PRACTICAL FILE

ON

ARTIFICIAL INTELLIGENCE LAB

(KCA-351)

Session 2023-24



**Department of Computer**

**Application**

### Submitted To:

Ms. Neelam Rawat

Department of Computer Applications

KIET Group of Institutions

### Submitted By:

Shivanshu Panwar

2200290140148

**Artificial Intelligence (KCA-351)**

**Lab Manual**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S.No.** | Program List | **Due Date** | **Completion Date** | **Faculty Signature** |
| **Ex-1** | Installation of Python/Anaconda and introduction of Google Colab | First Week |  |  |
| **Ex-2** | Write a program to implement water jug problem' | First Week |  |  |
| **Ex-3** | Write a program to implement missionary and cannibal algorithm | Second Week |  |  |
| **Ex-4** | Write a program to implement breadth first search | Second Week |  |  |
| **Ex-5** | Write a program to implement depth first search | Third Week |  |  |
| **Ex-6** | Write a program to implement 8 Queen Problems | Third Week |  |  |
| **Ex-7** | Write a program to implement first order logic | Fourth Week |  |  |
| **Ex-8** | Write a program to demonstrate the working of Bayesian network. | Fourth Week |  |  |
| **Ex-9** | Write a program to Implement pattern recognition problems of handwritten character/ digit recognition | Fifth Week |  |  |
| **Ex-10** | Write a program to Implement pattern recognition problems of speech recognition | Sixth Week |  |  |
| **Ex-11** | Write a program to implement naive bayes classification problem | Seventh Week |  |  |
| **Ex-12** | Write a program to implement k-mean clustering problem | Eight Week |  |  |
| **Ex-13** | Write a program to convert text to speech using NLP tool | Ninth Week |  |  |
| **Ex-14** | Pilot Project in AI using Python | Tenth Week |  |  |

|  |  |
| --- | --- |
| **EX No-1** | **Installation of Python/Anaconda and introduction of Google Colab** |

Python is a widely used high-level programming language. To write and execute code in python, we first need to install Python on our system.

Installing Python on Windows takes a series of few easy steps.

### Step 1 − Select Version of Python to Install

Python has various versions available with differences between the syntax and working of different versions of the language. We need to choose the version which we want to use or need. There are different versions of Python 2 and Python 3 available

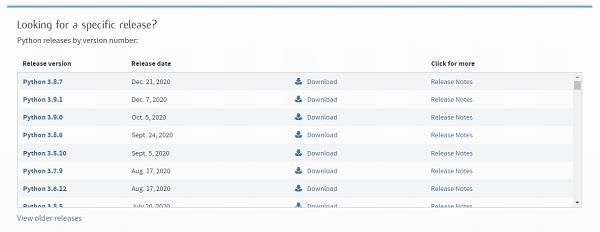
**Step 2 − Download Python Executable Installer**

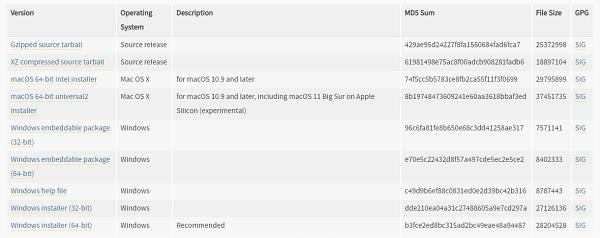
On the web browser, in the official site of python ([www.python.org](https://www.tutorialspoint.com/www.python.org)), move to the Download for Windows section.

All the available versions of Python will be listed. Select the version required by you and click on Download. Let suppose, we chose the Python 3.9.1 version.

On clicking download, various available executable installers shall be visible with different operating system specifications. Choose the installer which suits your system operating system and download the instlaller. Let suppose, we select the Windows installer(64 bits).

The download size is less than 30MB





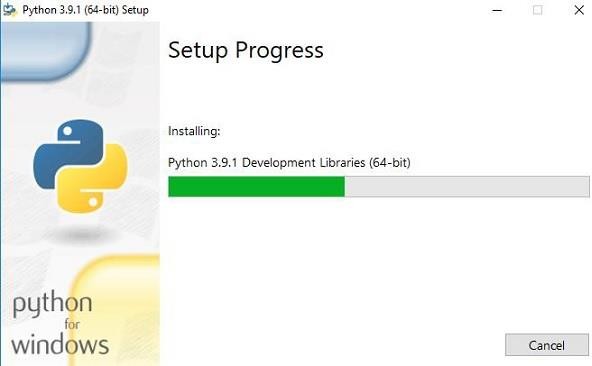
### Step 3 − Run Executable Installer

We downloaded the Python 3.9.1 Windows 64 bit installer.

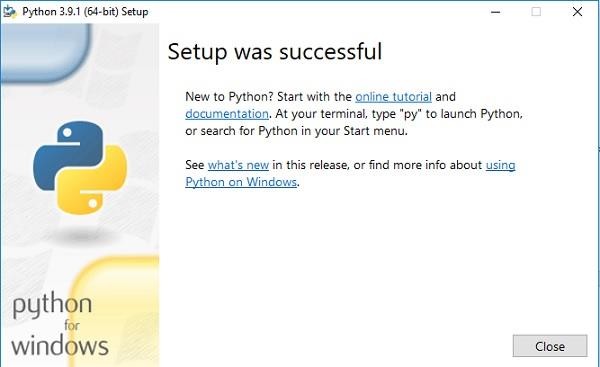
Run the installer. Make sure to select both the checkboxes at the bottom and then click Install New.



On clicking the Install Now, The installation process starts.



The installation process will take few minutes to complete and once the installation is successful, the following screen is displayed.



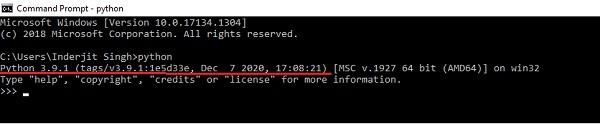
**Step 4 − Verify Python is installed on Windows**

To ensure if Python is succesfully installed on your system. Follow the given steps −

Open the command prompt.

Type ‘python’ and press enter.

The version of the python which you have installed will be displayed if the python is successfully installed on your windows

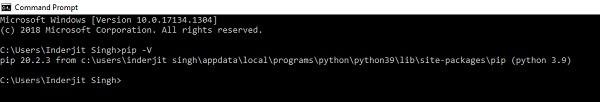


**Step 5 − Verify Pip was installed**

Pip is a powerful package management system for Python software packages. Thus, make sure that you have it installed.

To verify if pip was installed, follow the given steps −

* Open the command prompt.
* Enter pip –V to check if pip was installed.
* The following output appears if pip is installed successfully.



We have successfully installed python and pip on our Windows system.

|  |  |
| --- | --- |
| **EX NO – 2** | **Write a program to implement water jug problem** |

**Problem Statement**: Consider the following problem: A Water Jug Problem: You are given two jugs, a 4-gallon one and a 3-gallon one, a pump which has unlimited water which you can use to fill the jug, and the ground on which water may be poured. Neither jug has any measuring markings on it. How can you get exactly 2 gallons of water in the 4-gallon jug?

A Water Jug Problem: You are given two jugs, a 4-gallon one and a 3-gallon one, a pump which has unlimited water which you can use to fill the jug, and the ground on which water may be poured. Neither jug has any measuring markings on it. How can you get exactly 2 gallons of water in the 4-gallon jug? Here the initial state is (0, 0). The goal state is (2, n) for any value of n.

**Solution -**

x=int(input("enter the value of initial jug x: "))

y=int(input("enter the value of initial jug y: "))

xf=int(input("enter the value of final jug x: "))

yf=int(input("enter the value of final jug y: "))

print("list of many option\n");

print("r1 :fill 4 litre gallon in x")

print("r2 :fill 3  litre gallon in y")

print("r3 :empty  litre gallon in x")

print("r4 :empty  litre gallon in y")

print("r5 :take some water from y to x")

print("r6 :take some water from x to y")

print("r7 :complete from y to x")

print("r8 :complete from x to y")

while True:

    rule=int(input("enter rule no\n"))

    if(rule==1):

        if x<4:

            x=4

    if(rule==2):

        if y<3:

            y=3

    if(rule==3):

        if x>0:

            x=0

    if(rule==4):

        if y>0:

            y=0

    if (rule==5):

        if (x+y)>=4 and y>0 :

             x=4

             y=y-(x-3)

    if (rule==6):

        if (x+y)>=3 and x>0 :

             x=x-(3-y)

             y=3

    if (rule==7):

        if (x+y)<=4 and y>0 :

             x=x+y

             y=0

    if (rule==8):

        if (x+y)<=3 and x>0 :

             x=0

             y=x+y

    if x+y<=yf and x>0:

        x=0

        y=x+y

    print("X=",x)

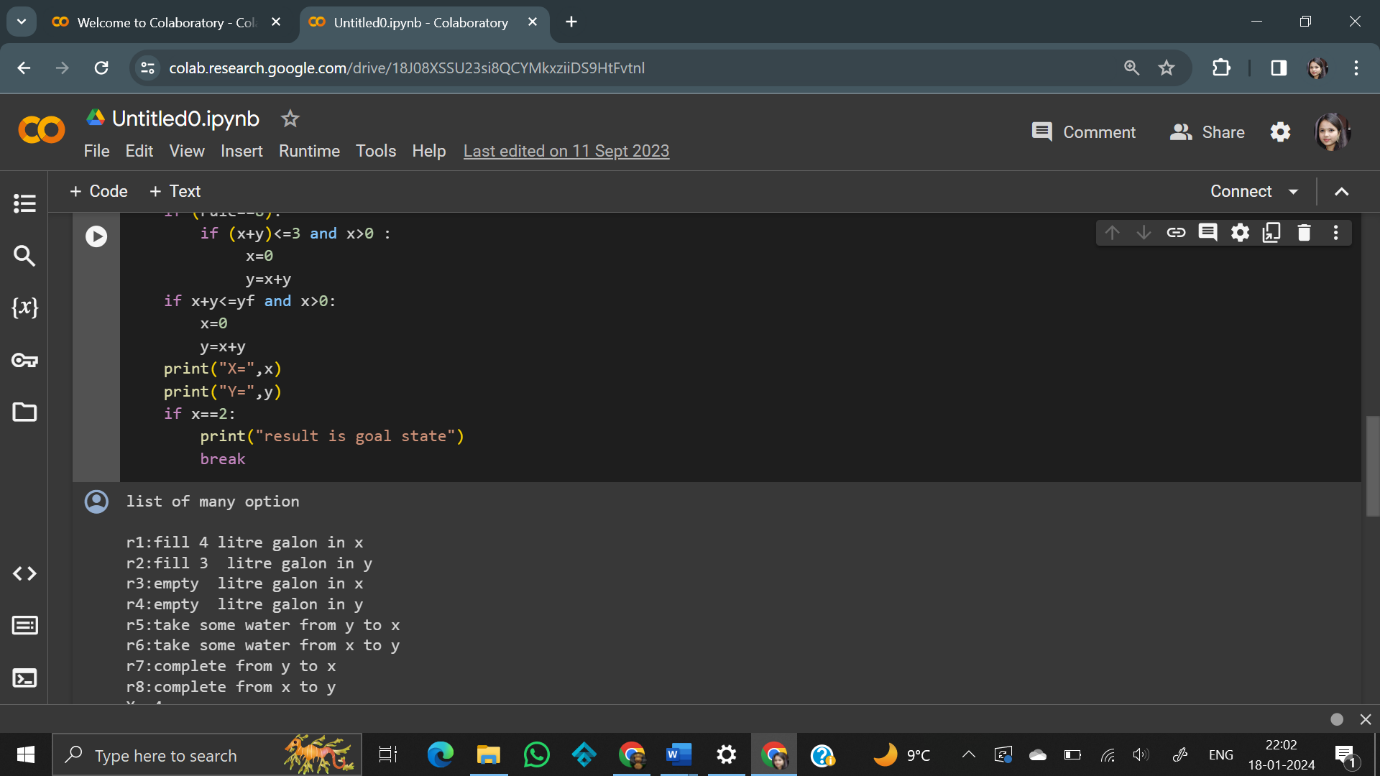
    print("Y=",y)

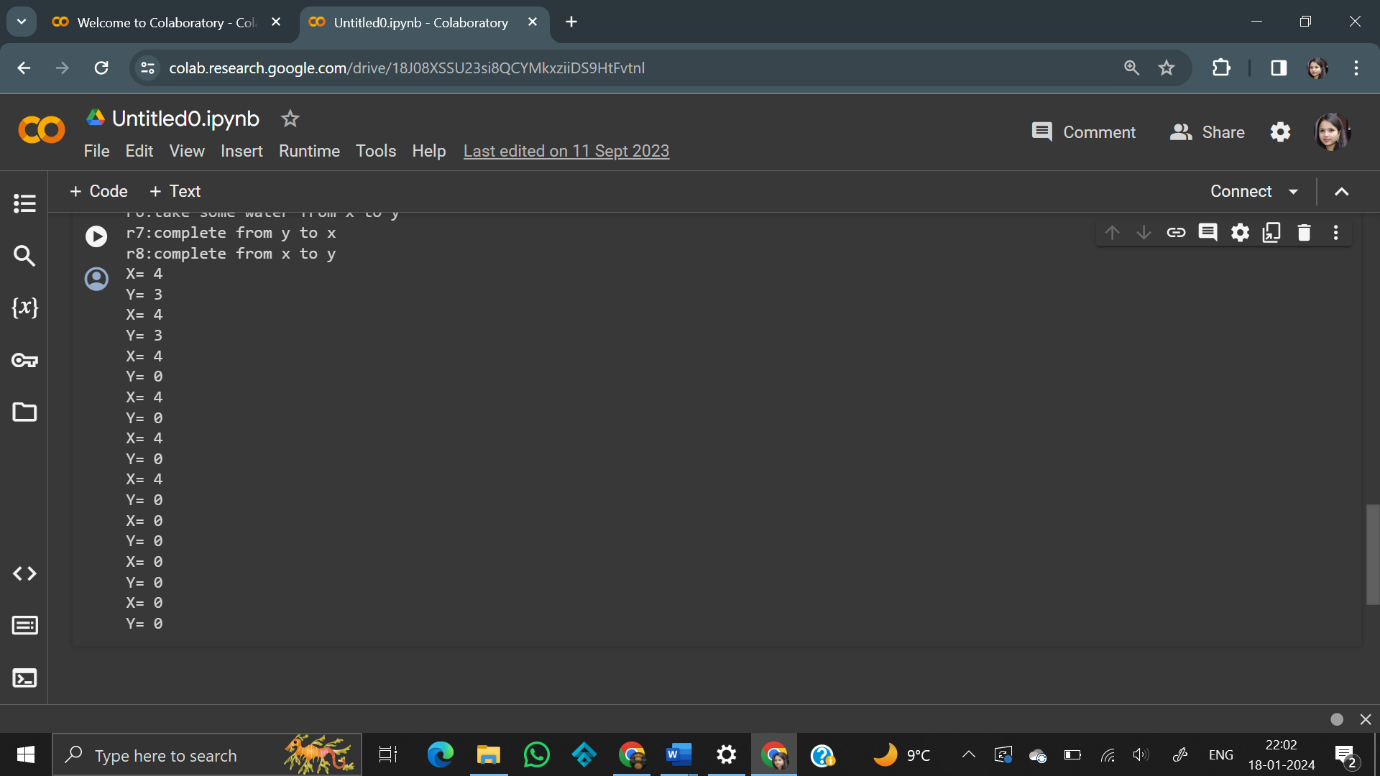
    if x==2:

        print("result is goal state")

        break

OUTPUT –





|  |  |
| --- | --- |
| **EX NO-3** | **Write a program to implement missionary and cannibal algorithm** |

Missionaries and Cannibals is a problem in which 3 missionaries and 3 cannibals want to cross from the left bank of a river to the right bank of the river. There is a boat on the left bank, but it only carries at most two people at a time (and can never cross with zero people). If cannibals ever outnumber missionaries on either bank, the cannibals will eat the missionaries. A state can be represented by a triple, (m c b), where m is the number of missionaries on the left, c is the number of cannibals on the left, and b indicates whether the boat is on the left bank or right bank. For example, the initial state is (3 3 L) and the goal state is (3 3 R)

**Solution -**

M1=3

C1=3

M2=0

C2=0

print("Display the all rule")

print("Rule1:One missionary is selling boat from bank 1 to bank 2")

print("Rule 2:Two missionary is selling boat from bank 1 to bank 2")

print("Rule 3:One missionary and one Cannibal is selling boat from bank 1 to bank 2")

print("Rule 4:One cannibal is selling boat from bank 1 to bank 2")

print("Rule 5:Two cannibal are selling boat from bank 1 to bank 2")

print("Rule 6:One missionary is selling boat from bank 2 to bank 1")

print("Rule 7:Two missionary is selling boat from bank 2 to bank 1")

print("Rule 8:One missionary and one Cannibal is selling boat from bank 2 to bank 1")

print("Rule 9:One cannibal is selling boat from bank 2 to bank 1")

print("Rule 10:Two cannibal are selling boat from bank 2 to bank 1")

print("initial state=(3M,3C,Bank-1)")

print("Goal state=(3M ,3C,bank-2)")

while((M2!=3) or (C2!=3)):

    r=int(input("Enter rule"))

    if(r==1):

      M1-=1

      M2+=1

    elif(r==2):

      M1-=2

      M2+=2

    elif(r==3):

      M1-=1

      C1-=1

      M2+=1

      C2+=1

    elif(r==4):

      C1-=1

      C2+=1

    elif(r==5):

      C1-=2

      C2+=2

    elif(r==6):

      M1+=1

      M2-=1

    elif(r==7):

      M1+=2

      M2-=2

    elif(r==8):

      M1+=1

      C1+=1

      M2-=1

      C2-=1

    elif(r==9):

      C1+=1

      C2-=1

    elif(r==10):

      C1+=2

      C2-=2

    print(M1,C1)

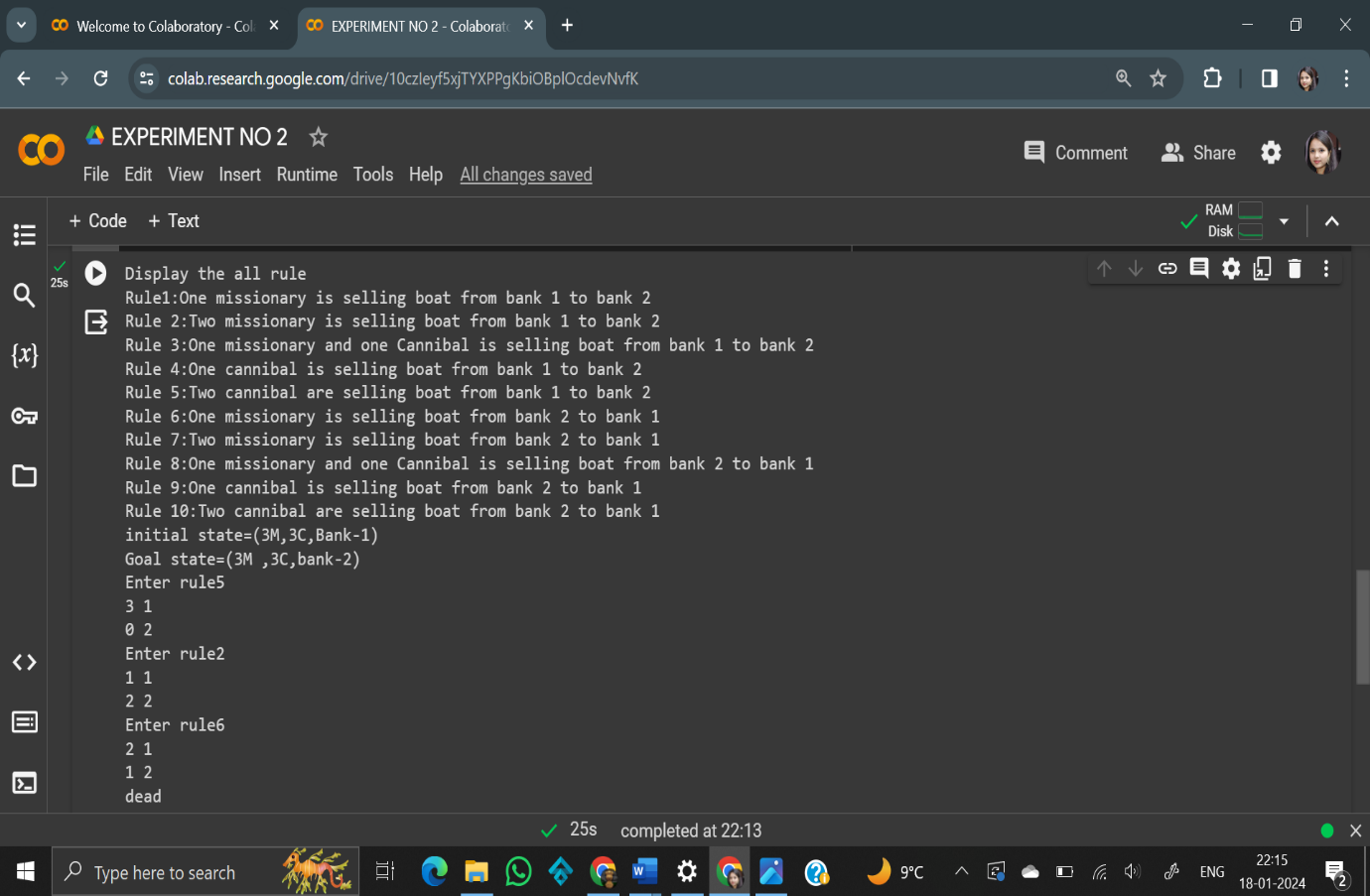
    print(M2,C2)

    if((M1>0 and M1<C1) or (M2>0 and M2<C2)):

      print("dead")

      break

**Output -**



|  |  |
| --- | --- |
| **EX NO -4** | **Write a program to implement breadth first search** |

Breadth-First Search (BFS) is a popular graph traversal algorithm used to explore and analyze graphs and trees. It operates in a way that explores all the vertices (nodes) at the current depth level before moving on to the next level. BFS is typically implemented using a queue data structure. Here is a step-by-step algorithm for BFS:

**Solution -**

graph = {

    'A' : ['B' , 'C'],

    'B' : ['D' , 'E'],

    'C' : ['D' , 'F'],

    'D' : ['G' , 'B'],

    'E' : ['B' , 'C'],

    'F' : ['C' , 'H'],

    'G' : ['I'] ,

    'H' : ['K'] ,

    'K' : ['L'] ,

    'L' : ['M'] ,

    'M' : ['N'] ,

    'I' : ['J'] ,

    'N' : [ ] ,

    'J' : [ ]

}

goal = ['J' , 'N']

visited=[]

queue=[]

def bfs(visited ,graph ,node):

  visited.append(node)

  queue.append(node)

  while queue :

    s=queue.pop(0)

    print( s,end=" ")

    if s in goal :

      return

    for nbd in graph[s]:

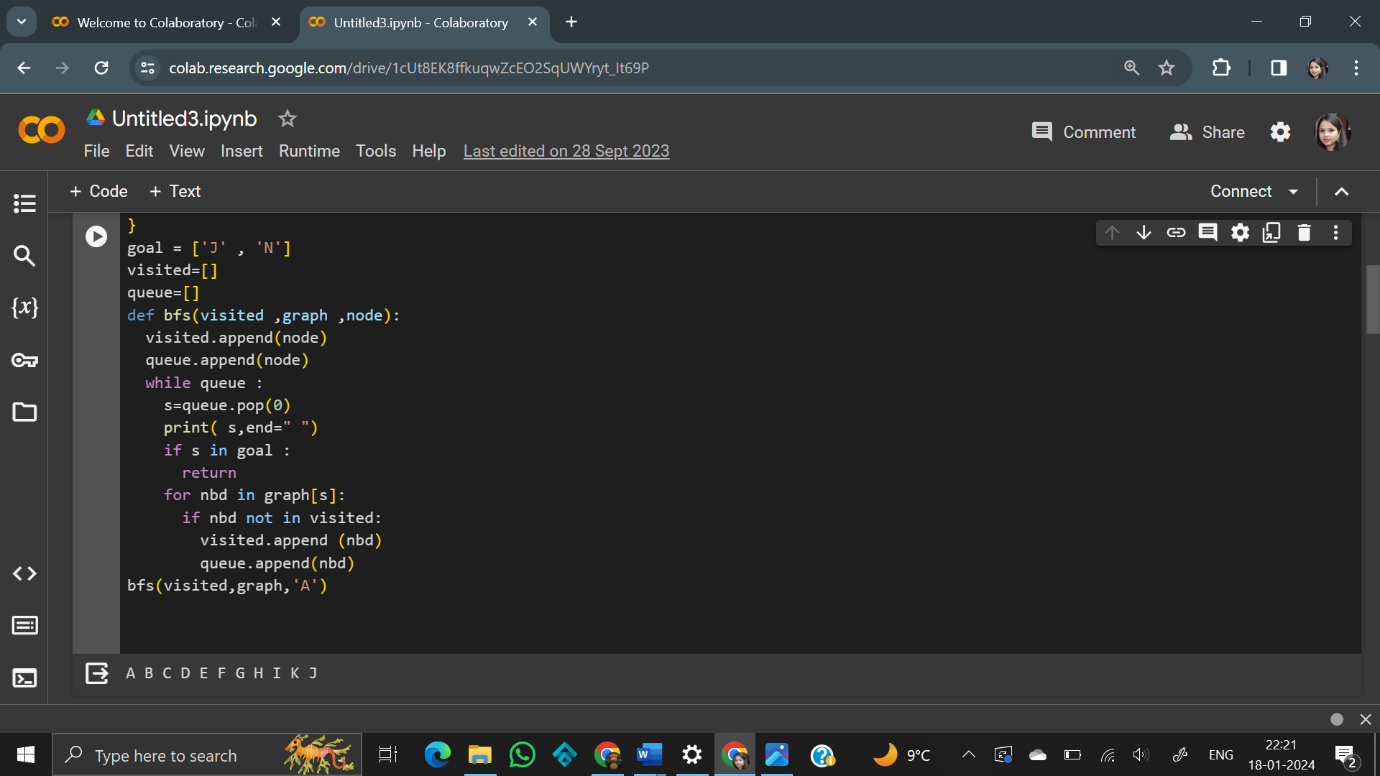
      if nbd not in visited:

        visited.append (nbd)

        queue.append(nbd)

bfs(visited,graph,'A')

**Output -**



|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | **EX NO-5** | **Write a program to implement depth first search** | |

Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking.

**Solution -**

graph = {

    'A' : ['B' , 'C'],

    'B' : ['D' , 'E'],

    'C' : ['D' , 'F'],

    'D' : ['G' , 'B'],

    'E' : ['B' , 'C'],

    'F' : ['C' , 'H'],

    'G' : ['I'] ,

    'H' : ['K'] ,

    'K' : ['L'] ,

    'L' : ['M'] ,

    'M' : ['N'] ,

    'I' : ['J'] ,

    'N' : [ ] ,

    'J' : [ ]

}

goal = ['J' , 'N']

visited =set()

stack=[]

def dfs(visited ,graph ,node) :

  visited.add(node)

  stack.append(node)

  while stack:

    curr\_node=stack.pop(-1)

    print(curr\_node,end=" ")

    if curr\_node in goal:

      return

    for nbd in reversed(graph[curr\_node]):

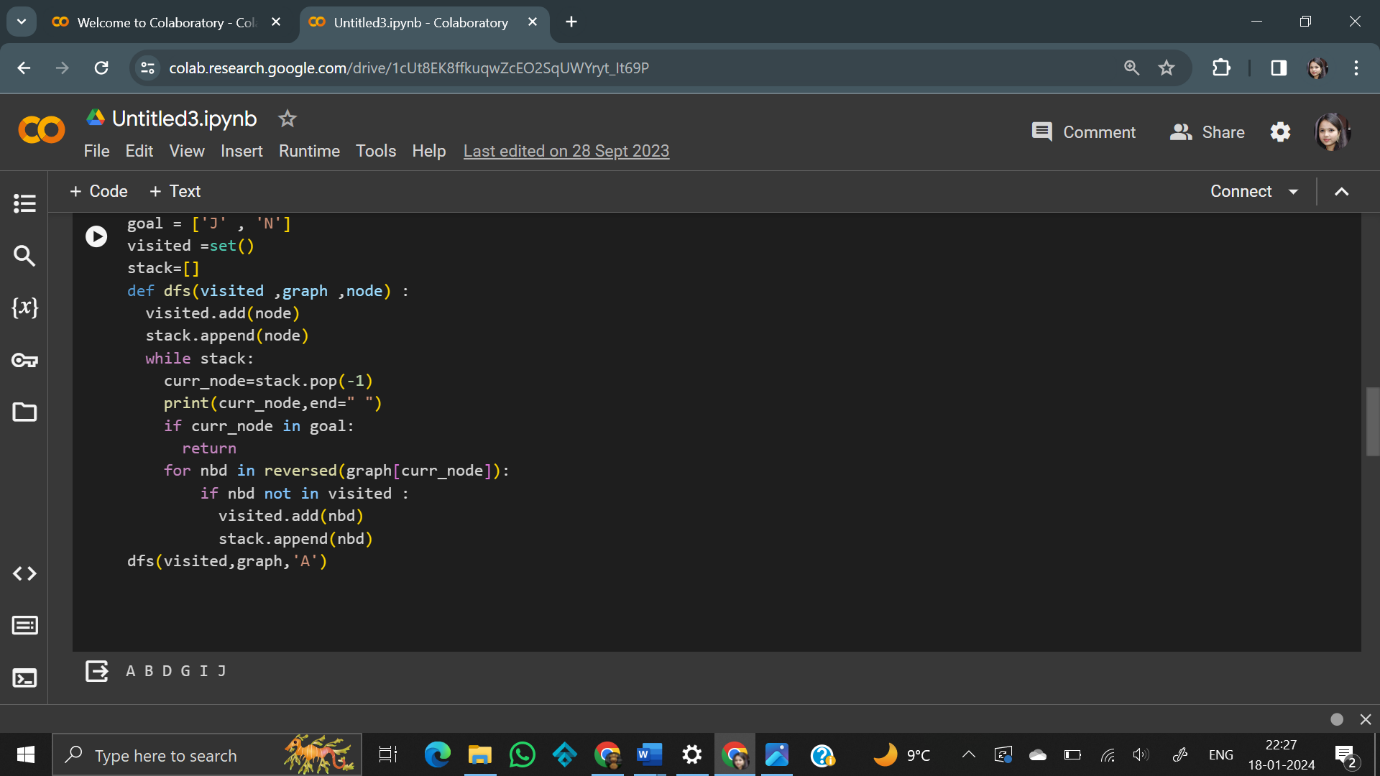
        if nbd not in visited :

          visited.add(nbd)

          stack.append(nbd)

dfs(visited,graph,'A')

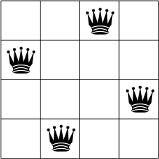
**Output -**



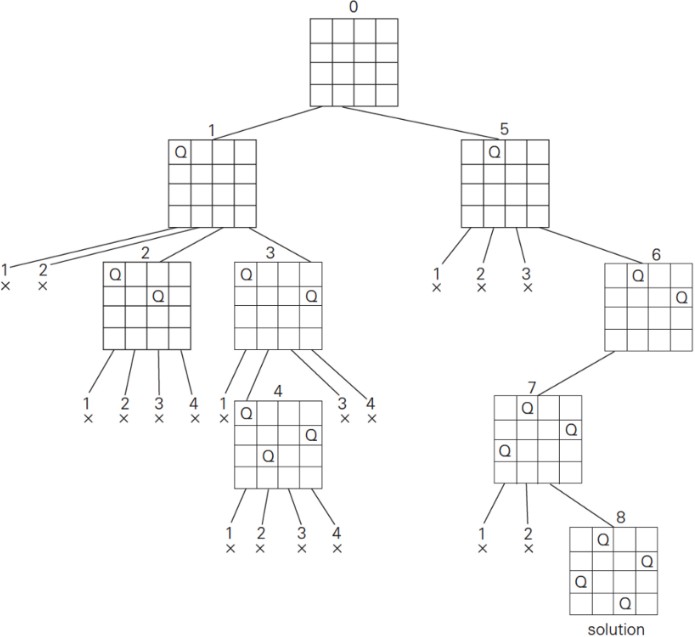
|  |  |
| --- | --- |
| **EX NO -6** | **Write a program to implement 4/8 Queen Problems** |

The **4-Queens Problem** consists in placing four queens on a 4 x 4 chessboard so that no two queens can capture each other. That is, no two queens are allowed to be placed on the same row, the same column or the same diagonal.

The following figure illustrates a solution to the 4-Queens Problem: none of the 4 queens can capture each other.



Although this particular problem isn’t very impressive, keep in mind that you can generalize it to chessboards with.



**Solution –**

graph={

    'A' : ['B' , 'C' ,'D' ,'E'],

    'B' : ['F' , 'G'],

    'C' : ['H'],

    'D' : ['I'],

    'E' : ['J' , 'K'],

    'F' : [ ],

    'G' : ['L'],

    'H' : ['M'],

    'I' : ['N'],

    'J' : ['O'],

    'K' : [ ],

    'L' : [ ],

    'M' : ['P'],

    'N' : ['Q'],

    'O' : [ ],

    'P' : [ ],

    'Q' : [ ]

}

label={

    'A' : [0,0],

    'B' : [1,1],

    'C' : [1,2],

    'D' : [1,3],

    'E' : [1,4],

    'F' : [2,3],

    'G' : [2,4],

    'H' : [2,4],

    'I' : [2,1],

    'J' : [2,1],

    'K' : [2,2],

    'L' : [3,2],

    'M' : [2,1],

    'N' : [3,4],

    'O' : [3,3],

    'P' : [4,3],

    'Q' : [4,2]

}

goal=['P','Q']

visited=set()

stack=[]

def dfs(visited,graph,node):

  visited.add(node)

  stack.append(node)

  while stack:

    curr\_node=stack.pop(-1)

    print(curr\_node,label[curr\_node],end=" ")

    if curr\_node in goal:

     return

    for nbd in reversed(graph[curr\_node]):

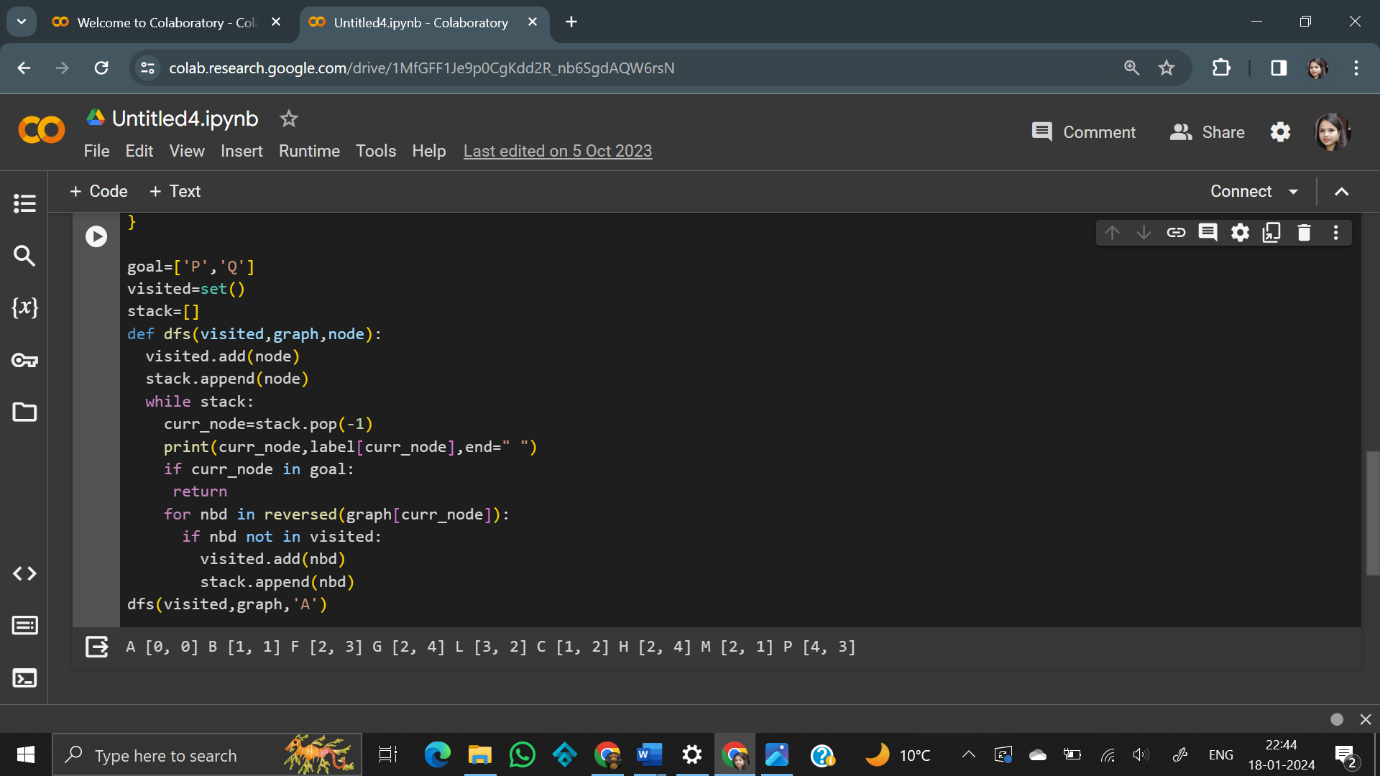
      if nbd not in visited:

        visited.add(nbd)

        stack.append(nbd)

dfs(visited,graph,'A')

**Output -**



|  |  |
| --- | --- |
| **EX NO- 7** | **Write a program to implement first order logic** |

Python library that enables using logic programming in python. The aim of the library is to explore ways to use symbolic reasoning with machine learning. Pytholog supports probabilities. Pytholog gives facts indices (first term) and uses binary search to search for relevant facts instead of looping over all knowledge base. So when defining rules, make sure that the main search terms are in the first position to speed up the search queries

**Solution -**

pip install pytholog

import pytholog as pl

kb4 = pl.KnowledgeBase("flavor")

kb4(["likes(divyanshu,samosa)",

        "likes(tushar,cookies)",

        "likes(riya,pasta)",

        "likes(ashish,samosa)",

        "likes(arpita,lemonade)",

        "food\_type(samosa,snack)",

        "food\_type(pasta,noodles)",

        "food\_type(cookies,biscuit)",

        "food\_type(lemonade,juice)",

        "food\_type(lemonade,drink)",

        "flavor(sweet,juice)",

        "flavor(savory, snack)",

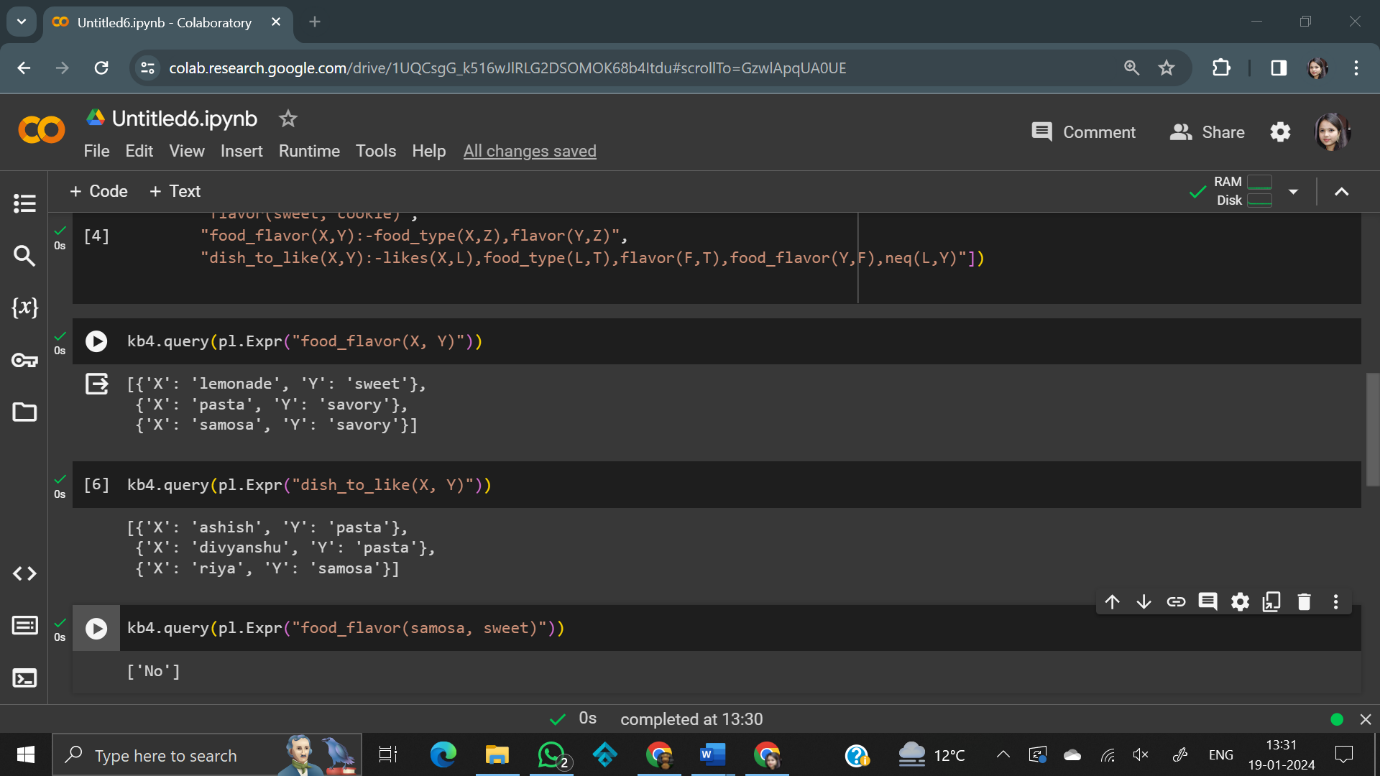
        "flavor(savory, noodles)",

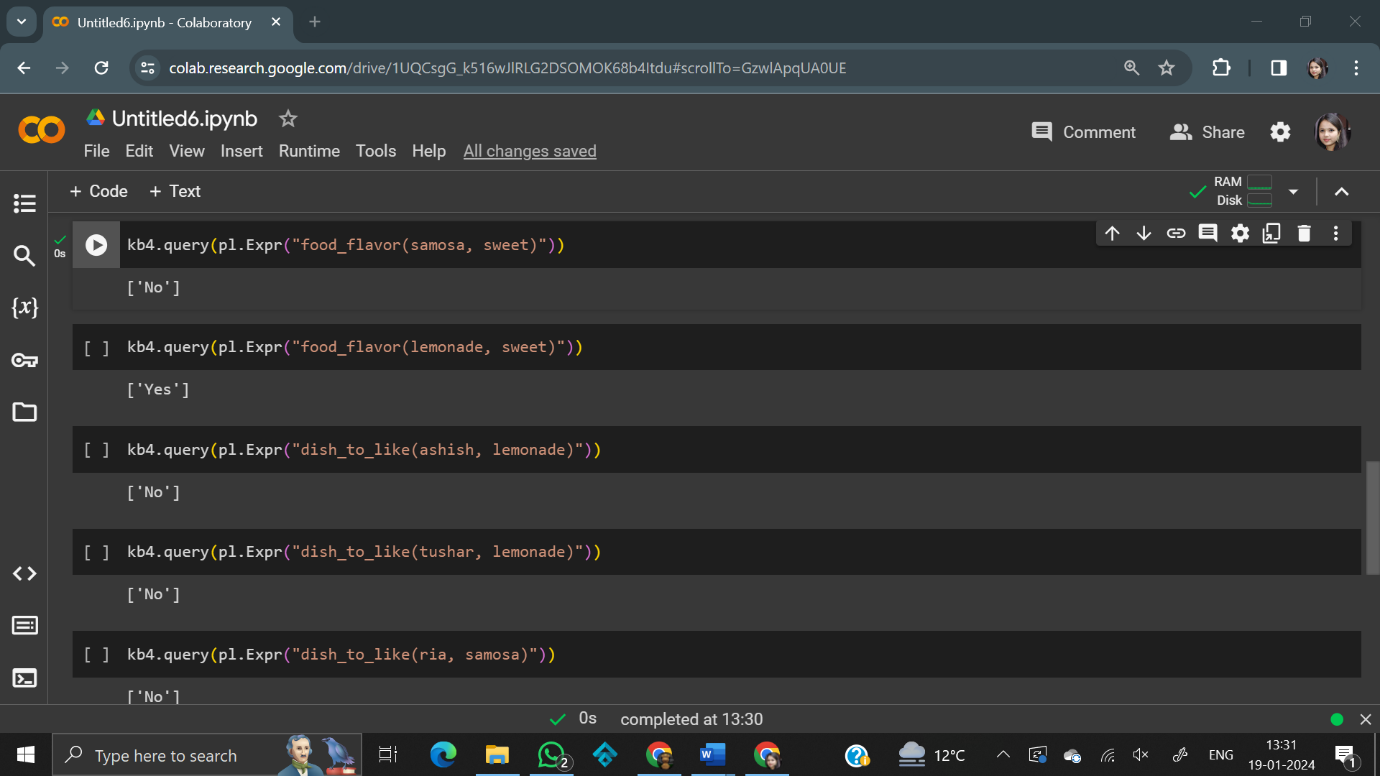
        "flavor(sweet, cookie)",

        "food\_flavor(X,Y):-food\_type(X,Z),flavor(Y,Z)",

        "dish\_to\_like(X,Y):-likes(X,L),food\_type(L,T),flavor(F,T),food\_flavor(Y,F),neq(L,Y)"])

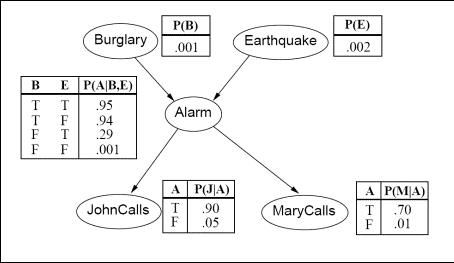
**Output –**





|  |  |
| --- | --- |
| **EX NO- 8** | **Write a program to demonstrate the working of Bayesian network.** |

Write a program to demonstrate the working of Bayesian network for the following graph:



* + Calculate the probability of a burglary if John and Mary calls (0: True, 1: False)
  + Calculate the probability of alarm starting if there is a burglary and an earthquake (0: True, 1: False)

Calculate the probability of alarm starting if there is a burglary and an earthquake (0: True, 1: False)

**Solution -**

pip install pgmpy

import pgmpy .  inference

import pgmpy . models

import networkx as nx

import pylab as plt

model = pgmpy.models.BayesianNetwork([('Burglary', 'Alarm'), ('Earthquake', 'Alarm'), ('Alarm', 'JohnCalls'), ('Alarm', 'MaryCalls')])

cpd\_burglary = pgmpy.factors.discrete.TabularCPD("Burglary", 2, [[0.001], [0.999]])

cpd\_earthquake = pgmpy.factors.discrete.TabularCPD("Earthquake", 2, [[0.002], [0.998]])

cpd\_alarm = pgmpy.factors.discrete.TabularCPD('Alarm', 2, [[0.95, 0.94, 0.29, 0.001], [0.05, 0.06, 0.71, 0.999]], evidence=['Burglary', 'Earthquake'], evidence\_card=[2, 2])

cpd\_john = pgmpy.factors.discrete.TabularCPD('JohnCalls', 2, [[0.90, 0.05], [0.10, 0.95]], evidence=['Alarm'], evidence\_card=[2])

cpd\_mary = pgmpy.factors.discrete.TabularCPD('MaryCalls', 2, [[0.70, 0.01], [0.30, 0.99]], evidence=['Alarm'], evidence\_card=[2])

model.add\_cpds(cpd\_burglary, cpd\_earthquake, cpd\_alarm, cpd\_john, cpd\_mary)

model.check\_model()

print('Probability distribution, P(Burglary)')

print(cpd\_burglary)

print()

print('Probability distribution, P(Earthquake)')

print(cpd\_earthquake)

print()

print('Joint probability distribution, P(Alarm | Burglary, Earthquake)')

print(cpd\_alarm)

print()

print('Joint probability distribution, P(JohnCalls | Alarm)')

print(cpd\_john)

print()

print('Joint probability distribution, P(MaryCalls | Alarm)')

print(cpd\_mary)

print()

infer = pgmpy.inference.VariableElimination(model)

posterior\_probability1 = infer.query(['Burglary'], evidence={'JohnCalls': 0, 'MaryCalls': 0})

print('Posterior probability of Burglary if JohnCalls (True) and MaryCalls (True)')

print(posterior\_probability1)

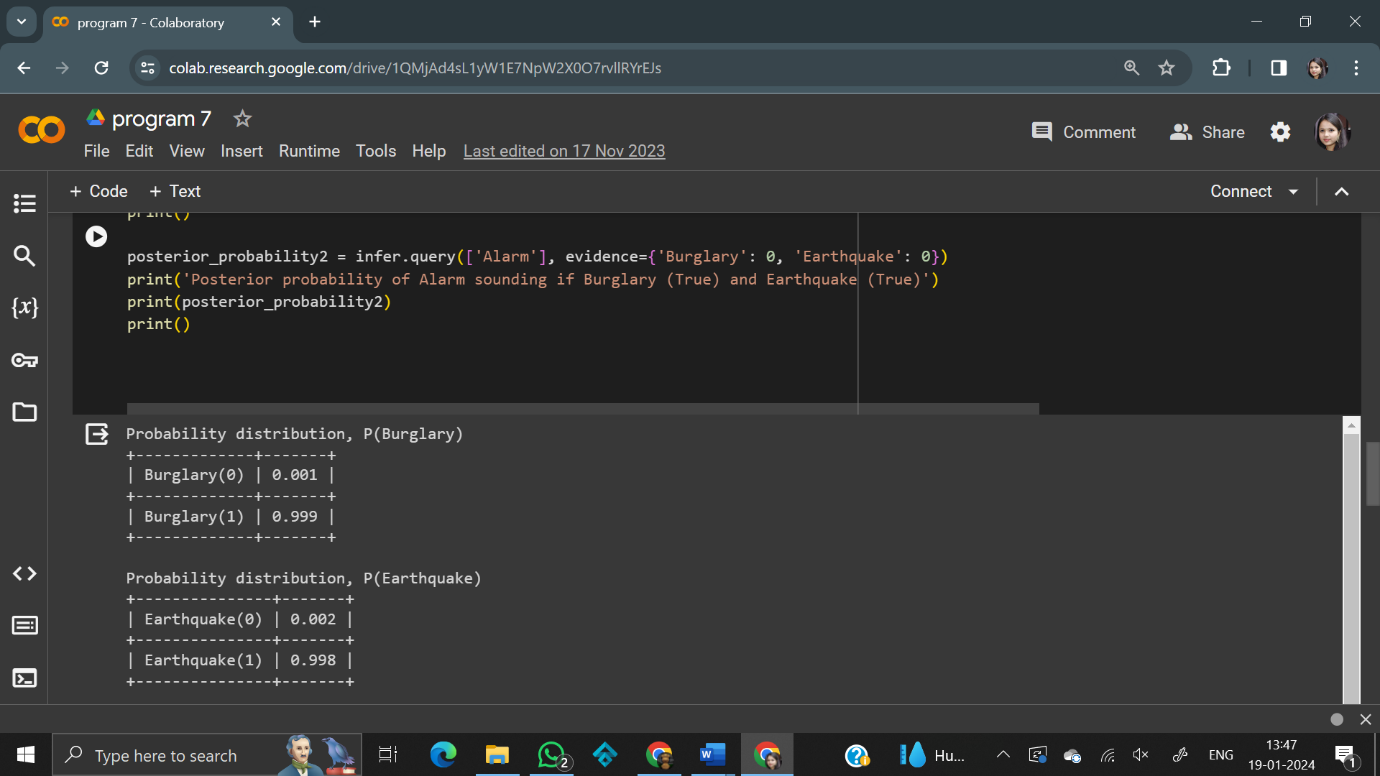
print()

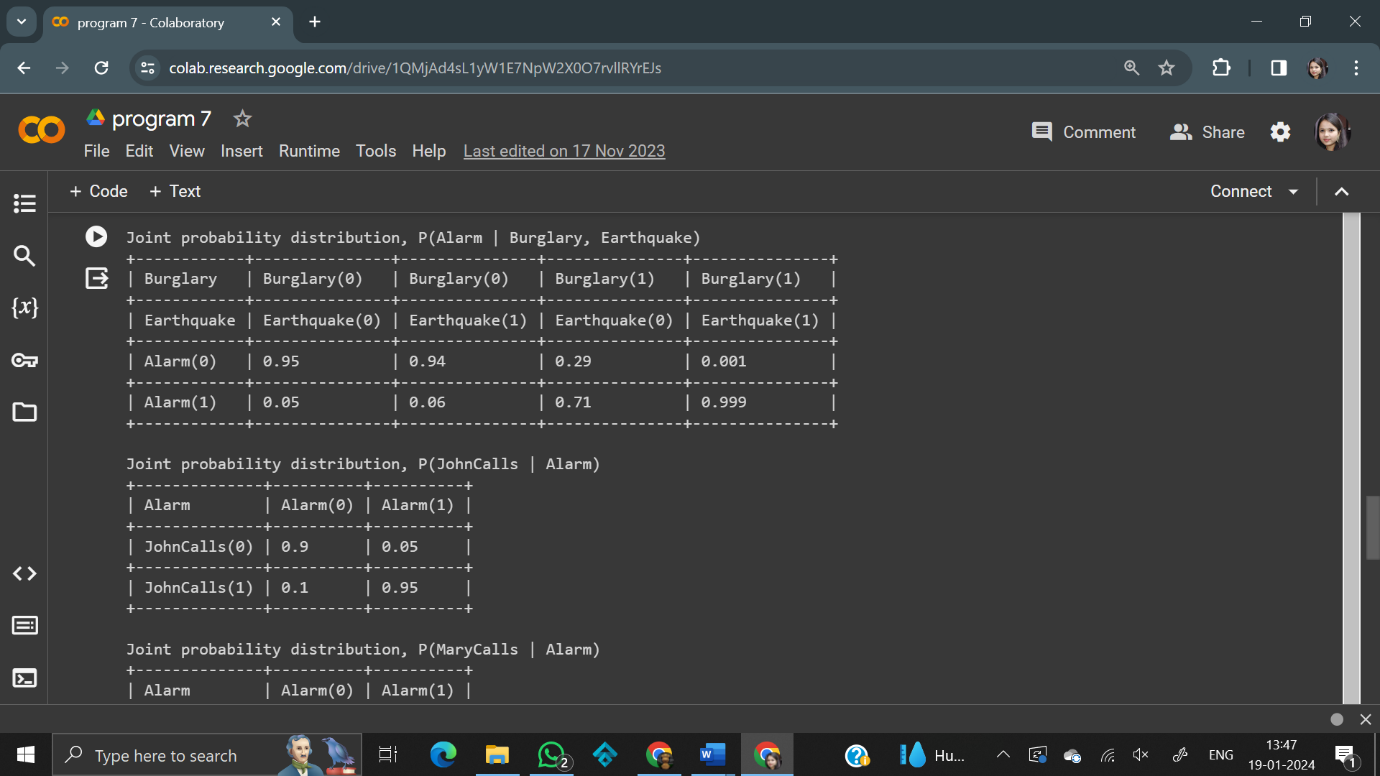
posterior\_probability2 = infer.query(['Alarm'], evidence={'Burglary': 0, 'Earthquake': 0})

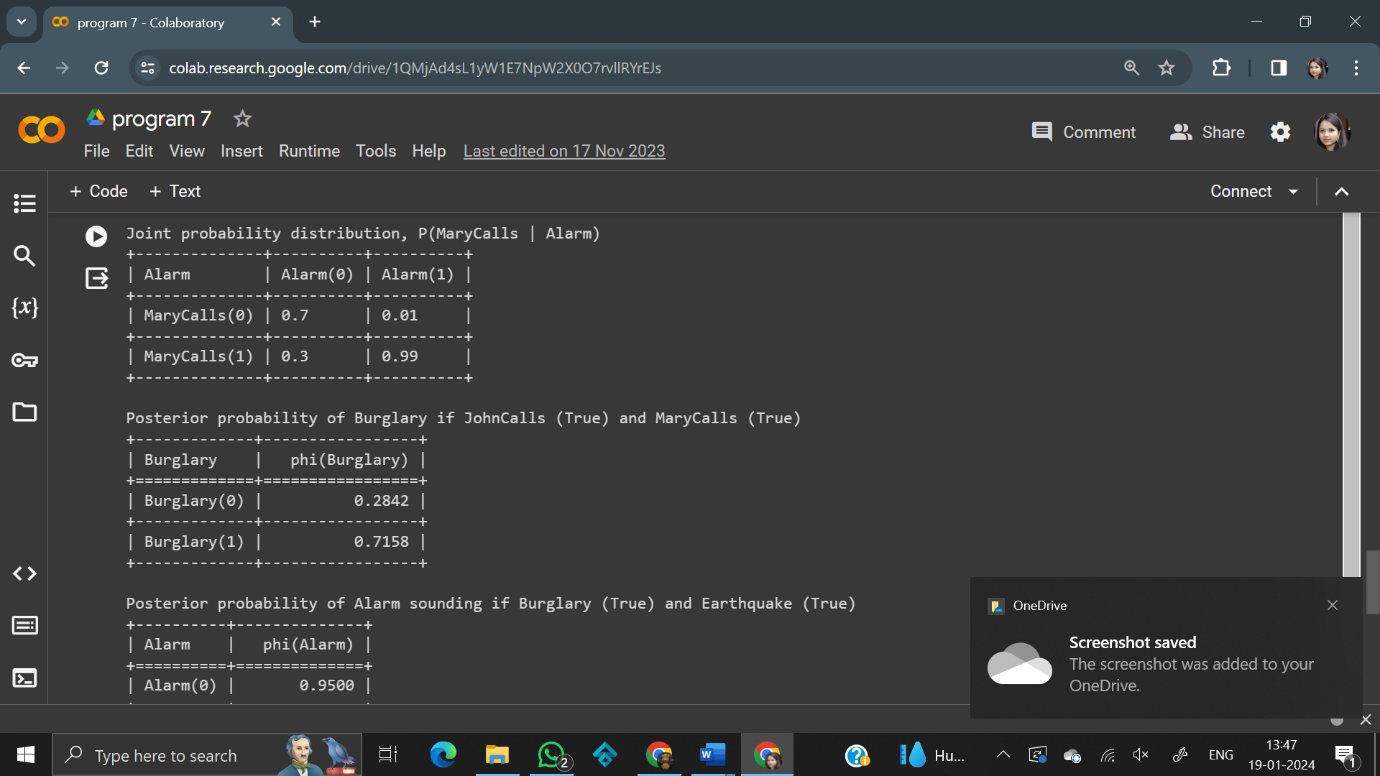
print('Posterior probability of Alarm sounding if Burglary (True) and Earthquake (True)')

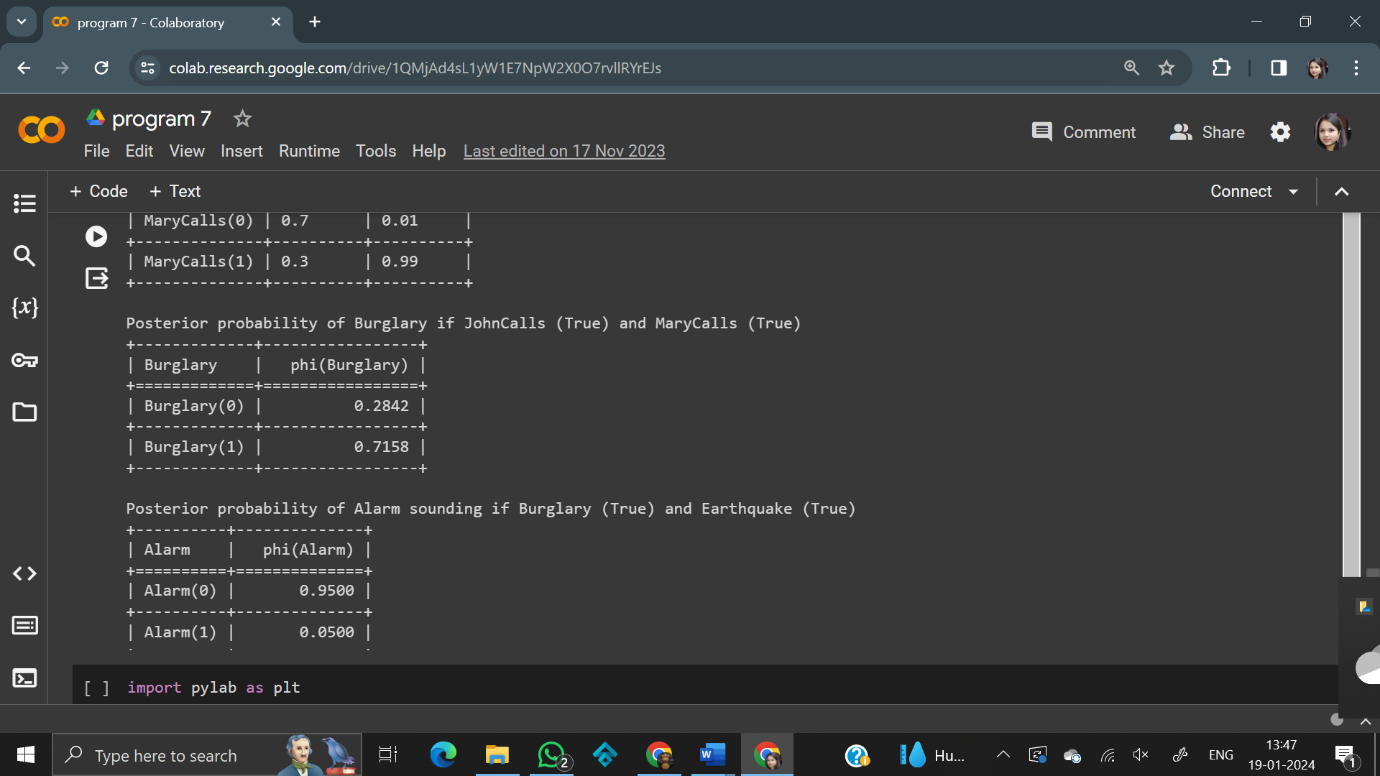
print(posterior\_probability2)

**Output -**









|  |  |
| --- | --- |
| **EX NO -9** | **Write a program to Implement pattern recognition problems of**  **handwritten character/ digit recognition** |

Write a program to Implement pattern recognition problems of handwritten digit recognition using dataset as <https://scikitlearn.org/stable/modules/generated/sklearn.datasets.load_digits.html>

import matplotlib.pyplot as plt

import numpy as np

# https:// scikit-Learn .org/stable/modules/generated/sklearn.datasets.load\_digits.html

from sklearn import  datasets

digits = datasets.load\_digits()

from sklearn .datasets import  load\_digits

digits = load\_digits ()

print (digits .data.shape)

(1797, 64)

CodeText

dir(digits)

['DESCR', 'data', 'feature\_names', 'frame', 'images', 'target', 'target\_names']

print (type (digits.images))

print (type (digits.target))

<class 'numpy.ndarray'>

<class 'numpy.ndarray'>

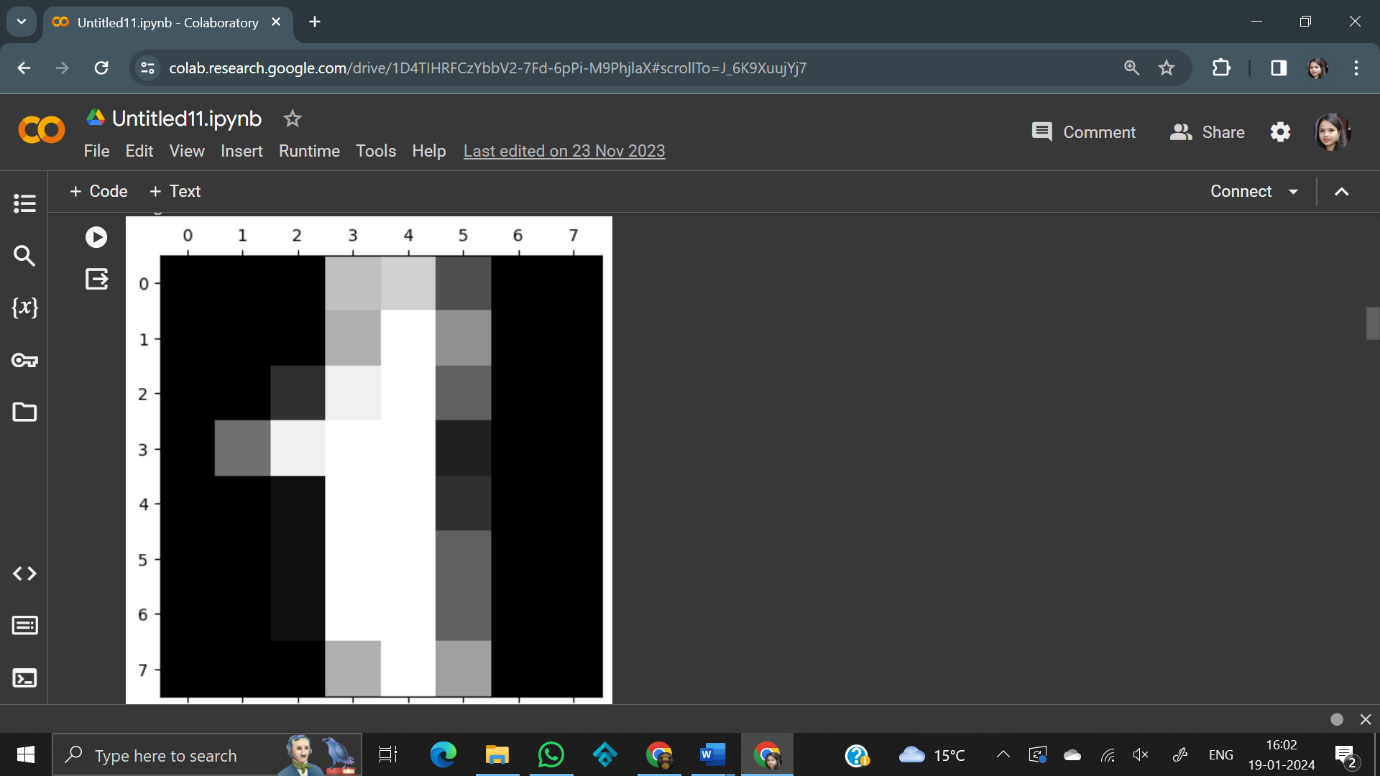
CodeText

print (digits.images.shape )

plt.gray()

plt.matshow(digits.images[1])

**Output -**



print (digits.images.shape )

plt.gray()

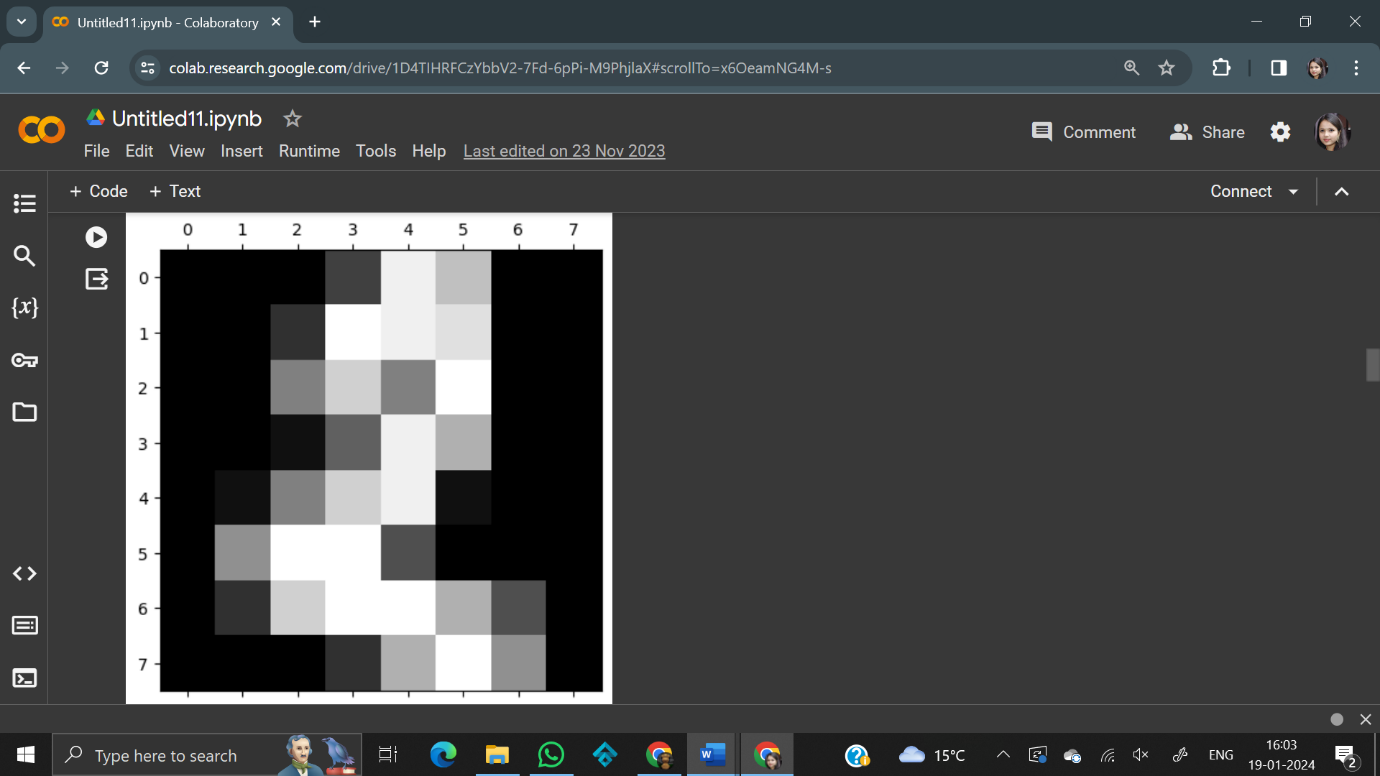
plt.matshow(digits.images[2])

(1797, 8, 8)

<matplotlib.image.AxesImage at 0x7ce8923b6b90>

<Figure size 640x480 with 0 Axes>

**Output -**



print (digits.images.shape )

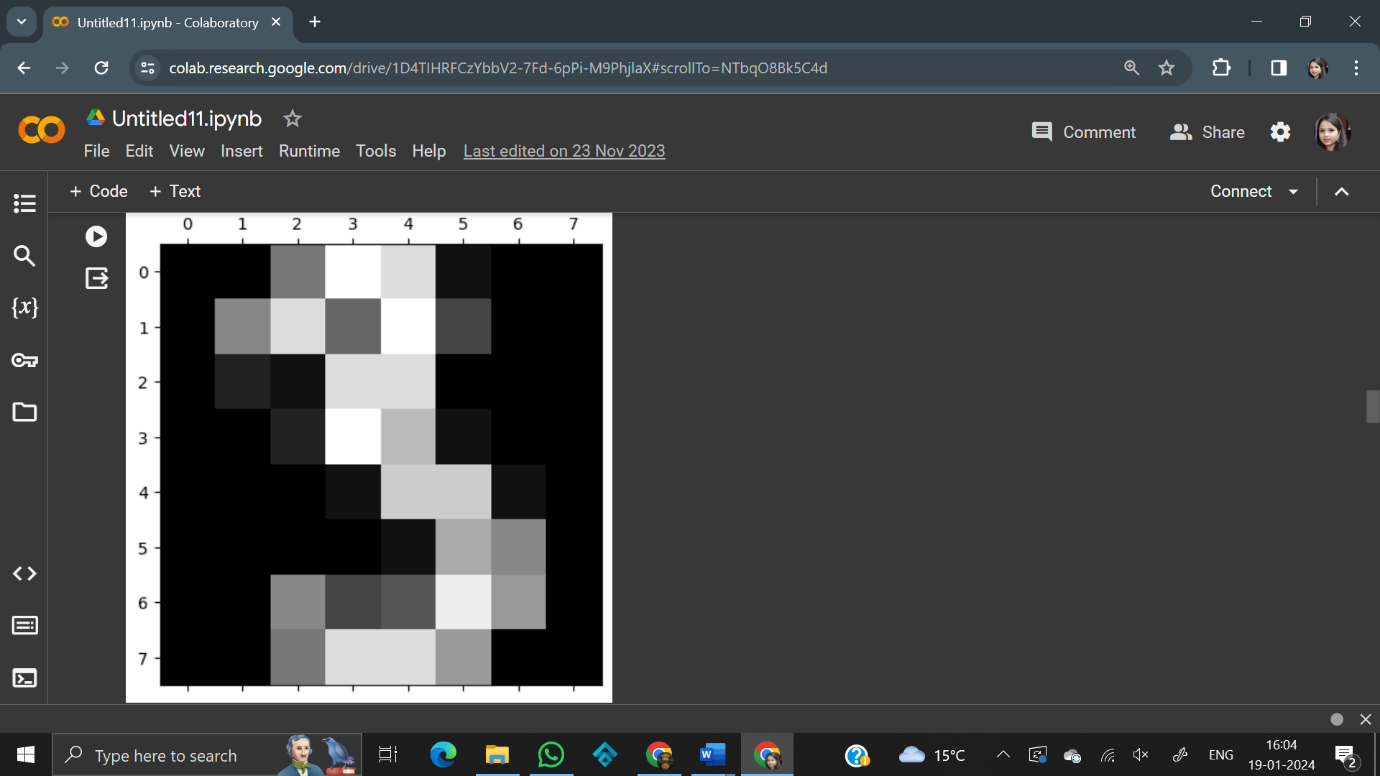
plt.gray()

plt.matshow(digits.images[3])

1797, 8, 8)

<matplotlib.image.AxesImage at 0x7ce892267370>

<Figure size 640x480 with 0 Axes>



print (digits.images.shape )

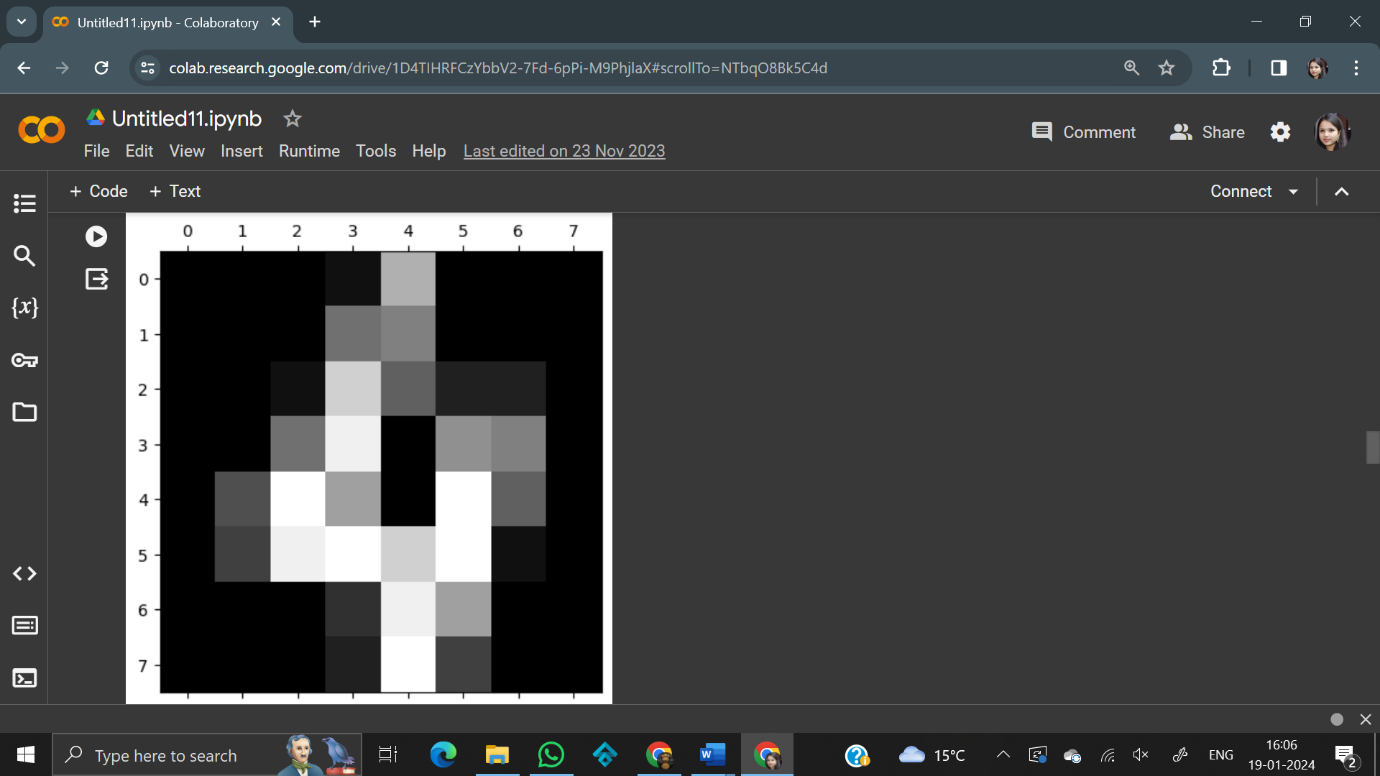
plt.gray()

plt.matshow(digits.images[4])

1797, 8, 8)

<matplotlib.image.AxesImage at 0x7ce892102740>

<Figure size 640x480 with 0 Axes>



print (digits.images.shape )

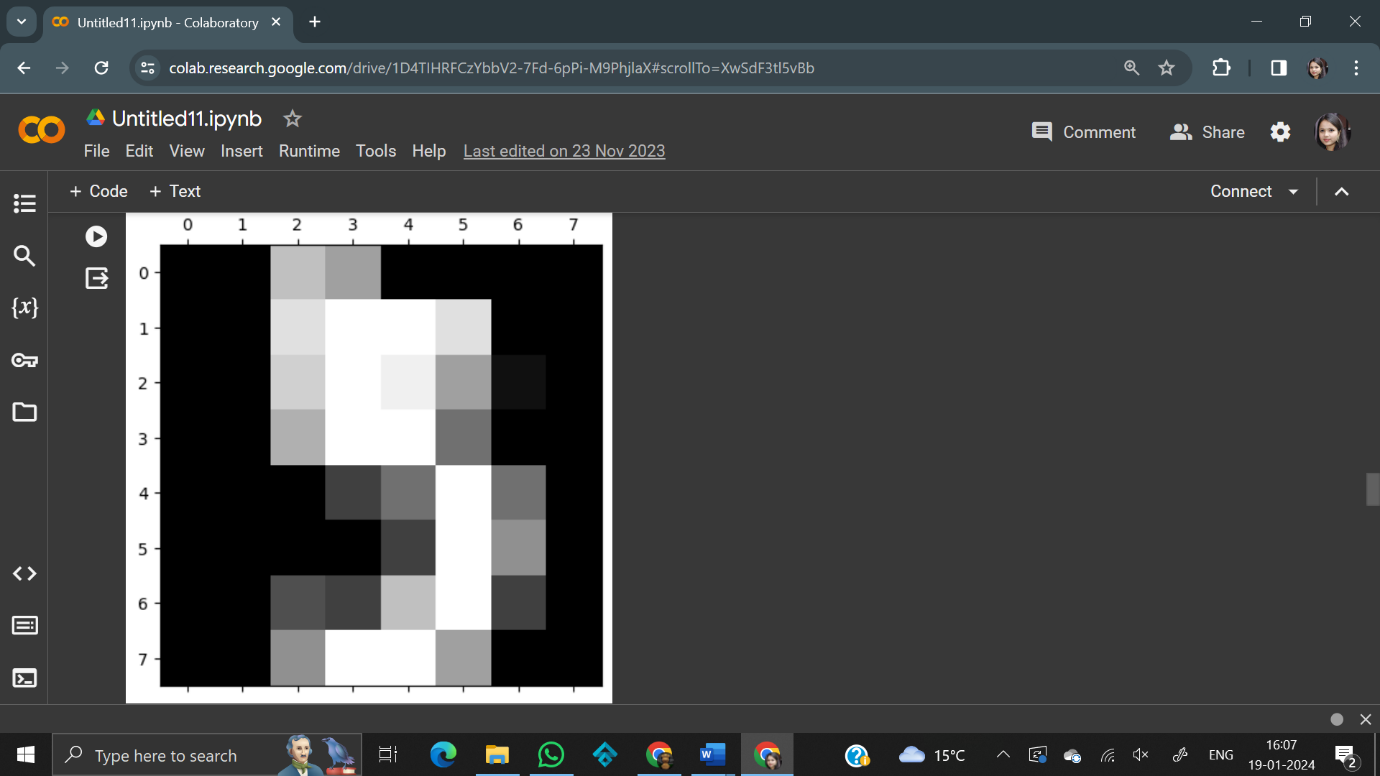
plt.gray()

plt.matshow(digits.images[5])

(1797, 8, 8)

<matplotlib.image.AxesImage at 0x7ce89219b130>

<Figure size 640x480 with 0 Axes>



Print (digits.images.shape )

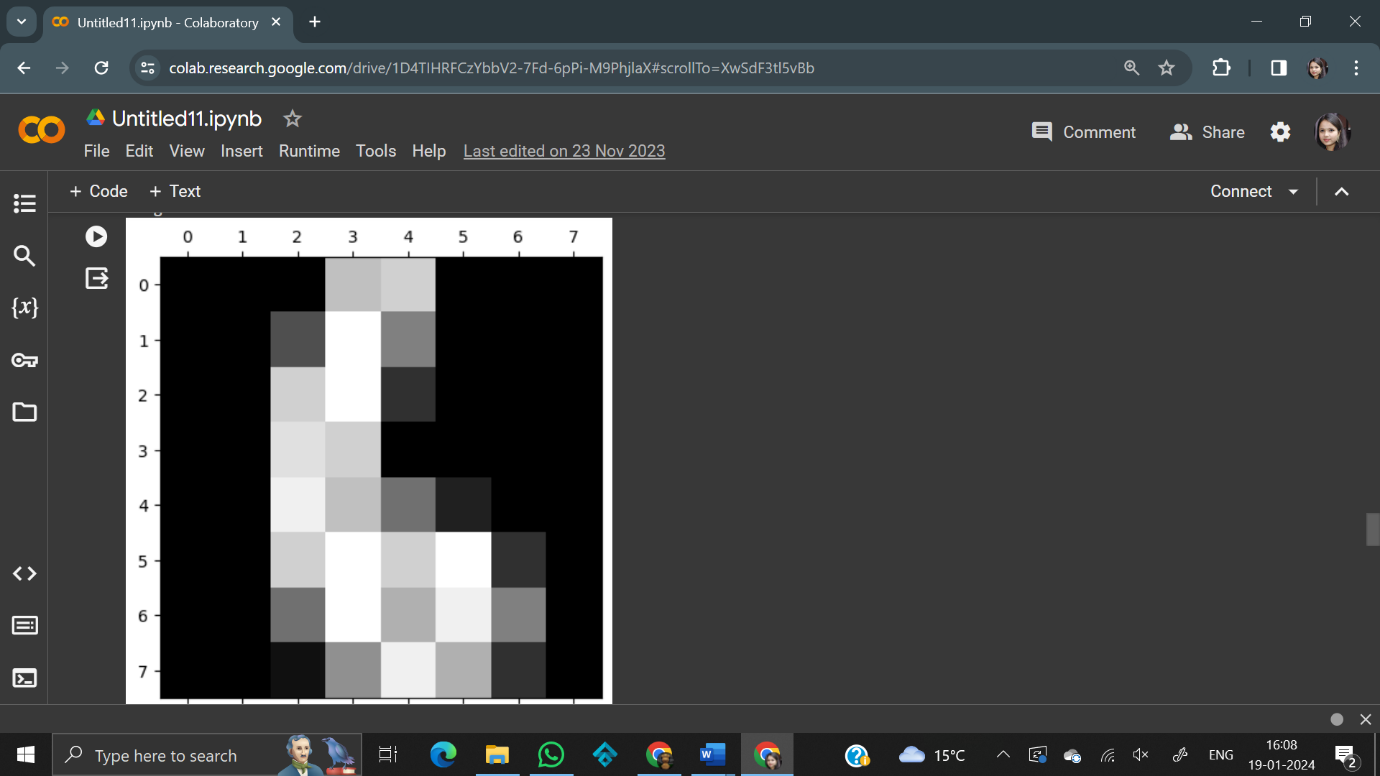
plt.gray()

plt.matshow(digits.images[6])

(1797, 8, 8)

<matplotlib.image.AxesImage at 0x7ce892026fe0>

<Figure size 640x480 with 0 Axes>



print (digits.images.shape )

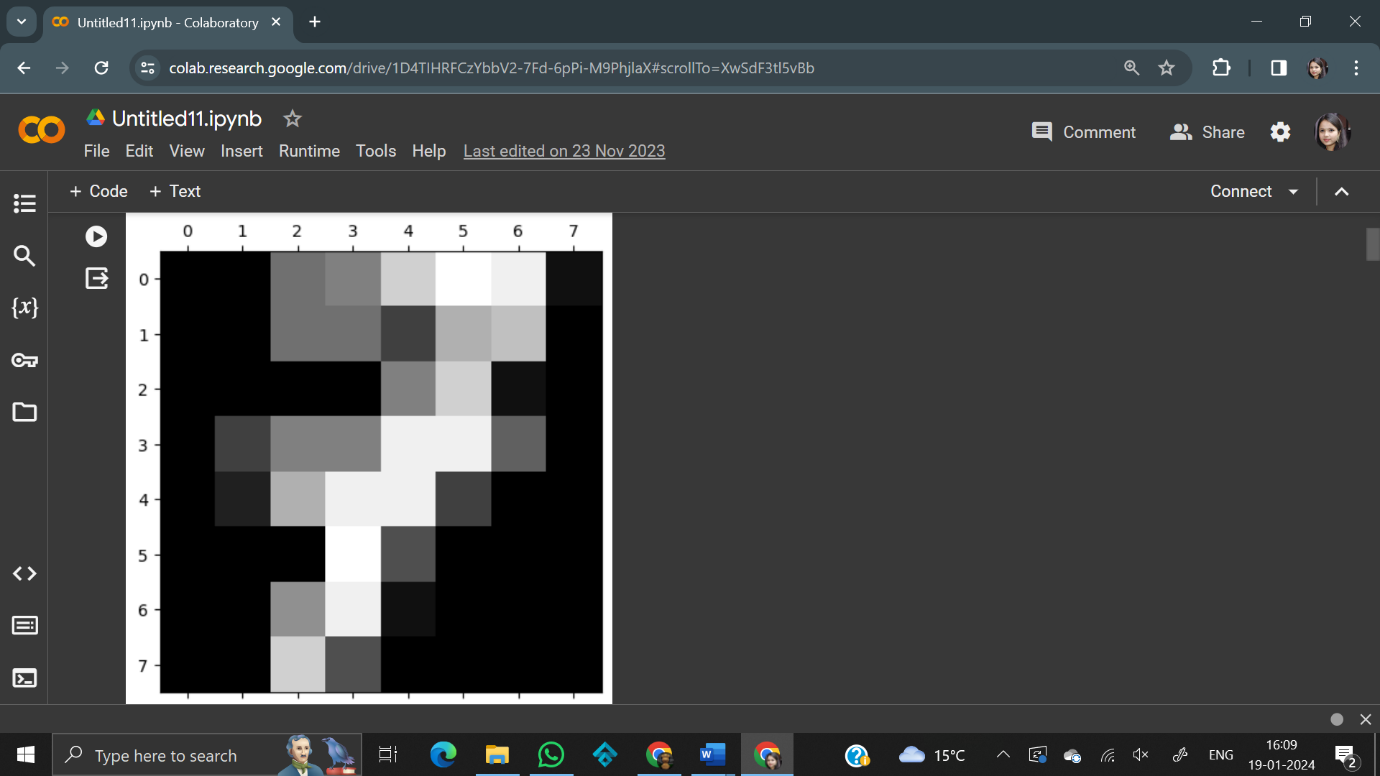
plt.gray()

plt.matshow(digits.images[7])

1797, 8, 8)

<matplotlib.image.AxesImage at 0x7ecb77166500>

<Figure size 640x480 with 0 Axes>



print (digits.images.shape )

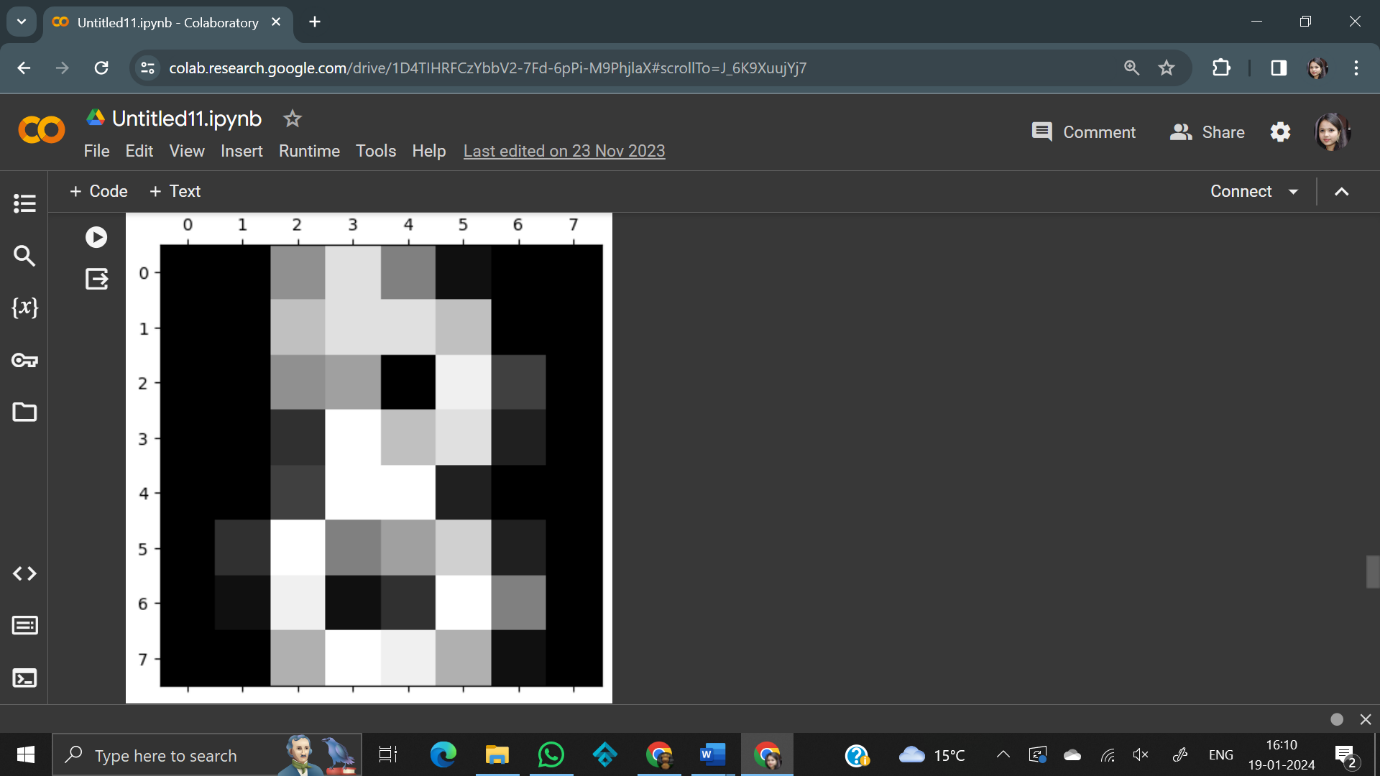
plt.gray()

plt.matshow(digits.images[8])

(1797, 8, 8)

<matplotlib.image.AxesImage at 0x7ce8920b64d0>

<Figure size 640x480 with 0 Axes>



print (digits.images.shape )

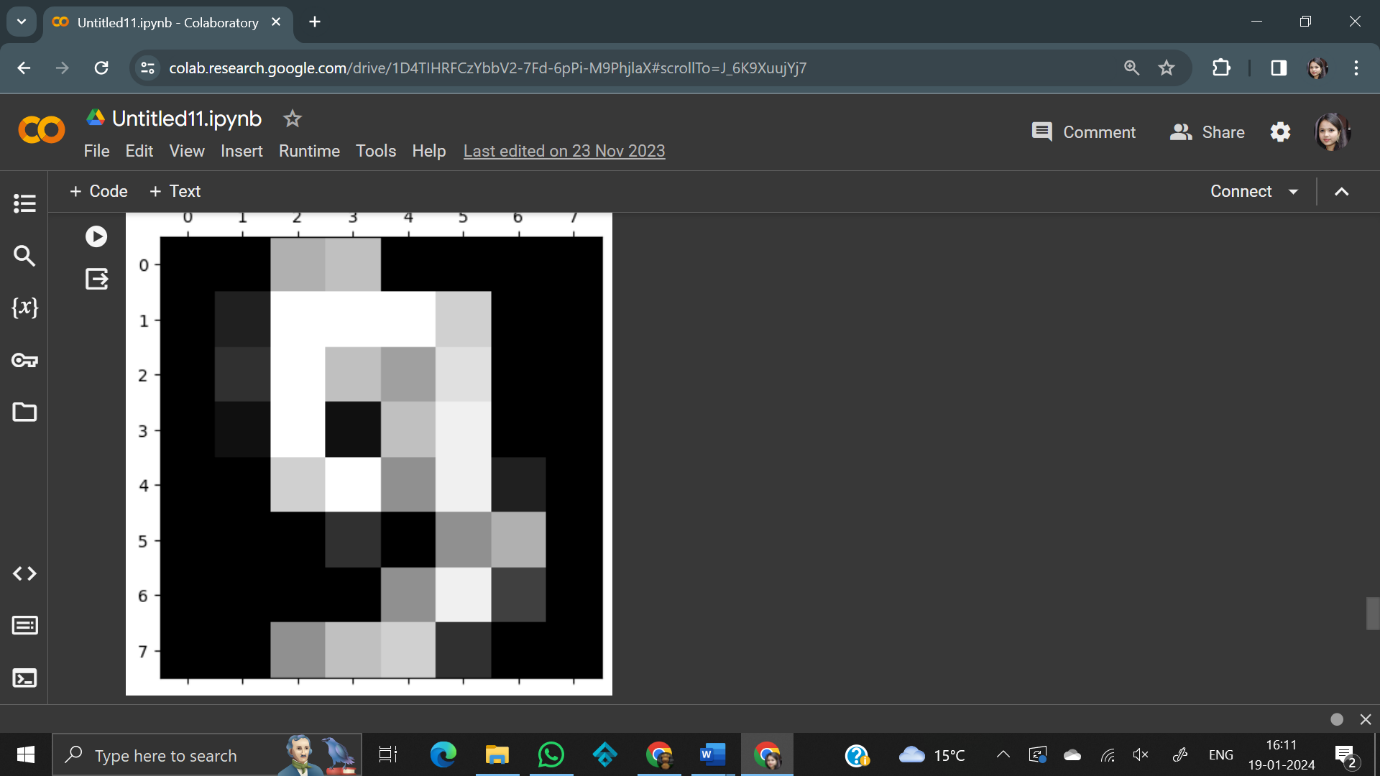
plt.gray()

plt.matshow(digits.images[9])

(1797, 8, 8)

<matplotlib.image.AxesImage at 0x7ce891f3ace0>

<Figure size 640x480 with 0 Axes>



print (digits.images.shape )

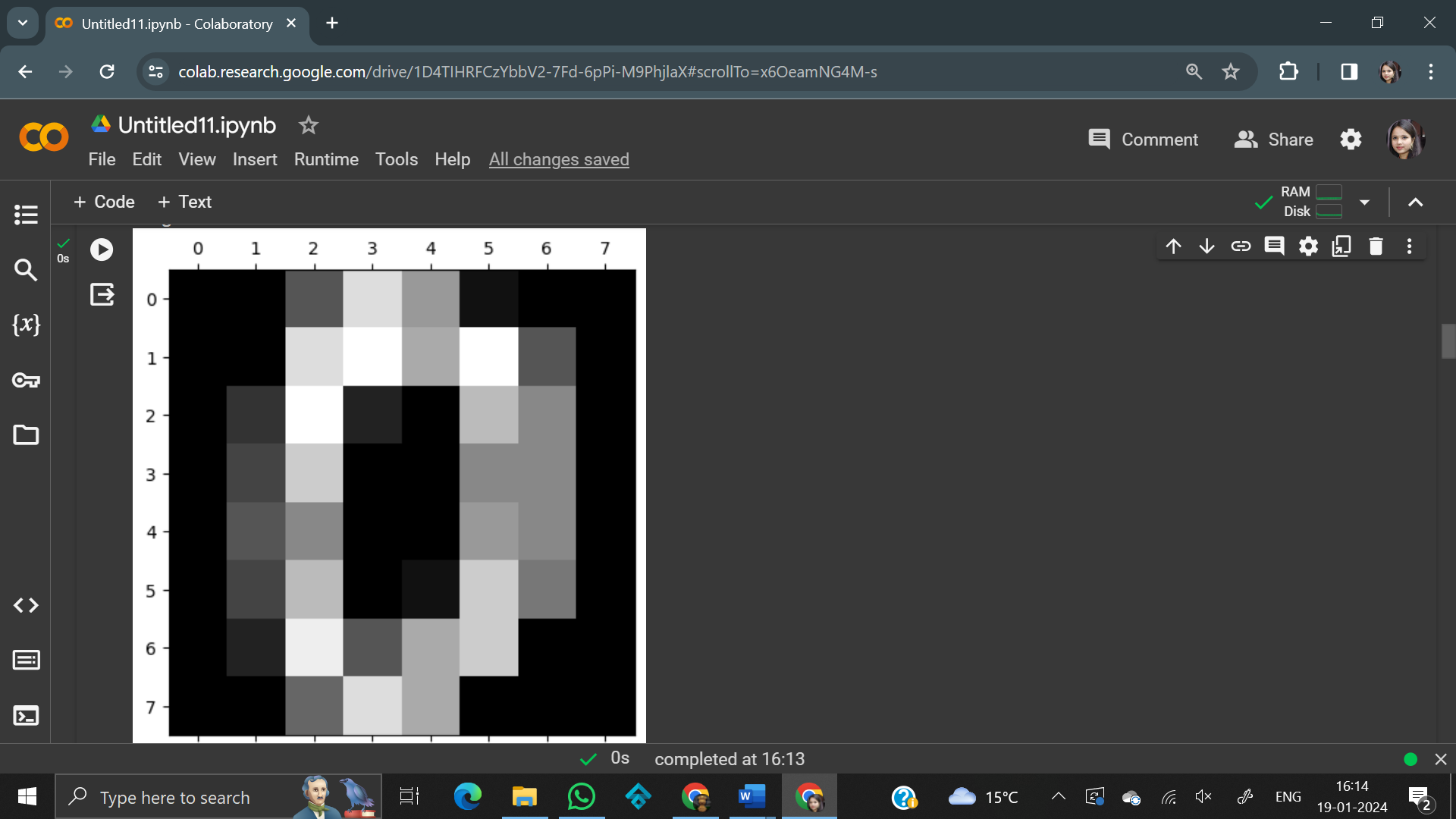
plt.gray()

plt.matshow(digits.images[0])

(1797, 8, 8)

<matplotlib.image.AxesImage at 0x7b7920407c40>

<Figure size 640x480 with 0 Axes>



|  |  |
| --- | --- |
| **EX NO-10** | **Write a program to Implement pattern recognition problems of speech recognition** |

import speech\_recognition as sr

r= sr.Recognizer()

with sr.Microphone() as source:

print("Speak anything :")

audio = r.listen(source)

try:

text = r.recognize\_google(audio)

if text =='exit':

flag=0

print("You said : {}".format(text))

except:

print("Sorry could not recognize what you said")

|  |  |
| --- | --- |
| **EX NO - 11** | **Write a program to implement Naive Bayes classification problem** |

**Naive Bayes classification**: Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other.

Imported Libraries:

* sklearn import preprocessing
* sklearn.naive\_bayes import GaussianNB

**Solution –**

weather=['sunny', 'sunny', 'overcast','rainy', 'rainy', 'overcast','sunny', 'rainy', 'sunny','overcast','rainy']

temp=['hot','hot','hot','mild','cool','cool','cool', 'mild', 'cool','mild','mild','hot','hot']

play=['no','no','yes','yes','no','yes','no', 'yes','yes','no','yes', 'no','no']

from sklearn  import preprocessing

le = preprocessing.LabelEncoder()

weather\_encoded = le.fit\_transform(weather)

print(weather\_encoded)

type(weather\_encoded)

temp\_encoded = le.fit\_transform(temp)

play\_encoded= le.fit\_transform(play)

print(temp\_encoded)

print (play\_encoded)

features = list (zip(weather\_encoded,temp\_encoded))

print (features)

from  sklearn.naive\_bayes import GaussianNB

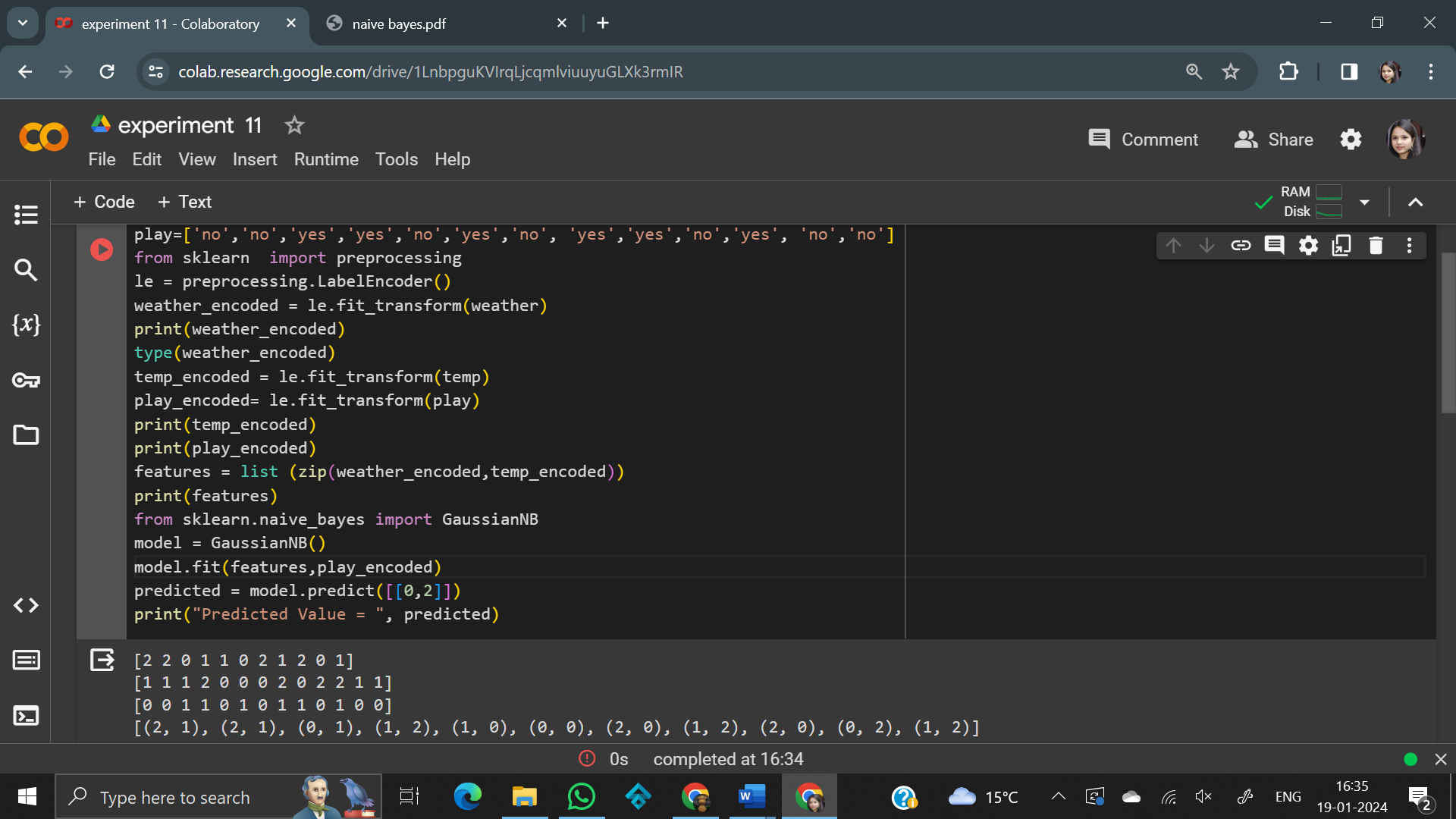
model = GaussianNB()

model.fit(features , play\_encoded)

predicted = model.predict([[0,2]])

print("Predicted Value = ", predicted)

**Output -**



|  |  |
| --- | --- |
| **EX NO - 12** | **Write a program to implement k-mean clustering problem** |

### K MEANS ALGORITHM:-

1. Specify number of clusters K.
2. Initialize centroids by first shuffling the dataset and then randomly selecting K data points for the centroids without replacement.
3. Keep iterating until there is no change to the centroids. i.e assignment of data points to clusters isn’t changing.
   * Compute the sum of the squared distance between data points and all centroids.
   * Assign each data point to the closest cluster (centroid).
   * Compute the centroids for the clusters by taking the average of the all data points that belong to each cluster.

**Solution -**

import matplotlib.pyplot as plt

from sklearn import datasets

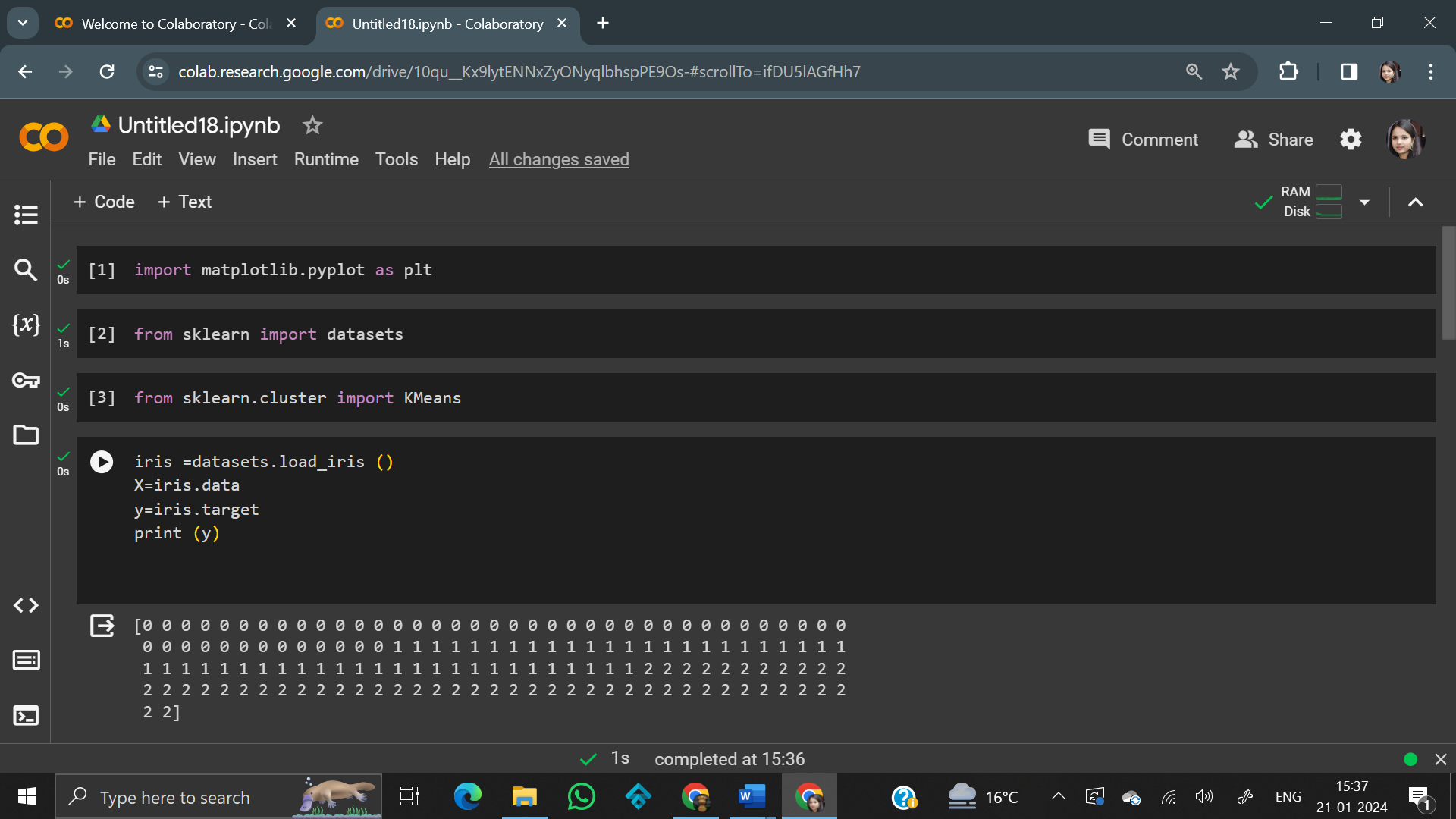
from sklearn.cluster import KMeans

iris =datasets.load\_iris ()

X=iris.data

y=iris.target

print (y)

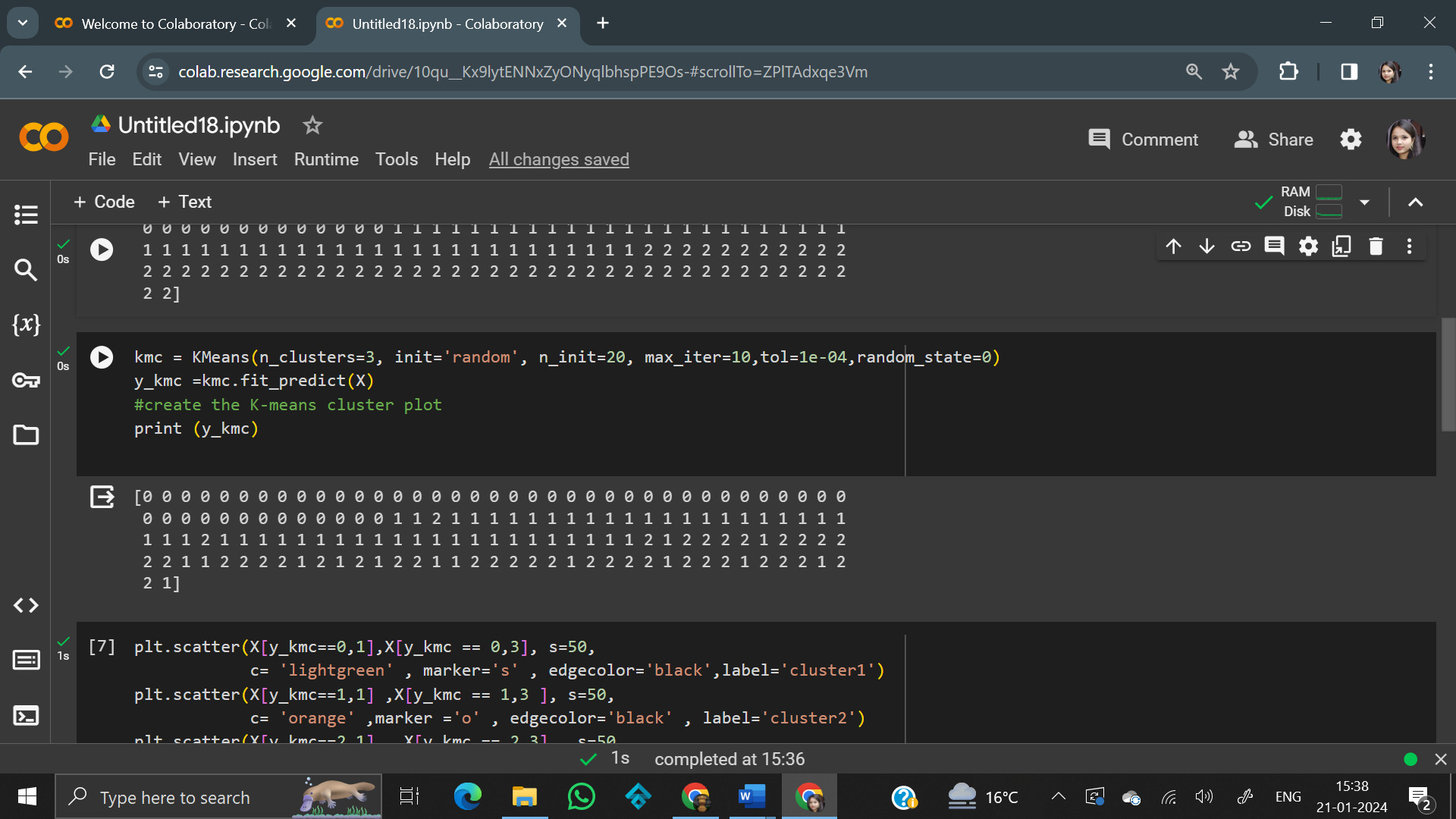


kmc = KMeans(n\_clusters=3, init='random', n\_init=20, max\_iter=10,tol=1e-04,random\_state=0)

y\_kmc =kmc.fit\_predict(X)

#create the K-means cluster plot

print (y\_kmc)



plt.scatter(X[y\_kmc==0,1],X[y\_kmc == 0,3], s=50,

            c= 'lightgreen' , marker='s' , edgecolor='black',label='cluster1')

plt.scatter(X[y\_kmc==1,1] ,X[y\_kmc == 1,3 ], s=50,

            c= 'orange' ,marker ='o' , edgecolor='black' , label='cluster2')

plt.scatter(X[y\_kmc==2,1] , X[y\_kmc == 2,3] , s=50,

            c= 'blue' , marker ='p', edgecolor ='black', label='cluster3')

plt.scatter(kmc.cluster\_centers\_[:,1],kmc.cluster\_centers\_[:,3],

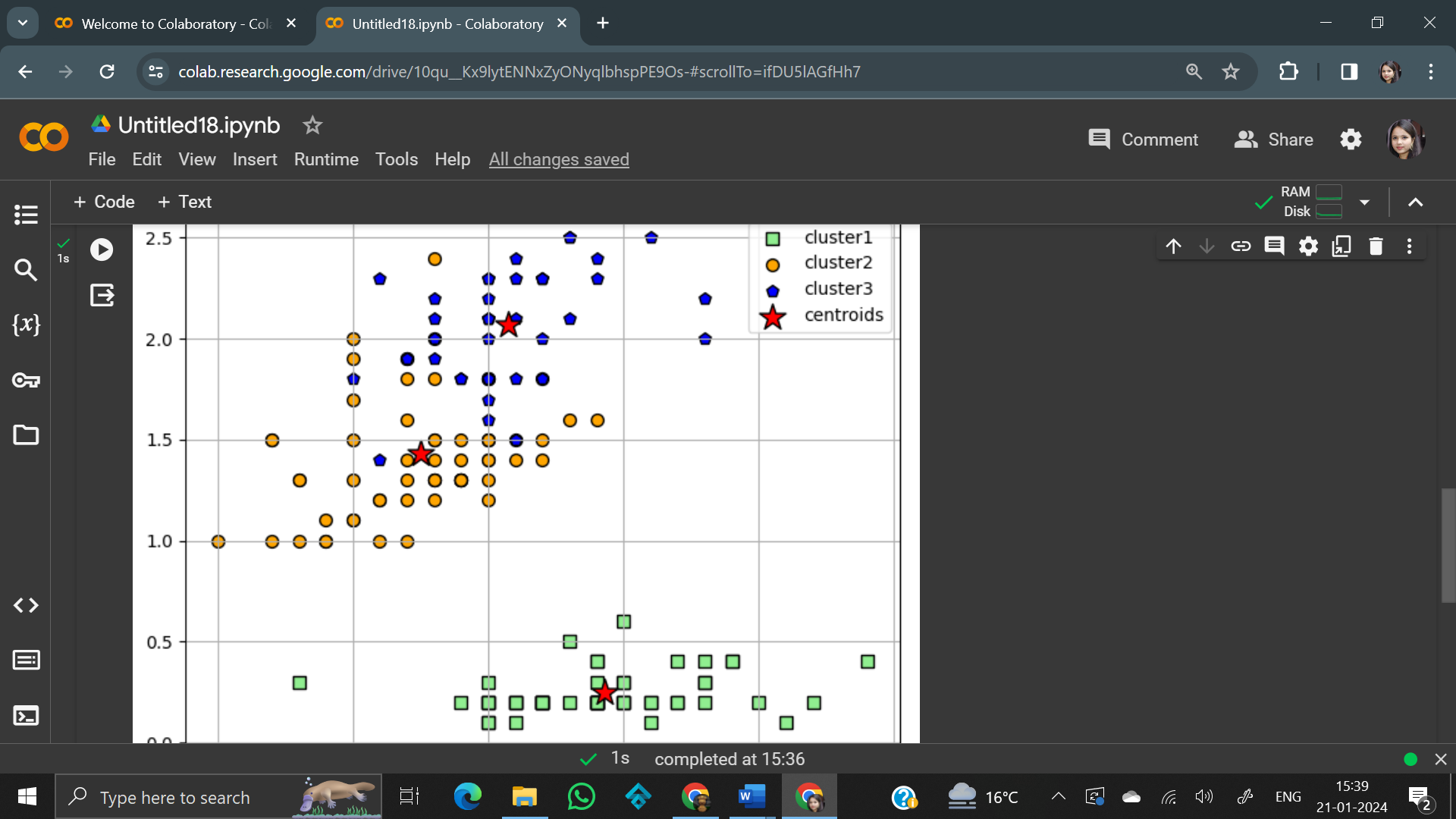
            s=250,marker='\*',c='red', edgecolor='black', label='centroids')

plt.legend (scatterpoints=1)

plt.grid()

plt.tight\_layout()

plt.show()



|  |  |
| --- | --- |
| **EX NO - 13** | **Write a program to convert text to speech** |

**Solution –**

import speech recognition as sr

import pyttsx3

# Initialize the recognizer

r = sr.Recognizer()

# Function to convert text to

# speech

def SpeakText(command):

# Initialize the engine

engine = pyttsx3.init()

engine.say(command)

engine.runAndWait()

text=" hi how are you, today is sunny day"

SpeakText(text)

# Loop infinitely for user to

# speak

while(1):

# Exception handling to handle

# exceptions at the runtime

try:

# use the microphone as source for input.

with sr.Microphone() as source2:

# wait for a second to let the recognizer

# adjust the energy threshold based on

# the surrounding noise level

r.adjust\_for\_ambient\_noise(source2, duration=0.2)

#listens for the user's input

audio2 = r.listen(source2)

# Using ggogle to recognize audio

MyText = r.recognize\_google(audio2)

MyText = MyText.lower()

print("Did you say "+MyText)

SpeakText(MyText)

except sr.RequestError as e:

print("Could not request results; {0}".format(e))

except sr.UnknownValueError:

print("unknown error occured")