RAJALAKSHMI ENGINEERING COLLEGE (Autonomous)

RAJALAKSHMI NAGAR, THANDALAM, CHENNAI-602105

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

RAJALAKSHMI
ENGINEERING COLLEGE
An AUTONOMOUS Institution
Affiliated to ANNA UNIVERSITY, Chennai

AI19341

PRINCIPLES OF ARTIFICIAL INTELLIGENCE LAB

THIRD YEAR

FIFTH SEMESTER

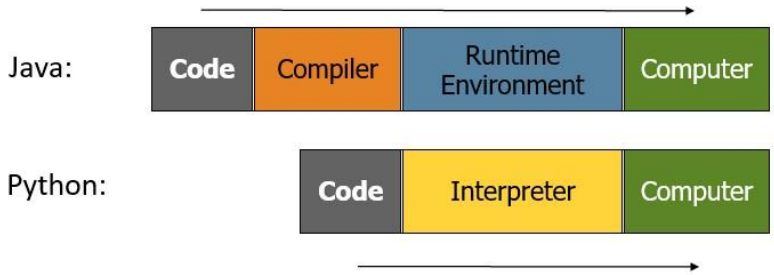
INDEX

S.NO	DATE	EXP NAME	VIVA MARK	SIGNATURE
1.		8QUEENS PROBLEM		
2.		DEPTH FIRST SEARCH		
3.		DEPTH FIRST SEARCH – WATER JUG PROBLEM		
4.		MINMAX ALGORITHM		
5.		A* SEARCH ALGORITHM		
6.		INTRODUCTION TO PROLOG		
7.		PROLOG FAMILY TREE		
8.		IMPLEMENTATION ARTIFICIAL NEURAL NETWORKS FOR AN APPLICATION USING PYTHON - REGRESSION		
9.		IMPLEMENTATION OF DECISION TREE CLASSIFICATION TECHNIQUES		
10.		IMPLEMENTATION OF CLUSTERING TECHNIQUES K - MEANS		

Working Tools and Language

PYTHON LANGUAGE

- Interpreter Languages
- Interpreted
- Not compiled like Java
- Code is written and then directly executed by an interpreter
- Type commands into interpreter and see immediate results



Python Installation Steps

Windows:

- Download Python from http://www.python.org
- Install Python.
- Run **Idle** from the Start Menu.

Mac OS X:

- Python is already installed.
- Open a terminal and run `python` or run Idle from Finder.

Linux:

- Chances are you already have Python installed. To check, run `python` from the terminal.
- If not, install from your distribution's package system.

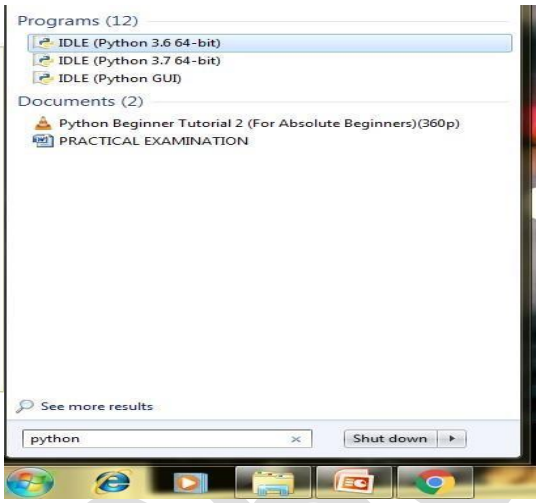
Note: For step by step installation instructions, see the course web site.

Step 1: Download the Python 3 Installer

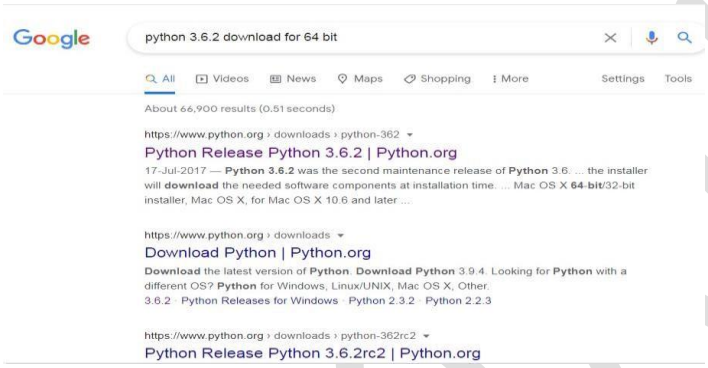
- Open a browser window and navigate to the Download page for Windows at python.org.
- Underneath the heading at the top that says Python Releases for Windows, click on the link for the Latest Python 3 Release – Python 3.x.x. (As of this writing, the latest version is Python 3.7.2.)
- Scroll to the bottom and select either Windows x86-64 executable installer for 64-bit or Windows x86 executable installer for 32-bit.

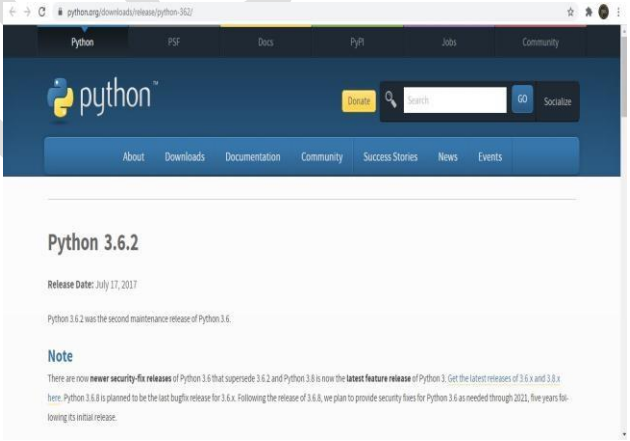
Step 2: Run the Installer

- Once you have chosen and downloaded an installer, simply run it by double-clicking on the downloaded file. A dialog should appear that looks something like this:



INSTALLATION STEPS ON PYTHON

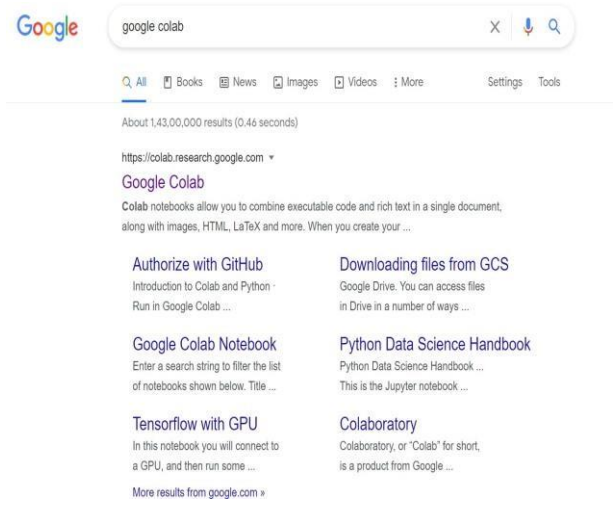


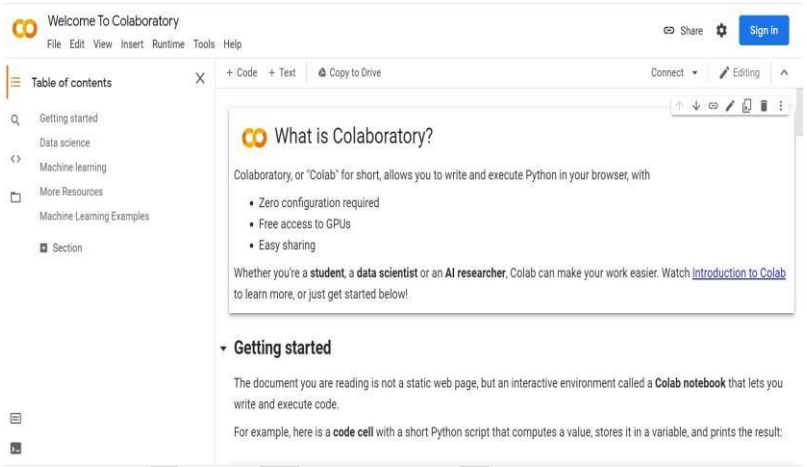


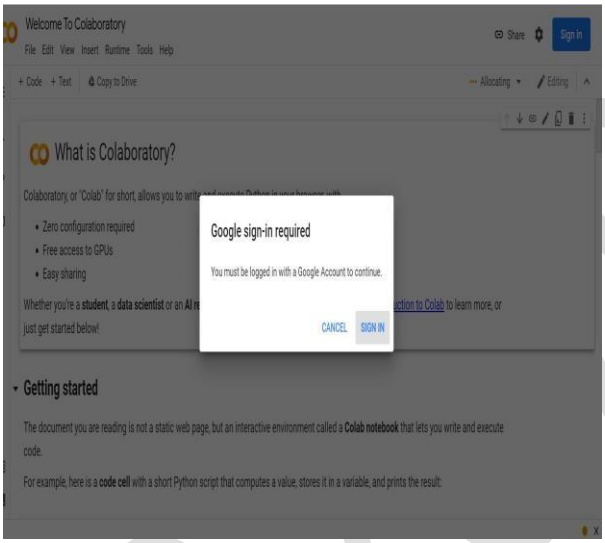
Files

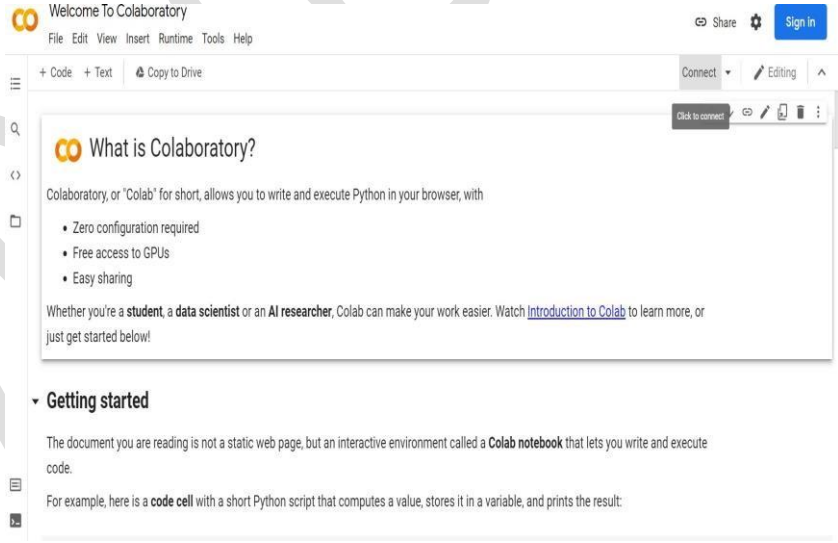
Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		e1a36bfffdd1d3a780b1825daf16e56c	22580749	SIG
XZ compressed source tarball	Source release		2c6884647199489727836 4fc18730dd9	16907204	SIG
Mac OS X 64-bit/32-bit installer	Mac OS X	for Mac OS X 10.6 and later	86e6193fd56b1e757fc8a5a2bb6c52ba	27561006	SIG
Windows help file	Windows		e520a5c1c3e3f02f68e3db23f74a7a90	8010498	SIG
Windows x86-64 embeddable zip file	Windows	for AMD64/EM64T/x64	0fdfe9f79e0991815d6fc1712871c17f	7047535	SIG
Windows x86-64 executable installer	Windows	for AMD64/EM64T/x64	4377e7d4e6877c248446f7cd6a1430cf	31434856	SIG
Windows x86-64 web-based installer	Windows	for AMD64/EM64T/x64	58ffad3d92a590a463908dfedbc68c18	1312496	SIG
Windows x86 embeddable zip file	Windows		2ca4768fdbadf6e670e97857bfab83e8	6332409	SIG
Windows x86 executable installer	Windows		8d8e1711ef9a4b3d3d0ce21e4155c0f5	30507592	SIG
Windows x86 web-based installer	Windows		ccb7d66e3465eaf40ade05b76715b56c	1287040	SIG

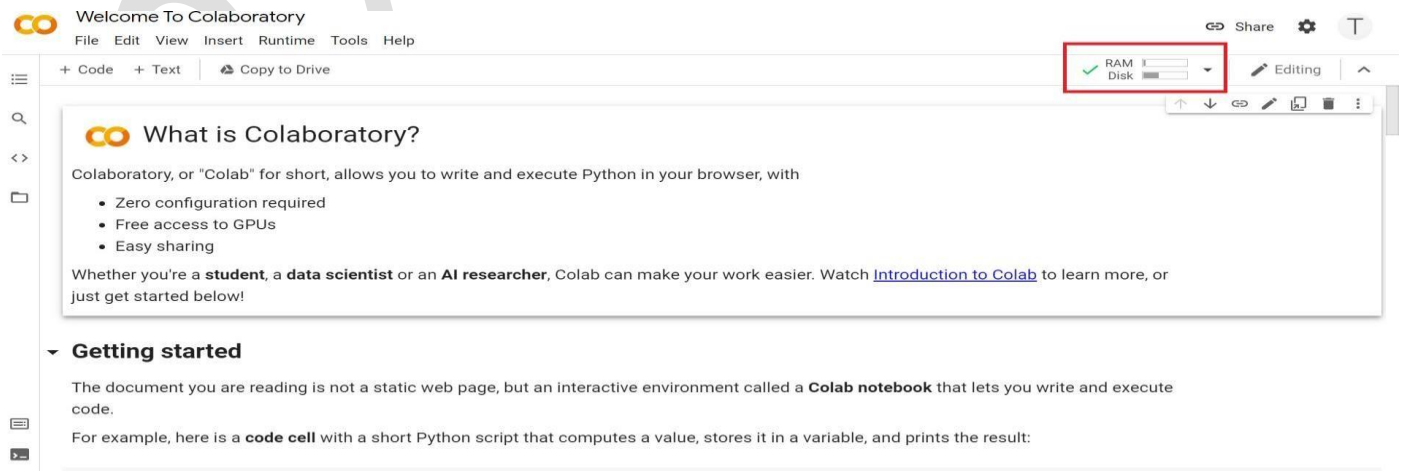
WORKING WITH GOOGLE COLABORATORY

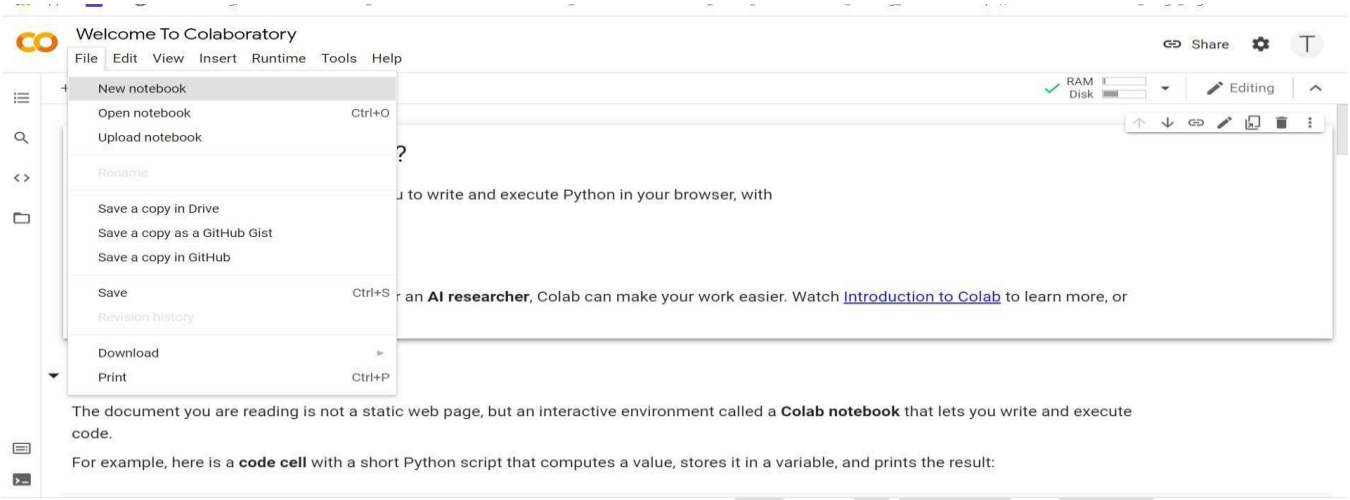


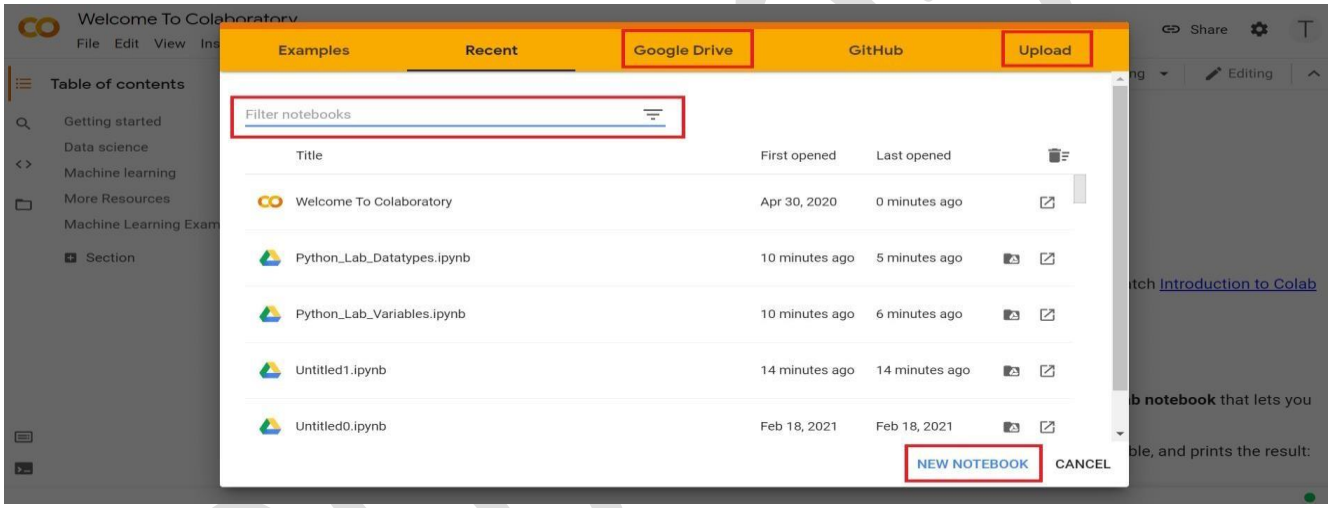


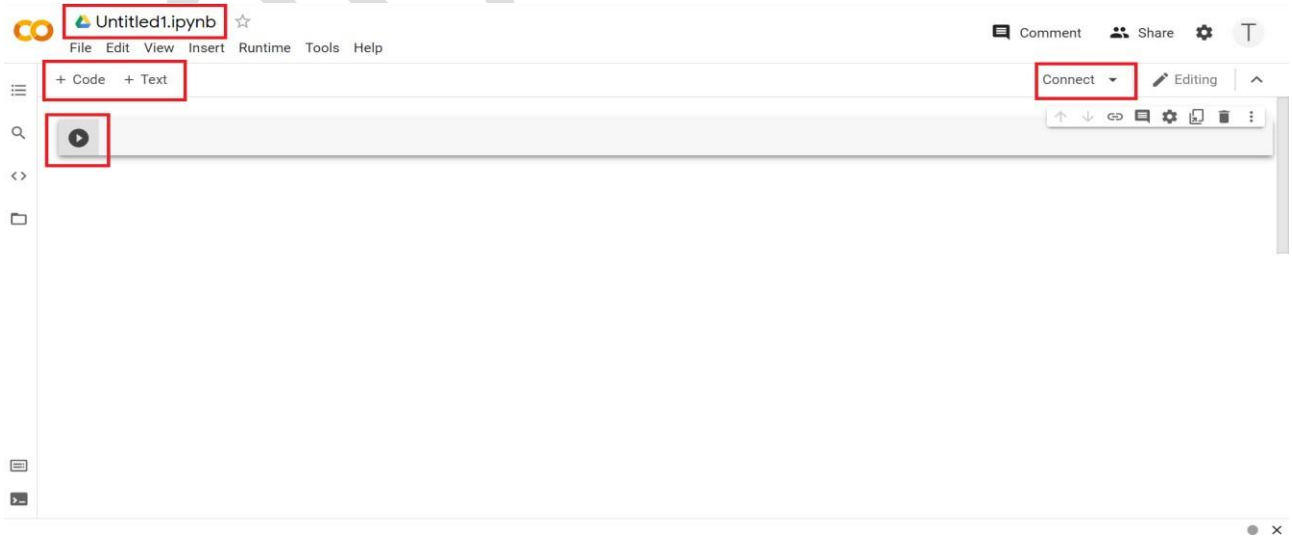







Welcome To Colaboratory
File Edit View Insert Runtime Tools Help
Share
RAM
Disk
Editing
New notebook
Open notebook Ctrl+O
Upload notebook
Rename
Save a copy in Drive
Save a copy as a GitHub Gist
Save a copy in GitHub
Save Ctrl+S
Revision history
Download
Print Ctrl+P
to write and execute Python in your browser, with
an AI researcher, Colab can make your work easier. Watch Introduction to Colab to learn more, or
The document you are reading is not a static web page, but an interactive environment called a Colab notebook that lets you write and execute code.
For example, here is a code cell with a short Python script that computes a value, stores it in a variable, and prints the result:


Welcome To Colaboratory
Share
Editing
Table of contents
Getting started
Data science
Machine learning
More Resources
Machine Learning Exam
Section
Examples
Recent
Google Drive
GitHub
Upload
Filter notebooks
Title
First opened
Last opened
Welcome To Colaboratory
Apr 30, 2020
0 minutes ago
Python_Lab_Datatypes.ipynb
10 minutes ago
5 minutes ago
Python_Lab_Variables.ipynb
10 minutes ago
6 minutes ago
Untitled1.ipynb
14 minutes ago
14 minutes ago
Untitled0.ipynb
Feb 18, 2021
Feb 18, 2021
NEW NOTEBOOK
CANCEL


Untitled1.ipynb
File Edit View Insert Runtime Tools Help
Comment
Share
+ Code
+ Text
Connect
Editing

EX.NO: 01 **DATE:**

REG.NO: 220701050

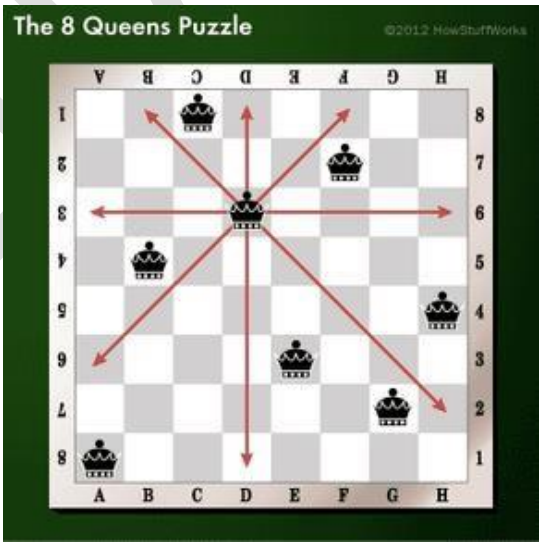
8- QUEENS PROBLEM

AIM :

To implement an 8-Queesns problem using Python.

You are given an 8x8 board; find a way to place 8 queens such that no queen can attack any other queen on the chessboard. A queen can only be attacked if it lies on the same row, same column, or the same diagonal as any other queen. Print all the possible configurations.

To solve this problem, we will make use of the Backtracking algorithm. The backtracking algorithm, in general checks all possible configurations and test whether the required result is obtained or not. For the given problem, we will explore all possible positions the queens can be relatively placed at. The solution will be correct when the number of placed queens = 8.



CODE:

```
def share_diagonal(x0, y0, x1, y1):
    dx = abs(x0 - x1)
    dy = abs(y0 - y1)

    return dy == dx


def col_clashes(bs, c):
    for i in range(c):
        if share_diagonal(i, bs[i], c, bs[c]):
            return True

    return False


def has_clashes(the_board):
    for col in range(1, len(the_board)):
        if col_clashes(the_board, col):
            return True
    return False


def main():
    import random
    rng = random.Random()

    bd = list(range(8))
    num_found = 0
    tries = 0
    result = []
    while num_found < 10:
        rng.shuffle(bd)
        tries += 1

        if not has_clashes(bd) and bd not in result:
            print("Found solution {0} in {1} tries.".format(bd, tries))
            tries = 0
            num_found += 1
            result.append(list(bd))
    print(result)


main()
```

OUTPUT:

```
Found solution [2, 6, 1, 7, 5, 3, 0, 4] in 75 tries.
Found solution [4, 2, 0, 6, 1, 7, 5, 3] in 738 tries.
Found solution [3, 6, 2, 7, 1, 4, 0, 5] in 646 tries.
Found solution [7, 3, 0, 2, 5, 1, 6, 4] in 473 tries.
Found solution [5, 3, 6, 0, 7, 1, 4, 2] in 113 tries.
Found solution [3, 0, 4, 7, 1, 6, 2, 5] in 481 tries.
Found solution [4, 6, 1, 3, 7, 0, 2, 5] in 1227 tries.
Found solution [5, 3, 1, 7, 4, 6, 0, 2] in 161 tries.
Found solution [5, 3, 0, 4, 7, 1, 6, 2] in 117 tries.
Found solution [5, 1, 6, 0, 3, 7, 4, 2] in 64 tries.
[[2, 6, 1, 7, 5, 3, 0, 4], [4, 2, 0, 6, 1, 7, 5, 3], [3, 6, 2, 7, 1, 4, 0, 5],
```

RESULT:

Thus, the 8-Queesns program has been implemented successfully.

EX.NO: 02

DATE:

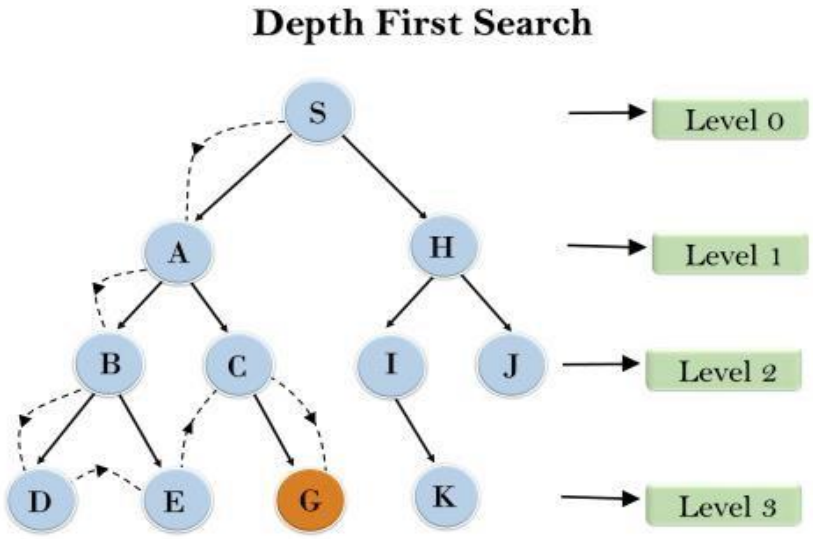
REG.NO: 220701050

DEPTH-FIRST SEARCH

AIM :

To implement a depth-first search problem using Python.

- Depth-first search (DFS) algorithm or searching technique starts with the root node of graph G, and then travel deeper and deeper until we find the goal node or the node which has no children by visiting different node of the tree.
- The algorithm, then backtracks or returns back from the dead end or last node towards the most recent node that is yet to be completely unexplored.
- The data structure (DS) which is being used in DFS Depth-first search is stack. The process is quite similar to the BFS algorithm.
- In DFS, the edges that go to an unvisited node are called discovery edges while the edges that go to an already visited node are called block edges.



CODE:

```
def dfs(graph, start, visited=None):
  if visited is None:
    visited = set()
  visited.add(start)
  print(start, end=" ")
  for neighbour in graph[start]:
    if neighbour not in visited:
      dfs(graph, neighbour, visited)

graph = {
  'A' : ['B','C'],
  'B' : ['D', 'E'],
  'C' : ['F'],
  'D' : [],
  'E' : ['F'],
  'F' : []
}

dfs(graph, 'A')
```

OUTPUT:

```
A B D E F C
```

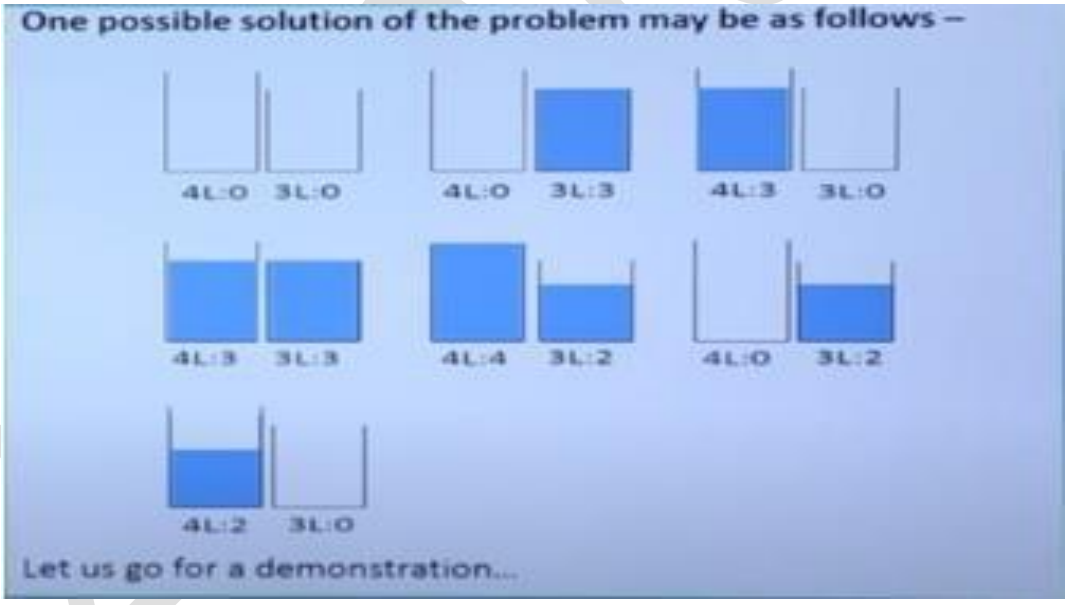
RESULT:

Thus, the depth-first search program has been implemented successfully.

EX.NO: 03 **DATE:**
REG.NO: 220701050

DEPTH-FIRST SEARCH – WATER JUG PROBLEM

In the **water jug problem in Artificial Intelligence**, we are provided with two jugs: one having the capacity to hold 3 gallons of water and the other has the capacity to hold 4 gallons of water. There is no other measuring equipment available and the jugs also do not have any kind of marking on them. So, the agent's task here is to fill the 4-gallon jug with 2 gallons of water by using only these two jugs and no other material. Initially, both our jugs are empty.



CODE:

```
def is_valid_state(state, visited):
    return state not in visited

def dfs(current_state, target, jug1_capacity, jug2_capacity, visited, solution):
    visited.add(current_state)
    solution.append(current_state)

    if current_state[0] == target or current_state[1] == target:
        return True

    jug1, jug2 = current_state

    possible_states = [
        (jug1_capacity, jug2),
        (jug1, jug2_capacity),
        (0, jug2),
        (jug1, 0),
        (max(jug1 - (jug2_capacity - jug2), 0), min(jug2 + jug1, jug2_capacity)),
        (min(jug1 + jug2, jug1_capacity), max(jug2 - (jug1_capacity - jug1), 0)),
    ]

    for state in possible_states:
        if is_valid_state(state, visited):
            if dfs(state, target, jug1_capacity, jug2_capacity, visited, solution):
                return True

    solution.pop()
    return False
```

```
def water_jug_problem(jug1_capacity, jug2_capacity, target):
    visited = set()
    solution = []

    if dfs((0, 0), target, jug1_capacity, jug2_capacity, visited, solution):
        return solution
    else:
        return "No solution found."

jug1_capacity = 4
jug2_capacity = 3
target = 2

solution = water_jug_problem(jug1_capacity, jug2_capacity, target)
print("Solution steps:")
for step in solution:
    print(step)
```

OUTPUT:

```
Solution steps:
(0, 0)
(4, 0)
(4, 3)
(0, 3)
(3, 0)
(3, 3)
(4, 2)
```

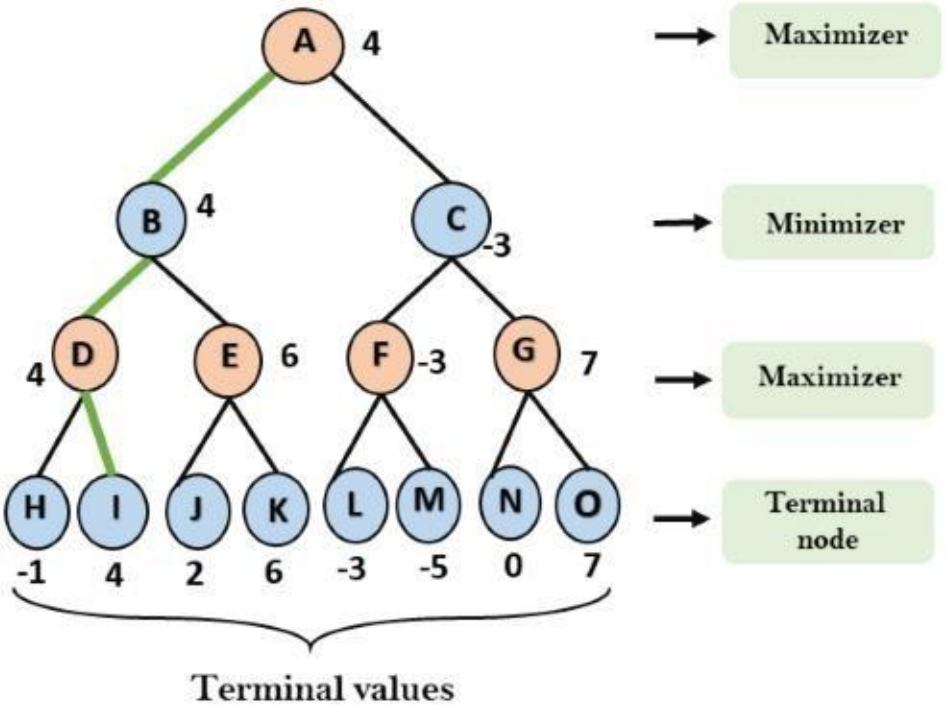

RESULT:

Thus, the water jug program has been implemented successfully.

EX.NO: 04 **DATE:**
REG.NO: 220701050

MINIMAX ALGORITHM

- A simple example can be used to explain how the minimax algorithm works. We've included an example of a game-tree below, which represents a two-player game.
- There are two players in this scenario, one named Maximizer and the other named Minimizer.
- Maximizer will strive for the highest possible score, while Minimizer will strive for the lowest possible score.
- Because this algorithm uses DFS, we must go all the way through the leaves to reach the terminal nodes in this game-tree.
- The terminal values are given at the terminal node, so we'll compare them and retrace the tree till we reach the original state.



CODE:

```
# Function to perform Minimax Algorithm
def minimax(depth, node_index, is_max, scores, height):
    if depth == height:
        return scores[node_index]

    if is_max:
        return max(
            minimax(depth + 1, node_index * 2, False, scores, height),
            minimax(depth + 1, node_index * 2 + 1, False, scores, height),
        )
    else:
        return min(
            minimax(depth + 1, node_index * 2, True, scores, height),
            minimax(depth + 1, node_index * 2 + 1, True, scores, height),
        )

def calculate_tree_height(scores):
    import math
    return math.ceil(math.log2(len(scores)))

scores = [3, 5, 6, 9, 1, 2, 0, -1]
height = calculate_tree_height(scores)

optimal_value = minimax(0, 0, True, scores, height)
print(f"The optimal value for the Maximizer is: {optimal_value}")
```

OUTPUT:

```
The optimal value for the Maximizer is: 5
```

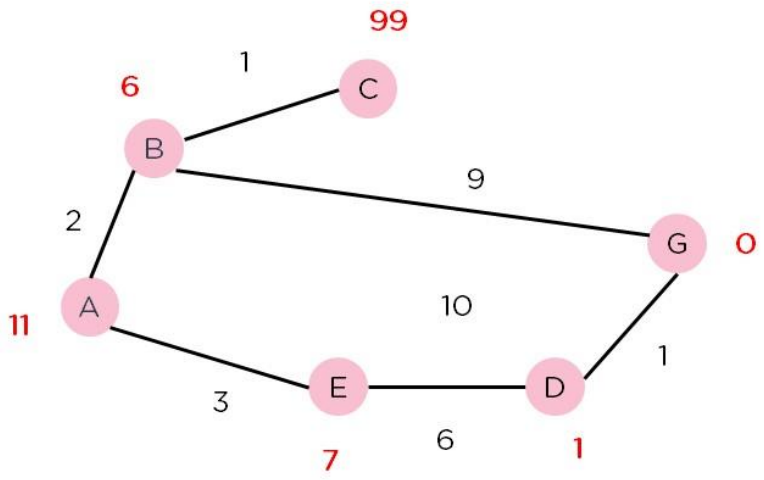
RESULT:

Thus, the minmax algorithm program has been implemented successfully.

EX.No : 05 **DATE :**
REG.NO: 220701050

A* SEARCH ALGORITHM

A heuristic algorithm sacrifices optimality, with precision and accuracy for speed, to solve problems faster and more efficiently.
All graphs have different nodes or points which the algorithm has to take, to reach the final node. The paths between these nodes all have a numerical value, which is considered as the weight of the path. The total of all paths transverse gives you the cost of that route.
Initially, the Algorithm calculates the cost to all its immediate neighboring nodes,n, and chooses the one incurring the least cost. This process repeats until no new nodes can be chosen and all paths have been traversed. Then, you should consider the best path among them. If f(n) represents the final cost, then it can be denoted as : f(n) = g(n) + h(n), where : g(n) = cost of traversing from one node to another. This will vary from node to node h(n) = heuristic approximation of the node's value. This is not a real value but an approximation cost.



CODE:

```
from queue import PriorityQueue
def a_star_search(graph, heuristic, start, goal):

    open_list = PriorityQueue()
    open_list.put((0, start))  # f(n), node

    came_from = {}

    g_score = {node: float('inf') for node in graph}
    g_score[start] = 0

    f_score = {node: float('inf') for node in graph}
    f_score[start] = heuristic[start]

    while not open_list.empty():
        _, current = open_list.get()

        if current == goal:
            path = []
            while current in came_from:
                path.append(current)
                current = came_from[current]
            path.append(start)
            return path[::-1], g_score[goal]

        for neighbor, cost in graph[current].items():
            tentative_g_score = g_score[current] + cost
            if tentative_g_score < g_score[neighbor]:
                came_from[neighbor] = current
                g_score[neighbor] = tentative_g_score
                f_score[neighbor] = g_score[neighbor] + heuristic[neighbor]
                open_list.put((f_score[neighbor], neighbor))

    return None, float('inf')

graph = {
    'A': {'B': 1, 'C': 3},
    'B': {'D': 1, 'E': 4},
    'C': {'F': 6},
    'D': {'G': 5},
    'E': {'G': 1},
    'F': {'G': 2},
    'G': {}
}

heuristic = {
    'A': 6,
    'B': 4,
    'C': 4,
    'D': 2,
    'E': 1,
    'F': 2,
    'G': 0
}

start_node = 'A'
goal_node = 'G'
path, cost = a_star_search(graph, heuristic, start_node, goal_node)

print(f"Shortest path: {path}")
print(f"Total cost: {cost}")
```

OUTPUT:

```
Shortest path: ['A', 'B', 'E', 'G']
Total cost: 6
```


RESULT:

Thus, the A* search program has been implemented successfully.

EX.NO : 06 **DATE:**
REG.NO: 220701050

INTRODUCTION TO PROLOG

AIM

To learn PROLOG terminologies and write basic programs.

TERMINOLOGIES

1. Atomic Terms: -

   Atomic terms are usually strings made up of lower- and uppercase letters, digits, and the underscore, starting with a lowercase letter.

   Ex:

   dog
   ab_c_321

2. Variables: -

   Variables are strings of letters, digits, and the underscore, starting with a capital letter or an underscore.

   Ex:

   Dog
   Apple_420

3. Compound Terms: -

   Compound terms are made up of a PROLOG atom and a number of arguments (PROLOG terms, i.e., atoms, numbers, variables, or other compound terms) enclosed in parentheses and separated by commas.

   Ex:

   is_bigger(elephant,X)
   f(g(X,_),7) 4. Facts: -

   A fact is a predicate followed by a dot.

   Ex:

   bigger_animal(whale).
   life_is_beautiful.

5. Rules: -

   A rule consists of a head (a predicate) and a body (a sequence of predicates separated by commas).

   Ex:

   is_smaller(X,Y):-is_bigger(Y,X).
   aunt(Aunt,Child):-sister(Aunt,Parent),parent(Parent,Child).

SOURCE CODE:

KB1:

woman(mia). woman(jody).
woman(yolanda).
playsAirGuitar(jody).
party.
Query 1: ?-woman(mia).
Query 2: ?-playsAirGuitar(mia).
Query 3: ?-party.
Query 4: ?-concert.

OUTPUT: -

```
?- woman(mia).
true.

?- playsAirGuitar(mia).
false.

?- party.
true.

?- concert.
ERROR: Unknown procedure: concert/0 (DWIM could not correct goal)
?-
```

KB2:

happy(yolanda). listens2music(mia).
Listens2music(yolanda):-happy(yolanda). playsAirGuitar(mia):-listens2music(mia).
playsAirGuitar(Yolanda):-listens2music(yolanda).

OUTPUT: -

```
?- playsAirGuitar(mia).
true .

?- playsAirGuitar(yolanda).
true.

?-
```

KB3: likes(dan,sally). likes(sally,dan).
likes(john,brittney). married(X,Y) :-
likes(X,Y) , likes(Y,X). friends(X,Y) :-
likes(X,Y) ; likes(Y,X).

OUTPUT: -

```
?- likes(dan,X).
X = sally.

?- married(dan,sally).
true.

?- married(john,brittney).
false.
```

KB4: food(burger).
food(sandwich).
food(pizza).
lunch(sandwich).
dinner(pizza).
meal(X):-food(X).

OUTPUT:

```
?-
|    food(pizza).
true.

?- meal(X),lunch(X).
X = sandwich ;

?- dinner(sandwich).
false.

?-
```

KB5:
owns(jack,car(bmw)).
owns(john,car(chevy)).
owns(olivia,car(civic)).
owns(jane,car(chevy)).
sedan(car(bmw)). sedan(car(civic)).
truck(car(chevy)).

OUTPUT:

```
?-
|    owns(john,X).
X = car(chevy).

?- owns(john,_).
true.

?- owns(Who,car(chevy)).
Who = john ,

?- owns(jane,X),sedan(X).
false.

?- owns(jane,X),truck(X).
X = car(chevy).
```


RESULT:

Thus, the Prolog terminologies and basic program has been implemented successfully.

EX.NO: 07 **DATE:**
REG.NO: 220701050

PROLOG- FAMILY TREE

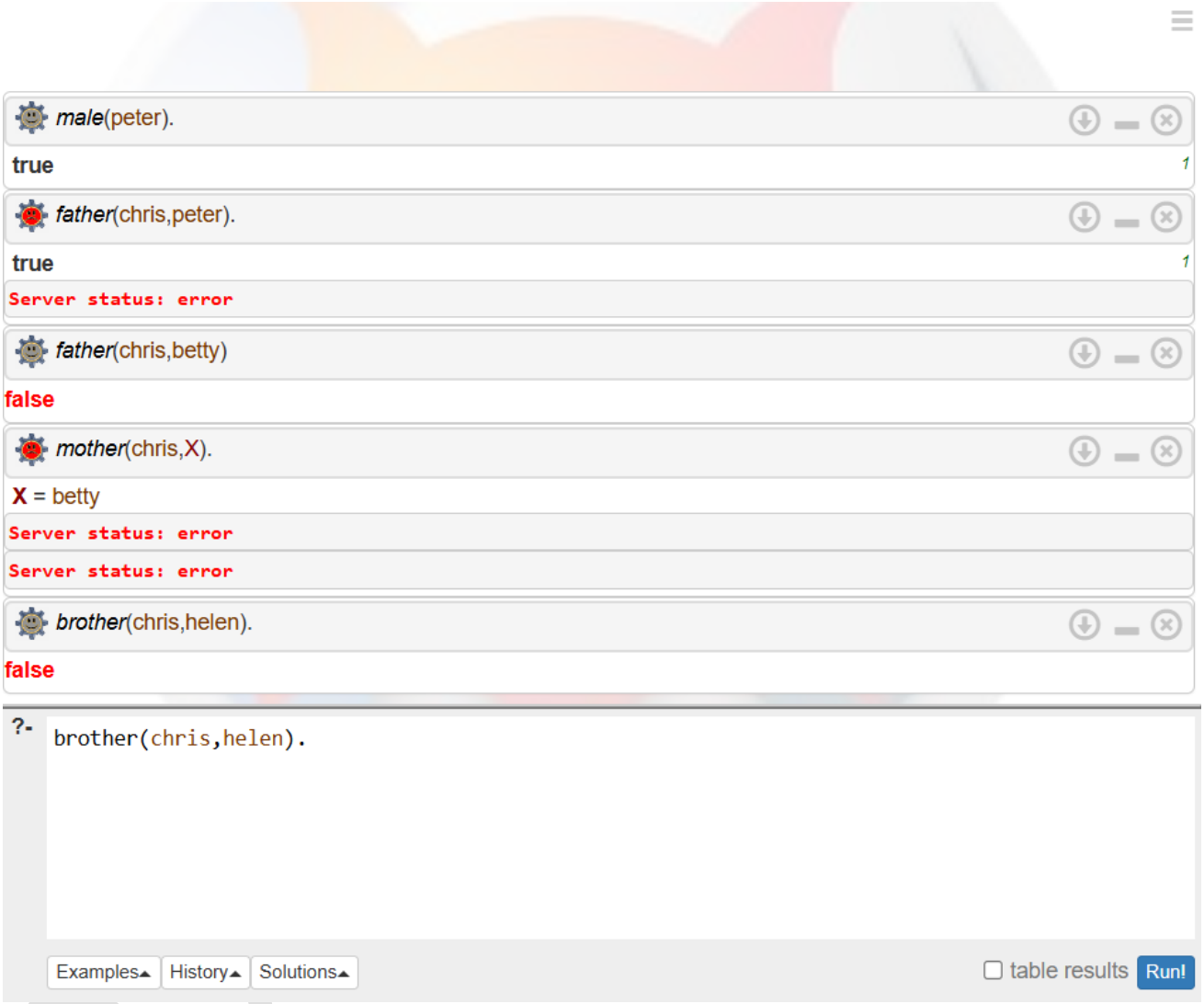
AIM

To develop a family tree program using PROLOG with all possible facts, rules, and queries.

SOURCE CODE:
KNOWLEDGE BASE:

```
/*FACTS :: */
male(peter).
male(john). male(chris).
male(kevin).

female(betty).
female(jeny). female(lisa).
female(helen).

parentOf(chris,peter).
parentOf(chris,betty).
parentOf(helen,peter).
parentOf(helen,betty).
parentOf(kevin,chris).
parentOf(kevin,lisa). parentOf(jeny,john).
parentOf(jeny,helen).

/*RULES :: */
/* son,parent
* son,grandparent*/

father(X,Y):- male(Y), parentOf(X,Y).

mother(X,Y):- female(Y), parentOf(X,Y).

grandfather(X,Y):- male(Y),parentOf(X,Z),parentOf(Z,Y).

grandmother(X,Y):- female(Y),parentOf(X,Z),parentOf(Z,Y).
```

brother(X,Y):- male(Y), father(X,Z), father(Y,W),Z==W.
sister(X,Y):- female(Y), father(X,Z),father(Y,W),Z==W.



RESULT:

Thus the family tree program using PROLOG with all possible facts, rules, and queries has been implemented successfully.

EX.NO : 08 **DATE :**

REG.NO: 220701050

IMPLEMENTING ARTIFICIAL NEURAL NETWORKS FOR AN APPLICATION USING PYTHON - REGRESSION

AIM :

To implementing artificial neural networks for an application in Regression using python.

Regression using Artificial Neural Networks

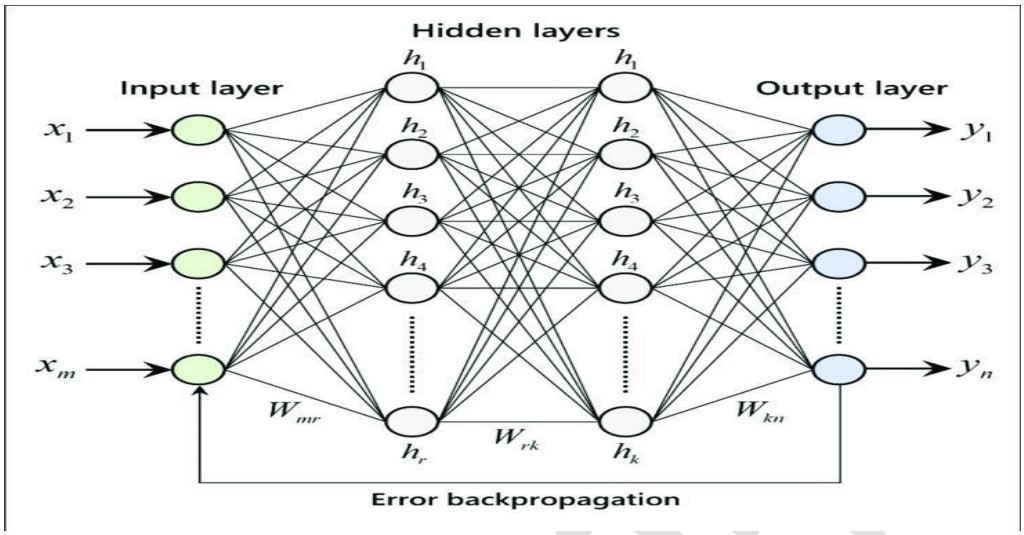
Why do we need to use Artificial Neural Networks for Regression instead of simply using Linear Regression?

The purpose of using Artificial Neural Networks for Regression over Linear Regression is that the linear regression can only learn the linear relationship between the features and target and therefore cannot learn the complex non-linear relationship. In order to learn the complex non-linear relationship between the features and target, we are in need of other techniques. One of those techniques is to use Artificial Neural Networks. Artificial Neural Networks have the ability to learn the complex relationship between the features and target due to the presence of activation function in each layer. Let's look at what are Artificial Neural Networks and how do they work.

Artificial Neural Networks

Artificial Neural Networks are one of the deep learning algorithms that simulate the workings of neurons in the human brain. There are many types of Artificial Neural Networks, Vanilla Neural Networks, Recurrent Neural Networks, and Convolutional Neural Networks. The Vanilla Neural Networks have the ability to handle structured data only, whereas the Recurrent Neural Networks and Convolutional Neural Networks have the ability to handle unstructured data very well. In this post, we are going to use Vanilla Neural Networks to perform the Regression Analysis.

Structure of Artificial Neural Networks



The Artificial Neural Networks consists of the Input layer, Hidden layers, Output layer. The hidden layer can be more than one in number. Each layer consists of n number of neurons. Each layer will be having an Activation Function associated with each of the neurons. The activation function is the function that is responsible for introducing non-linearity in the relationship. In our case, the output layer must contain a linear activation function. Each layer can also have regularizers associated with it. Regularizers are responsible for preventing overfitting.

Artificial Neural Networks consists of two phases,

- Forward Propagation
- Backward Propagation

Forward propagation is the process of multiplying weights with each feature and adding them. The bias is also added to the result. Backward propagation is the process of updating the weights in the model.

Backward propagation requires an optimization function and a loss function.

CODE:

```
import numpy as np
import tensorflow as tf
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X, y = make_regression(n_samples=500, n_features=5, noise=0.1, random_state=42)
y = y.reshape(-1, 1)

scaler_X = StandardScaler()
X_scaled = scaler_X.fit_transform(X)

scaler_y = StandardScaler()
y_scaled = scaler_y.fit_transform(y)

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_scaled, test_size=0.2, random_state=42)

model = tf.keras.Sequential([
    tf.keras.layers.Dense(16, activation='relu', input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(8, activation='relu'),
    tf.keras.layers.Dense(1)
])

model.compile(optimizer='adam',
              loss='mse',
              metrics=['mae'])

history = model.fit(X_train, y_train, epochs=100, batch_size=16, validation_split=0.1, verbose=1)

test_loss, test_mae = model.evaluate(X_test, y_test, verbose=0)

y_pred_scaled = model.predict(X_test)
y_pred = scaler_y.inverse_transform(y_pred_scaled)
y_test_original = scaler_y.inverse_transform(y_test)
print(f"Test MAE: {test_mae:.2f}")

print("\nSample Predictions:")
for i in range(5):
    print(f"Predicted: {y_pred[i][0]:.2f}, True: {y_test_original[i][0]:.2f}")
```

OUTPUT:

```
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 7.3554e-04 - mae: 0.0220 - val_loss: 9.3573e-04 - val_mae: 0.0241
Epoch 95/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 7.2037e-04 - mae: 0.0210 - val_loss: 0.0011 - val_mae: 0.0261
Epoch 96/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 6.8789e-04 - mae: 0.0207 - val_loss: 0.0010 - val_mae: 0.0255
Epoch 97/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 6.8820e-04 - mae: 0.0209 - val_loss: 0.0011 - val_mae: 0.0264
Epoch 98/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 7.2461e-04 - mae: 0.0216 - val_loss: 0.0010 - val_mae: 0.0249
Epoch 99/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 6.2160e-04 - mae: 0.0195 - val_loss: 0.0010 - val_mae: 0.0260
Epoch 100/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 6.1356e-04 - mae: 0.0194 - val_loss: 9.5559e-04 - val_mae: 0.0240
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
Test MAE: 0.03

Sample Predictions:
Predicted: -40.93, True: -38.46
Predicted: 4.83, True: 6.32
Predicted: 6.94, True: 6.62
Predicted: -122.65, True: -122.99
Predicted: 52.97, True: 54.43
```


RESULT:

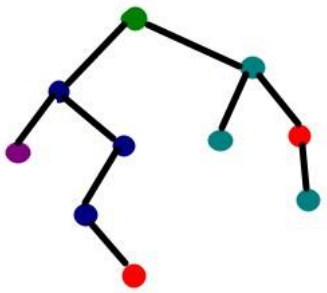
Thus, the artificial neural network application in regression program has been implemented successfully.

EX.NO : 09 **DATE :**

REG.NO: 220701050

IMPLEMENTATION OF DECISION TREE CLASSIFICATION TECHNIQUES

Decision Tree is one of the most powerful and popular algorithm. Decision-tree algorithm falls under the category of supervised learning algorithms. It works for both continuous as well as categorical output variables.



AIM:

To implement a decision tree classification technique for gender classification using python.

EXPLANATION:

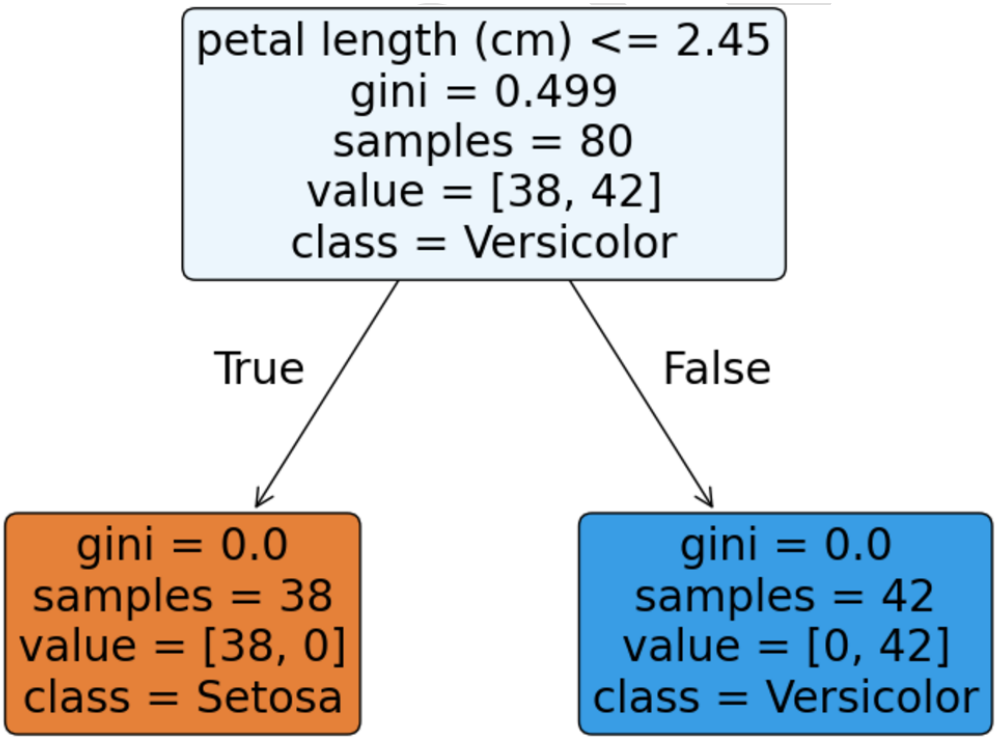
- Import tree from sklearn.
- Call the function DecisionTreeClassifier() from tree
- Assign values for X and Y.
- Call the function predict for Predicting on the basis of given random values for each given feature.
- Display the output.

CODE:

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
from sklearn import tree

data = load_iris()
X = data.data
y = data.target

X = X[y != 2]
y = y[y != 2]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = DecisionTreeClassifier(criterion='gini', max_depth=3, random_state=42)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=['Setosa', 'Versicolor']))

print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))

plt.figure(figsize=(10,8))
tree.plot_tree(model, filled=True, feature_names=data.feature_names, class_names=['Setosa', 'Versicolor'], rounded=True)
plt.show()
```

OUTPUT:

```
Accuracy: 1.00

Classification Report:
              precision    recall  f1-score   support

      Setosa       1.00      1.00      1.00        12
  Versicolor       1.00      1.00      1.00         8

    accuracy                           1.00        20
   macro avg       1.00      1.00      1.00        20
weighted avg       1.00      1.00      1.00        20


Confusion Matrix:
[[12  0]
 [ 0  8]]
```



RESULT:

Thus, the Decision Tree Classification program has been implemented successfully.

EX NO : 10 **DATE :**
REG.NO: 220701050

IMPLEMENTATION OF CLUSTERING TECHNIQUES K - MEANS

The ***k*-means clustering** method is an unsupervised machine learning technique used to identify clusters of data objects in a dataset. There are many different types of clustering methods, but *k*means is one of the oldest and most approachable. These traits make implementing *k*-means clustering in Python reasonably straightforward, even for novice programmers and data scientists.

If you're interested in learning how and when to implement *k*-means clustering in Python, then this is the right place. You'll walk through an end-to-end example of *k*-means clustering using Python, from preprocessing the data to evaluating results.

How does it work?

First, each data point is randomly assigned to one of the K clusters. Then, we compute the centroid (functionally the center) of each cluster, and reassign each data point to the cluster with the closest centroid. We repeat this process until the cluster assignments for each data point are no longer changing.

K-means clustering requires us to select K, the number of clusters we want to group the data into. The elbow method lets us graph the inertia (a distance-based metric) and visualize the point at which it starts decreasing linearly. This point is referred to as the "eblow" and is a good estimate for the best value for K based on our data.

AIM:

To implement a K - Means clustering technique using python language.

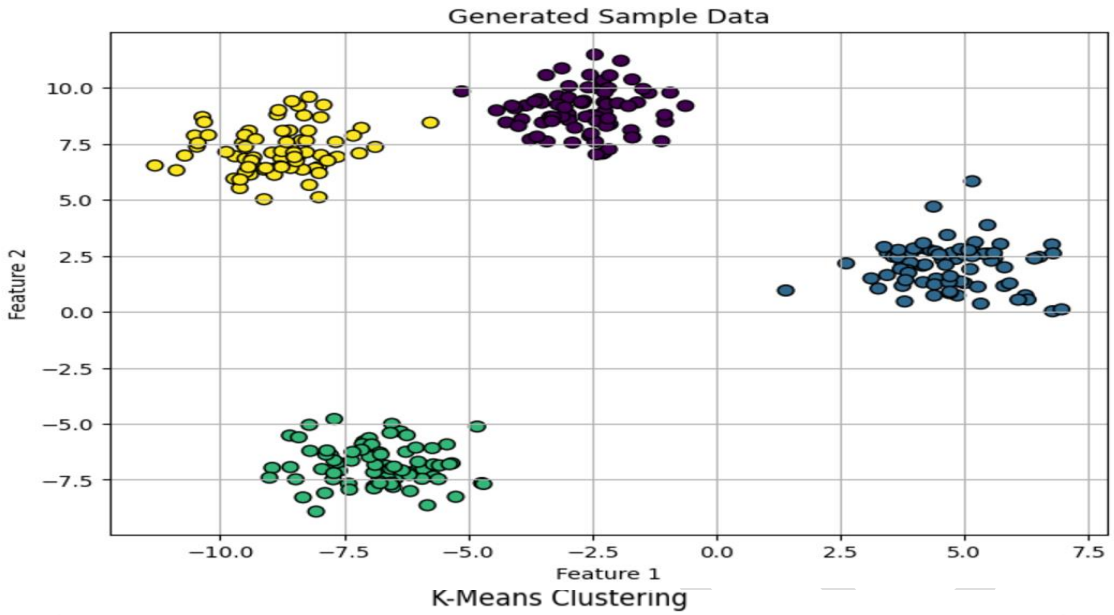
EXPLANATION:

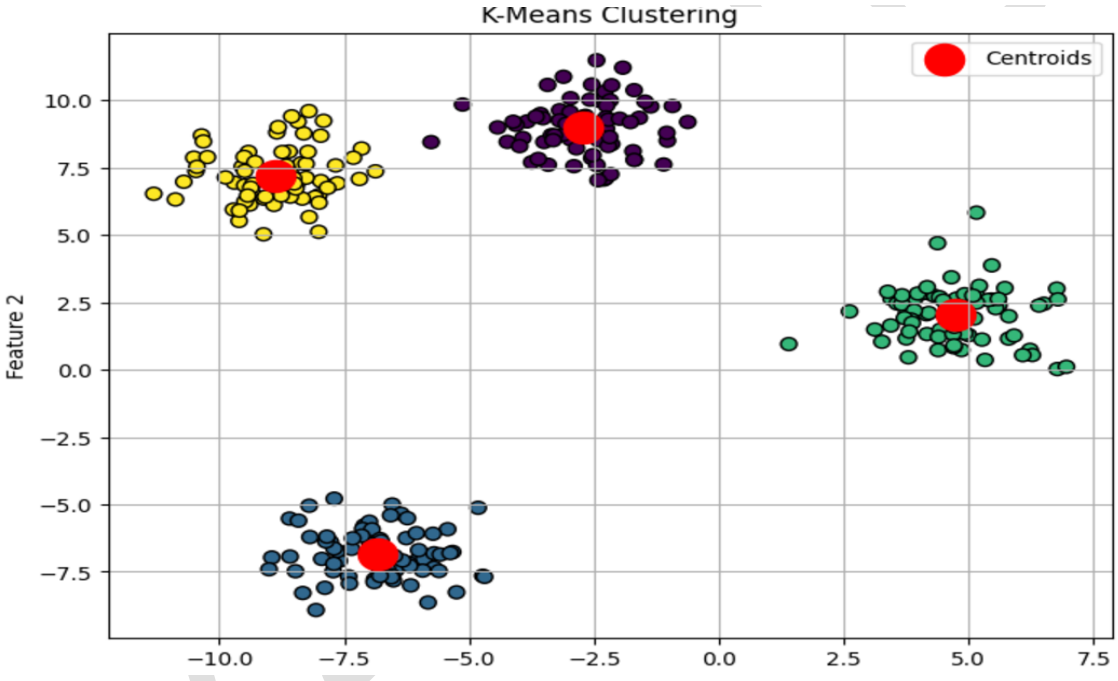
- Import KMeans from sklearn.cluster
- Assign X and Y.
- Call the function KMeans().
- Perform scatter operation and display the output.

CODE:

```
import numpy as np
import pandas as pd
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from sklearn.metrics import silhouette_score

np.random.seed(42)
X, y = make_blobs(n_samples=300, n_features=2, centers=4, cluster_std=1.0)

plt.figure(figsize=(8, 6))
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='viridis', marker='o', edgecolor='k', s=50)
plt.title('Generated Sample Data')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.grid()
plt.show()

kmeans = KMeans(n_clusters=4, random_state=42)
kmeans.fit(X)

y_kmeans = kmeans.predict(X)

plt.figure(figsize=(8, 6))
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, cmap='viridis', marker='o', edgecolor='k', s=50)
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=300, c='red', label='Centroids')
plt.title('K-Means Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.grid()
plt.show()

print(f"Inertia: {kmeans.inertia_:.2f}")

score = silhouette_score(X, y_kmeans)
print(f"Silhouette Score: {score:.2f}")
```

OUTPUT:


Generated Sample Data
Feature 2
Feature 1
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5


K-Means Clustering
Centroids
Feature 2
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5

RESULT:

Thus, the K-Means Clustering technique program has been implemented successfully.