples which can
     be executed interactively in the\n   interpreter.\n\n"..."\n   can refer to:\n\n   * the default python prompt of the interactive
     shell when entering\n     the code for an indented code block, when within a pair of\n     matching left and right delimiters (pa
     rentheses, square brackets,\n     curly braces or triple quotes), or after specifying a decorator.\n\n   * the "ellipsis" built-i
     n constant.\n\n2to3\n   a tool that tries to convert python 2.x code to python 3.x code by\n   handling most of the incompatibili
     ties which can be detected by\n   parsing the source and traversing the parse tree.\n\n   2to3 is available in the standard libra
     ry as "lib2to3"; a\n   standalone entry point is provided as "tools/scripts/2to3".  see\n   2to3 - automated python 2 to 3 code t
     ranslation.\n\nabstract base class\n   abstract base classes complement *duck-typing* by providin...'

[53] # This is our vocabulary
     V = set(words)

[54] print("Top ten Words in the text:",words[0:10])
     print("Total Unique Words: ",len(V))

     Top ten Words in the text: ['glossary', 'the', 'default', 'python', 'prompt', 'of', 'the', 'interactive', 'shell', 'often']
     Total Unique Words:  3994

[55] word_freq = {}
     word_freq = Counter(words)
     print(word_freq.most_common()[0:10])

     [('the', 1662), ('a', 941), ('and', 900), ('in', 630), ('to', 630), ('of', 587), ('is', 535), ('python', 515), ('for', 487), ('obj
```



```
[55] word_freq = {}
     word_freq = Counter(words)
     print(word_freq.most_common()[0:10])

     [('the', 1662), ('a', 941), ('and', 900), ('in', 630), ('to', 630), ('of', 587), ('is', 535), ('python', 515), ('for', 487), ('obj

[56] probs = {}
     Total = sum(word_freq.values())
     for k in word_freq.keys():
         probs[k] = word_freq[k]/Total
     print(probs)

     {'glossary': 0.00015991045014791716, 'the': 0.04429519469097305, 'default': 0.0013059353428746569, 'python': 0.013725646971029556,

[57] def my_autocorrect(input_word):
         input_word = input_word.lower()
         if input_word in V:
             return('Your word seems to be correct')
         else:
             sim = [1-(textdistance.Jaccard(qval=2).distance(v,input_word)) for v in word_freq.keys()]
             df = pd.DataFrame.from_dict(probs, orient='index').reset_index()
             df = df.rename(columns={'index':'Word', 0:'Prob'})
             df['Similarity'] = sim
             output = df.sort_values(['Similarity', 'Prob'], ascending=False).head()
```



```
[57]     else:
             sim = [1-(textdistance.Jaccard(qval=2).distance(v,input_word)) for v in word_freq.keys()]
             df = pd.DataFrame.from_dict(probs, orient='index').reset_index()
             df = df.rename(columns={'index':'Word', 0:'Prob'})
             df['Similarity'] = sim
             output = df.sort_values(['Similarity', 'Prob'], ascending=False).head()
             return(output)

[58] my_autocorrect("shlel")

            Word      Prob  Similarity
     1545   shlex   0.000133    0.600000
     3670   elem    0.000053    0.400000
      434   level   0.001146    0.333333
        7   shell   0.000480    0.333333
      584   delete  0.000213    0.285714

[59] import re
     from collections import Counter
     import numpy as np
     import pandas as pd
```



```
[61] def edit2(word):
         edit2s = set()
         edit_one = edit1(word)
         for w in edit_one:
             if w:
                 edit_two = edit1(w)
                 edit2s.update(edit_two)
         return edit2s

[62] def correct(word, probs, vocab, n=2):

         suggestions = []
         n_best = []
         suggestions = list((word in vocab and word) or edit1(word).intersection(vocab) or edit2(word).intersection(vocab))
         n_best = [[s,probs[s]] for s in list((suggestions))]
         print("Input Word = ", word, "\nSuggestions = ", suggestions[::-1],"Probability=", probs)
         return n_best

     my_word = 'chra'
     corrections = correct(my_word, probs, vocab, 4)

     Input Word =  chra
     Suggestions =  ['chr', 'char'] Probability= {'glossary': 0.00014891663150579535, 'the': 0.04117544861135241, 'default': 0.00104241

[62]
```

Conclusion

We have seen the logic and and implementation a simple autocorrection algorithm. This mental model can serve as a basis for creating more complex models, which are not simply based on the probability of occurrence of the terms, but also on the context and on the dictionary that the user has manually created through the various corrections provided while typing.

Signature of the Student

Deeksha Rawat

RA1911027010005

Date: 17-04-2022 **Ex No: 10**	**Implementation of Deep Learning Algorithm For An Application**	**Name: Deeksha Rawat** **Registration Number: RA1911027010005** **Section: N1** **Lab Batch: 1** **Day Order: 2**

IMAGE CLASSIFICATION USING CNN

AIM: To implement CNN algorithm on an image classification model using an open-source dataset.

Description of the Concept or Problem:

CNN (Convolutional Neural Networks) comes under the umbrella of deep learning neural networks. It is commonly used to analyze visual imagery and perform highly accurate image classification. In an image classification problem, we use CNN to take an input, i.e., a picture, output a class or probability that the input belongs to a particular class. Compared to other image classification algorithms, CNN does very little preprocessing, making it possible for them to learn filters rather than creating them as is the case in most algorithms. A CNN extracts features from images and hence makes for accurate computer vision tasks. CNNs learn feature detection through tens or hundreds of hidden layers. Each layer increases the complexity of the learned features.

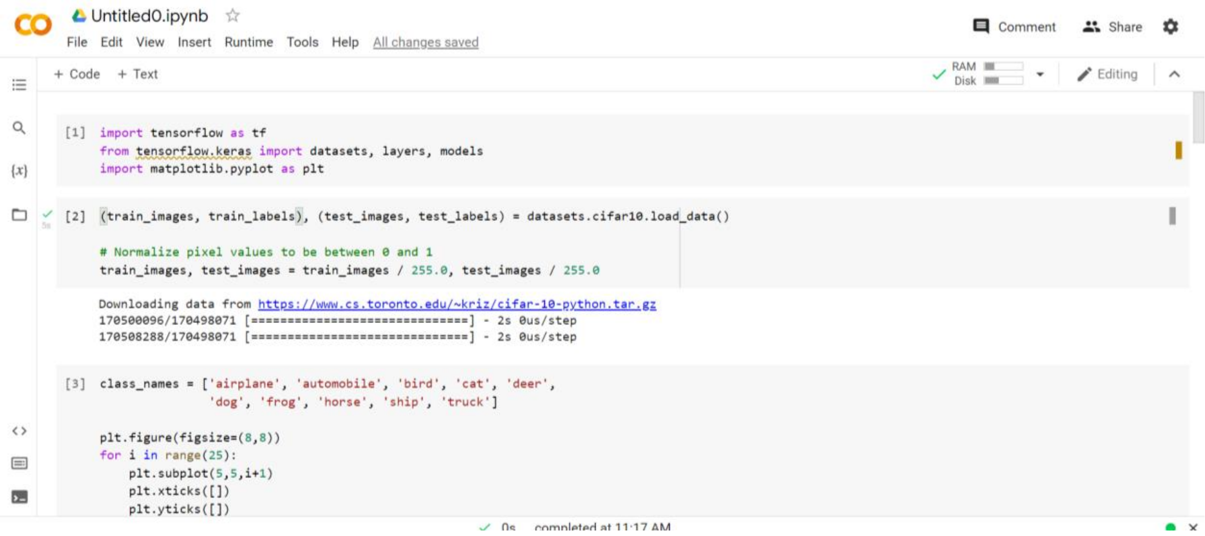
Manual Solution:

1. We first import the libraries, namely tensorflow and matplotlib.
2. The data is loaded. In this case, we use the CIFAR10 dataset. The dataset is divided into 50,000 training images and 10,000 testing images. The classes are mutually exclusive and there is no overlap between them.
3. To verify that the dataset looks correct, we plot the first 25 images from the training set and display the class name below each image.
4. We then create the convolutional base. As input, CNN takes tensors of shape, ignoring batch size.
5. We add Dense Layers on the top to perform classification in order to complete the model. Dense Layers take vector inputs.
6. The model is compiled and trained.
7. Finally, the model is evaluated for accuracy.

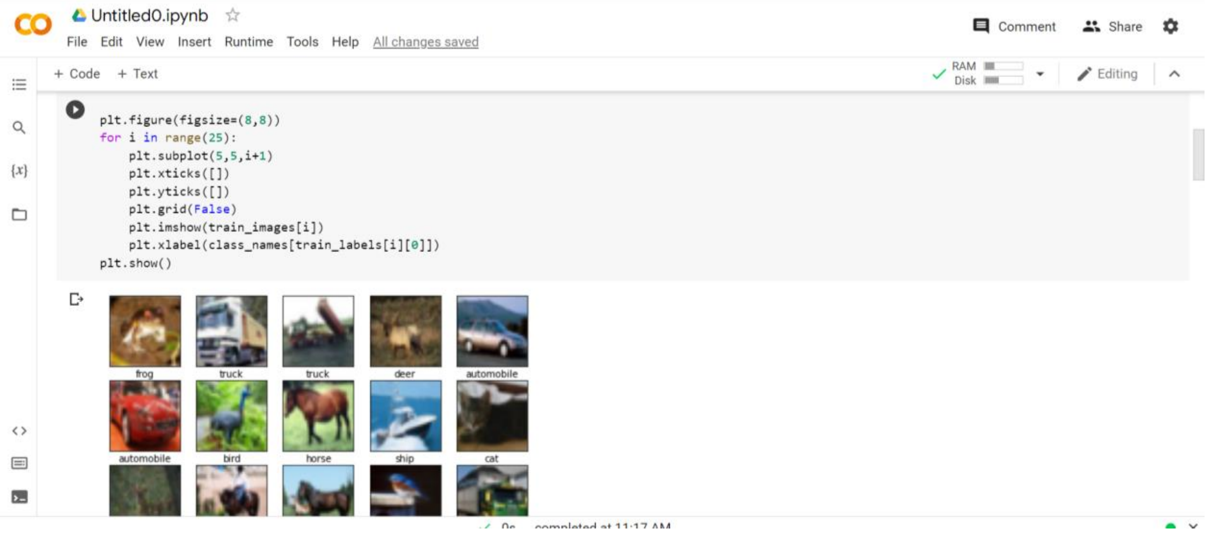
Program Implementation [Coding]:

```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
(train_images, train_labels), (test_images, test_labels) = datasets.cif
ar10.load_data()

train_images, test_images = train_images / 255.0, test_images / 255.0
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']
plt.figure(figsize=(8,8))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i])
    plt.xlabel(class_names[train_labels[i][0]])
plt.show()
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32,
 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.summary()
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))
model.summary()
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_l
ogits=True),
              metrics=['accuracy'])
history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))
plt.plot(history.history['accuracy'],label='accuracy')
plt.plot(history.history['val_accuracy'],label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')

test_loss, test_acc = model.evaluate(test_images,
                                     test_labels,
                                     verbose=2)
print('Test Accuracy is',test_acc)
```

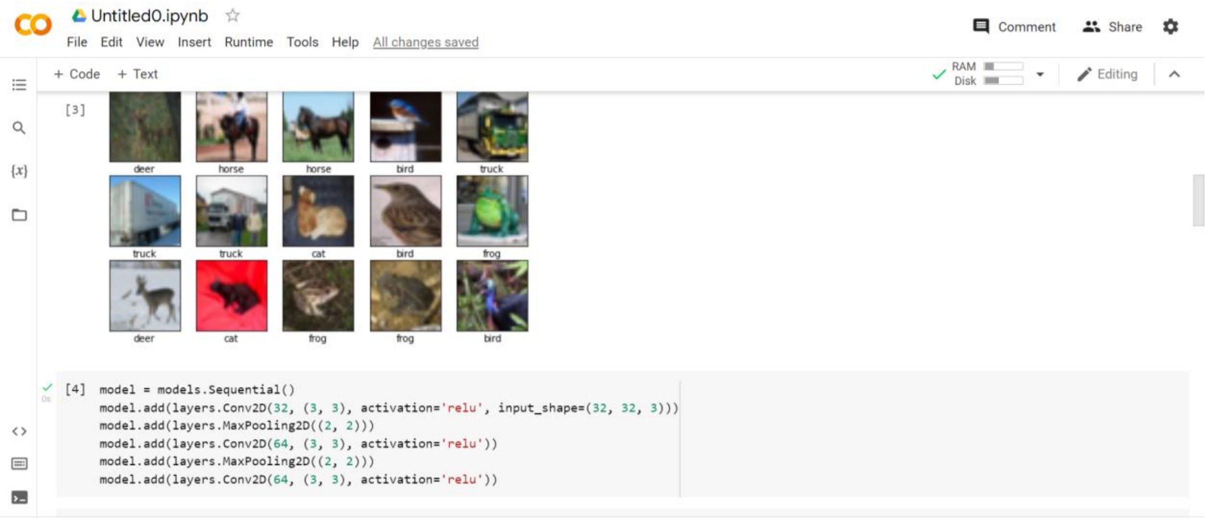
Screenshots of the Outputs:


Untitled0.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
RAM
Disk
Editing
✓ 0s completed at 11:17 AM

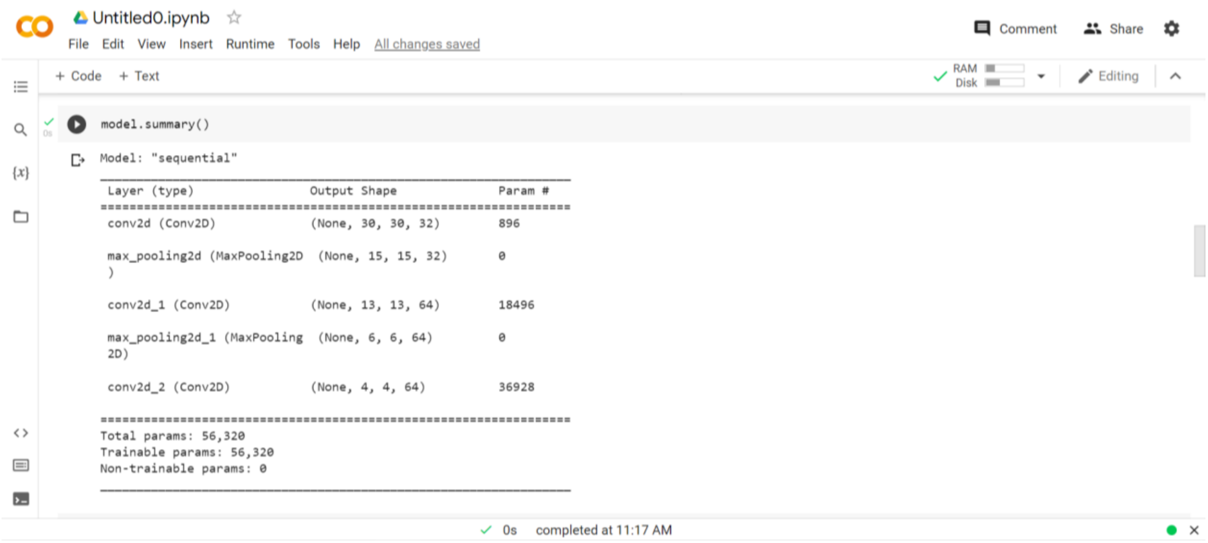
```
[1] import tensorflow as tf
    from tensorflow.keras import datasets, layers, models
    import matplotlib.pyplot as plt

[2] (train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()

    # Normalize pixel values to be between 0 and 1
    train_images, test_images = train_images / 255.0, test_images / 255.0

    Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
    170500096/170498071 [==============================] - 2s 0us/step
    170508288/170498071 [==============================] - 2s 0us/step

[3] class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
                   'dog', 'frog', 'horse', 'ship', 'truck']

    plt.figure(figsize=(8,8))
    for i in range(25):
        plt.subplot(5,5,i+1)
        plt.xticks([])
        plt.yticks([])
```



```
plt.figure(figsize=(8,8))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i])
    plt.xlabel(class_names[train_labels[i][0]])
plt.show()
```

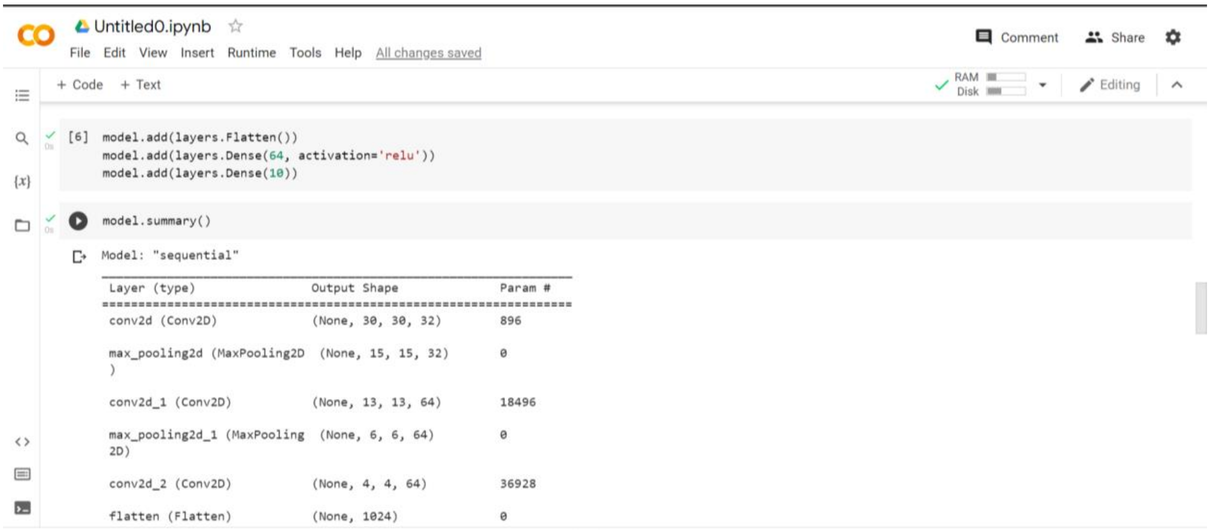


```
[4] model = models.Sequential()
    model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```


Untitled0.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
RAM Disk
Editing
✓ 0s completed at 11:17 AM

```
model.summary()
```

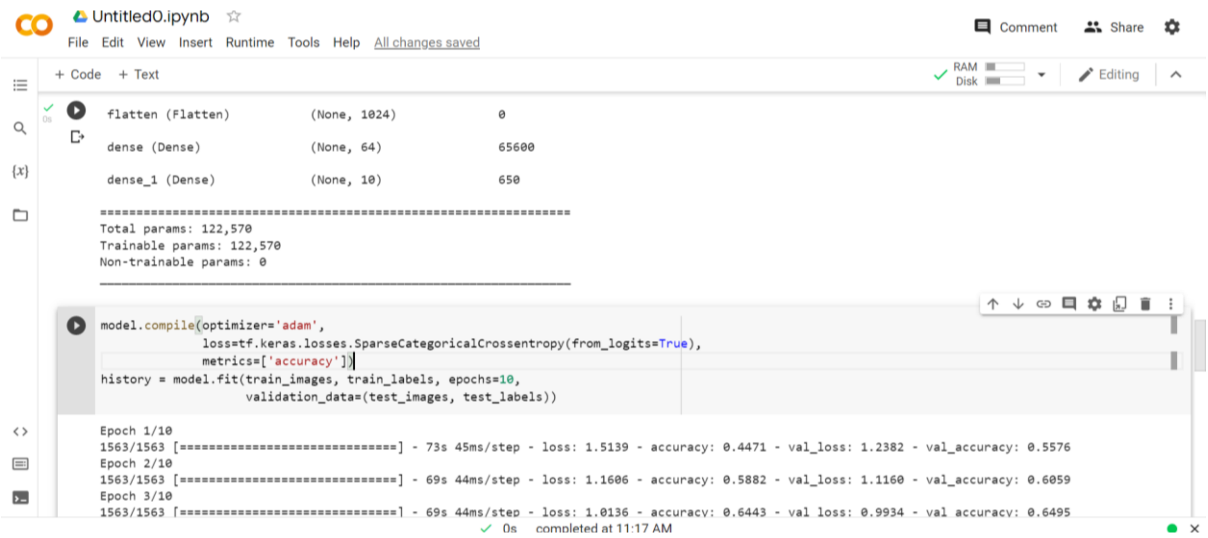
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 32)        896

 max_pooling2d (MaxPooling2D  (None, 15, 15, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 13, 13, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 6, 6, 64)         0
 2D)

 conv2d_2 (Conv2D)           (None, 4, 4, 64)          36928

=================================================================
Total params: 56,320
Trainable params: 56,320
Non-trainable params: 0
_________________________________________________________________
```



```
[6] model.add(layers.Flatten())
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(10))
```

```
model.summary()
```

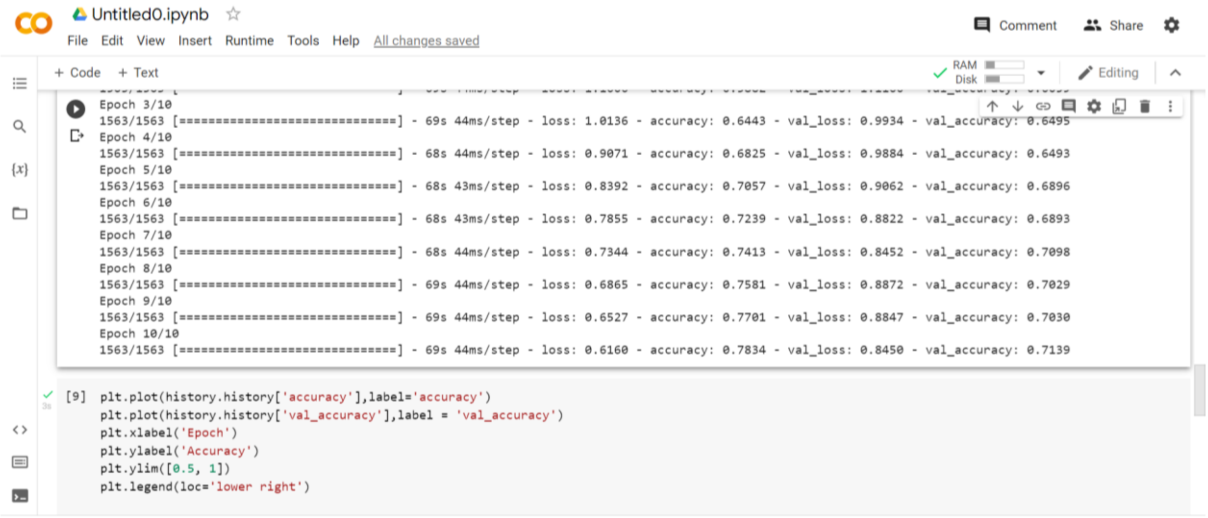
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 32)        896

 max_pooling2d (MaxPooling2D  (None, 15, 15, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 13, 13, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 6, 6, 64)         0
 2D)

 conv2d_2 (Conv2D)           (None, 4, 4, 64)          36928

 flatten (Flatten)           (None, 1024)              0
```


Untitled0.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
RAM Disk
Editing
✓ 0s completed at 11:17 AM

```
 flatten (Flatten)           (None, 1024)              0

 dense (Dense)               (None, 64)                65600

 dense_1 (Dense)             (None, 10)                650

=================================================================
Total params: 122,570
Trainable params: 122,570
Non-trainable params: 0
_________________________________________________________________
```

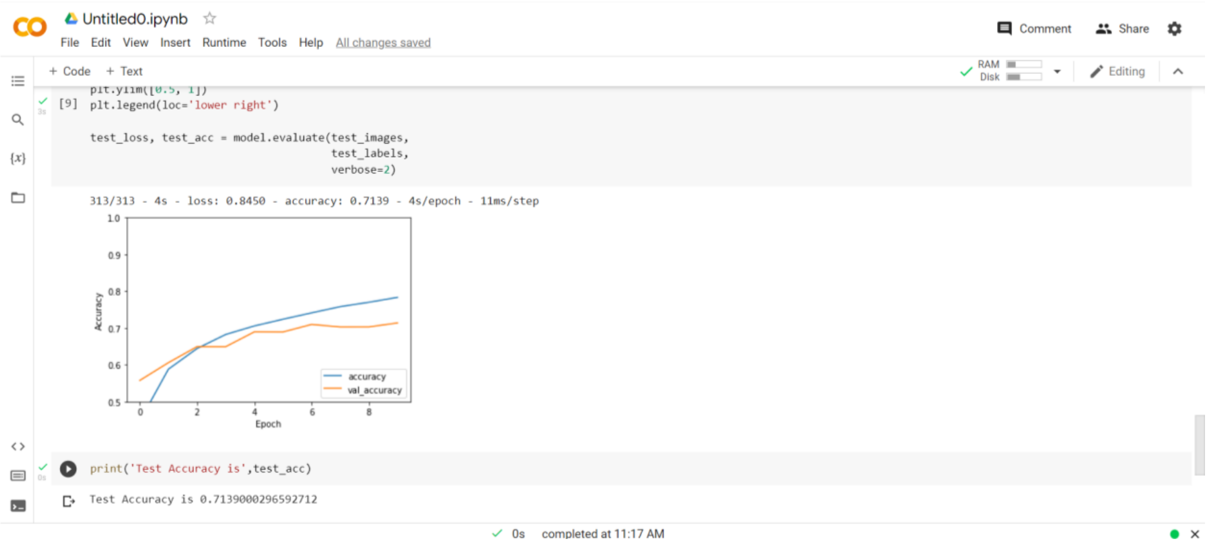
```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))
```

```
Epoch 1/10
1563/1563 [==============================] - 73s 45ms/step - loss: 1.5139 - accuracy: 0.4471 - val_loss: 1.2382 - val_accuracy: 0.5576
Epoch 2/10
1563/1563 [==============================] - 69s 44ms/step - loss: 1.1606 - accuracy: 0.5882 - val_loss: 1.1160 - val_accuracy: 0.6059
Epoch 3/10
1563/1563 [==============================] - 69s 44ms/step - loss: 1.0136 - accuracy: 0.6443 - val_loss: 0.9934 - val_accuracy: 0.6495
```



```
Epoch 3/10
1563/1563 [==============================] - 69s 44ms/step - loss: 1.0136 - accuracy: 0.6443 - val_loss: 0.9934 - val_accuracy: 0.6495
Epoch 4/10
1563/1563 [==============================] - 68s 44ms/step - loss: 0.9071 - accuracy: 0.6825 - val_loss: 0.9884 - val_accuracy: 0.6493
Epoch 5/10
1563/1563 [==============================] - 68s 43ms/step - loss: 0.8392 - accuracy: 0.7057 - val_loss: 0.9062 - val_accuracy: 0.6896
Epoch 6/10
1563/1563 [==============================] - 68s 43ms/step - loss: 0.7855 - accuracy: 0.7239 - val_loss: 0.8822 - val_accuracy: 0.6893
Epoch 7/10
1563/1563 [==============================] - 68s 44ms/step - loss: 0.7344 - accuracy: 0.7413 - val_loss: 0.8452 - val_accuracy: 0.7098
Epoch 8/10
1563/1563 [==============================] - 69s 44ms/step - loss: 0.6865 - accuracy: 0.7581 - val_loss: 0.8872 - val_accuracy: 0.7029
Epoch 9/10
1563/1563 [==============================] - 69s 44ms/step - loss: 0.6527 - accuracy: 0.7701 - val_loss: 0.8847 - val_accuracy: 0.7030
Epoch 10/10
1563/1563 [==============================] - 69s 44ms/step - loss: 0.6160 - accuracy: 0.7834 - val_loss: 0.8450 - val_accuracy: 0.7139
```

```
[9] plt.plot(history.history['accuracy'],label='accuracy')
    plt.plot(history.history['val_accuracy'],label = 'val_accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.ylim([0.5, 1])
    plt.legend(loc='lower right')
```



```
    plt.ylim([0.5, 1])
[9] plt.legend(loc='lower right')

    test_loss, test_acc = model.evaluate(test_images,
                                         test_labels,
                                         verbose=2)

    313/313 - 4s - loss: 0.8450 - accuracy: 0.7139 - 4s/epoch - 11ms/step
```

```
print('Test Accuracy is',test_acc)

Test Accuracy is 0.7139000296592712
```

✓ 0s completed at 11:17 AM

RESULT: Hence successfully implemented an image classification model on an open-source dataset using CNN. The accuracy was clocked at 71.4% with a loss of 0.8.

Signature of the Student

Deeksha Rawat

RA1911027010005