

INDEX

Name: DEVDHARSHAN E.P.

Subject: POAI Observation Class: CSE Div: 'A' Roll No: 220700060

School: _____

Sl. No.	Date	Title	Page No.
1.	31/7	Basic Python Programs using Google Colab.	9
2.	7/8	Domain: Career technology.	9
3.	4/9	N-Queens.	9
4.	4/9	Depth first search.	10
5.	11/9	A* algorithm.	10
6.	18/9	A0* algorithm.	10
7.	25/9	Decision tree	10
8.	9/10	K-means.	10
9.	16/10	Artificial neural network.	10
10.	23/10	Minimax.	10
11.	30/10	Introduction to prodos.	10
12.	6/11	Prodos - Family tree	10

Completed

```

1. def addnum(num1, num2):
    return num1 + num2

num1 = 10
num2 = 20
result = addnum(num1, num2)
print(result)

```

output: 30

2. concatenating list

```
test_list3 = [1, 4, 5, 6, 5]
```

```
test_list4 = [3, 5, 7, 2, 5]
```

```
test_list3 = test_list3 + test_list4
```

```
print("Concatenated list using +: ", test_list3)
```

output: Concatenated list using +: [1, 4, 5, 6, 5, 3, 5, 7, 2, 5]

```
list = ("apple", "banana")
```

3. ~~second largest element~~

```
list.add("org")
```

```
print(list) OP: ('apple', 'org', 'banana')
```

4. list of square

```
def square_list(n):
```

```
    return [i**2 for i in range(n)]
```

```
num = int(input("Enter number:"))
```

```
print(list_of_square)
```

OP: Enter no=5 list of square: [1, 4, 9, 16, 25]

```
n = int(input("Enter a number"))
```

```
num1 = 0
```

```
num2 = 1
```

```
print(num1)
```

```
print(num2)
```

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

```
    num = num1 + num2
```

```
    print(num)
```

```
    num1 = num2
```

```
    num2 = num
```


Job recommendation using all algorithms.

Domain: Career technology.

Abstract:

A job recommendation system uses machine learning to match job seekers with suitable job opportunities. By analyzing profiles and job listings, the system provides personalised job suggestions. It employs techniques like content based and collaborative filtering and uses natural language processing to understand job descriptions and resumes. The system aims to enhance job seekers experience and improve recruitment efficiency by delivering relevant job matches and adapting to user feedback.

Problem statement:

Develop a machine learning-based job recommendation system to match job seekers with suitable job opportunities. The system should analyze user profiles and job listings to deliver personalized recommendations. By utilizing algorithms like content-based and collaborative filtering, and employing natural language processing, the system aims to enhance the job search experience and increase recruitment efficiency. It should continuously adapt to user feedback and evolving data to provide relevant and timely job suggestions.

Solution:

Implement a machine learning job recommendation system that analyze job seekers profiles and job listings. Use content-based and collaborative filtering to deliver personalized job suggestions. Employ natural language processing for better understanding and matching. Continuously refine recommendations based on user feedback and interactions.

Target Audience:

1. Job seekers: People actively searching for jobs
2. Employers/Recruiters: Companies looking to hire the right candidates
3. Students/Graduates: New entrants to the job market seeking entry-level positions.
4. Career changers: Professionals transitioning to new industries or roles.
5. Freelancers/Gig workers: Individuals seeking short-term or freelance opportunities.
6. Mid-career professionals: Experienced professionals looking for advancement.
7. Tech-savvy users: Users who expect advanced, personalized recommendations.

N Queen Problem

Code:

```
def is-safe(board, row, col, n):
```

```
    for
```

AIM:

To execute the N-queen problem using Python.

def
code:

```
def is-safe(board, row, col, n):
```

```
    for i in range(col):
```

```
        if board[row][i] == 1:
```

```
            return False
```

```
    for i, j in zip(range(row, -1, -1), range(col, -1, -1)):
```

```
        if board[i][j] == 1:
```

```
            return False
```

```
    return True
```

```
def solve-n-queens_util(board, col, n):
```

```
    if col >= n:
```

```
        return True
```

```
    for i in range(n):
```

```
        if is-safe(board, i, col, n):
```

```
            board[i][col] = 1
```

```
            if solve-n-queens_util(board, col+1, n):
```

```
                return True
```

```
            board[i][col] = 0
```

```
    return False
```

```
def print-board(board, n):
```

```
    print("In solution:")
```

```
    for row in board:
```

```
        for cell in row:
```

```
            if cell == 1:
```

```
                print('Q', end='')
```

```
            else:
```



```
print(' ', end=' ')
print()
```

```
def solve_n_queens(n):
```

```
    board = [[' ' for _ in range(n)] for _ in range(n)]
```

```
    if not solve_n_queens_util(board, 0, n):
```

```
        print("No solution exists")
```

```
    return False
```

```
    print_board(board, n)
```

```
    return True
```

```
n = int(input("Enter the value of N: "))
```

```
solve_n_queens(n)
```

Output: Enter the value of N: 4

Solution:

```
Q - - - -
- - - Q -
- Q - - -
- - - - Q
- - Q - -
True
```

True. Enter the value of N: 8

Solution:

```
Q - - - - -
- - - - Q -
- - - Q - -
- - - - Q
- Q - - - -
- - - Q - -
- - - - Q -
- - Q - - -
True
```

Result:

Thus the ⁰execute the n queen problem using python is verified successfully.

Depth First search (graph)

Aim: Write a python code for depth first search (DFS)

Graph: 'A' = ['B', 'C']

'B' = ['A', 'D', 'F']

'C' = ['A', 'E', 'G']

'D' = ['B']

'E' = ['C']

'F' = ['B']

'G' = ['C']

Code:

```
def dfs(graph, start, visited = set()):
```

```
    if visited is not None:
```

```
        visited.add(start)
```

```
        print(start, end = ' ')
```

```
        for (start, end) in graph[start]:
```

```
            if neighbour not in visited:
```

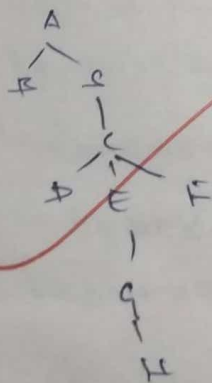
```
                dfs(graph, neighbour, visited)
```

```
dfs(graph, start, visited)
```

```
dfs(graph, 'A')
```

Output:

ABDEFCEFG



Result:

Thus the above python code for DFS is verified successfully.

Water Jug Problem using BFS

AIM: To write a python code for water jug problem using BFS.

Program:

```
from collections import deque

def min_steps(m, n, d):
    if d > max(m, n):
        return -1
    q = deque([(0, 0, 0)])
    visited = [[False] * (n+1) for _ in range(m+1)]
    visited[0][0] = True

    while q:
        jug1, jug2, steps = q.popleft()
        if jug1 == d or jug2 == d:
            return steps

        if not visited[m][jug2]:
            visited[m][jug2] = True
            q.append([m, jug2, steps+1])
        if not visited[jug1][n]:
            visited[jug1][n] = True
            q.append([jug1, n, steps+1])
        if not visited[0][jug2]:
            visited[0][jug2] = True
            q.append([0, jug2, steps+1])
        if not visited[jug1][0]:
            visited[jug1][0] = True
            q.append([jug1, 0, steps+1])

        pour1to2 = min(jug1, n-jug2)
        if not visited[jug1-pour1to2][jug2+pour1to2]:
            visited[jug1-pour1to2][jug2+pour1to2] = True
            q.append([jug1-pour1to2, jug2+pour1to2, steps+1])

        pour2to1 = min(jug2, m-jug1)
        if not visited[jug1+pour2to1][jug2-pour2to1]:
            visited[jug1+pour2to1][jug2-pour2to1] = True
            q.append([jug1+pour2to1, jug2-pour2to1, steps+1])

    return -1

if __name__ == '__main__':
    m, n, d = 4, 2, 2
    print(min_steps(m, n, d))
```


output:

4

Result: The above python code for water jug problem is executed successfully.

A* algorithm

Aim:

To write a python code for A* algorithm.

Program:

```
def xstar(start_node, stop_node)
```

```
    open_set = set(start_node)
```

```
    closed_set = set()
```

```
    g = {}
```

```
    parent[start_node] = start_node
```

```
    while len(open_set) > 0:
```

```
        n = node
```

```
        for v in open_set:
```

```
            if n == None or g[v] + heuristic[v] < g[n] + heuristic[n]:
```

```
                if n == stop_node or graph_node[n] == True:
```

```
                    else:
```

```
                        for (m, weight) in get_neighbors(n):
```

```
                            if m not in open_set and m not in closed_set:
```

```
                                open_set.add(m)
```

```
                                parent[m] = n
```

```
                                g[m] = g[n] + weight.
```

```
                    else:
```

```
                        if g[m] > g[n] + weight:
```

```
                            g[m] = g[n] + weight
```

```
                            parent[m] = n
```

```
                if m in closed_set:
```

```
                    closed_set.remove(m)
```

```
                    open_set.add(m)
```

```
            if n == node:
```

```
                print("path does not exist")
```

```
                return node
```

```
            if n == stop_node:
```

```
                print()
```

```
                while parent[n] != n:
```

```
                    path.append(n)
```

```
                    n = parent[n]
```

```
path.append(start_node)
```

```
path.remove()
```

```
print("Path found: {}".format(path))
```

```
return
```

```
open_set.remove(n)
```

```
closed_set.add(n)
```

```
print("path does not exist!")
```

```
return now
```

```
def get_neighbours(v):
```

```
    if v in graph_node:
```

```
        return graph_nodes[v]
```

```
    else:
```

```
        return now
```

```
def heuristic(n)
```

```
    if dist = (
```

```
        'A' = 11,
```

```
        'B' = 6,
```

```
        'C' = 99,
```

```
        'D' = 1,
```

```
        'E' = 7,
```

```
        'u' = 0 )
```

```
    return if dist(n)
```

```
graph_nodes = {
```

```
    'A' : [( 'B', 3 ), ( 'E', 3 ) ],
```

```
    'B' : [( 'C', 1 ), ( 'A', 9 ) ],
```

```
    'C' : now,
```

```
    'D' : [( 'D', 6 ) ],
```

```
    'E' : [( 'A', 3 ), ( 'C', 7 ) ] }
```

```
for ( 'A', 'C' )
```

Output:
path found: ['A', 'E', 'D', 'C']

~~Star~~ Result: Thus the Python program for A* algorithm is verified successfully.

Aim:

To implement A* algorithm.

PROGRAM:

class Graph:

def __init__(self, graph, heuristic):

self.graph = graph

self.heuristic = heuristic

self.solution = {}

def a_star(self, node):

print("Expanding: {node}")

if node not in self.graph or not self.graph[node]:
return

children = self.graph[node]

best_path = None

min_cost = float('inf')

for group in children:

cost = sum(self.heuristic[child] for child in group)

if cost < min_cost

min_cost = cost

best_path = group

self.solution[node] = best_path

print("Best path for {node} = {best_path} with least
{min_cost}")

for child in best_path:

self.a_star(child)

def get_solution(self):

return self.solution

graph = {

'A': [['B', 'C'], ['F']],

'B': ['E'],

'C': ['G'],

'D': ['H'],

'E': [],

'G': [],

'H': []}

heuristic = {

'A': 0, 'B': 1, 'C': 2, 'D': 4, 'E': 1, 'F': 3, 'G': 5, 'H': 7}

graph_obj = Graph(graph, heuristic)

graph_obj.no_start('A')

solution = graph_obj.get_solution()

print("solution:", solution)

output:

Expanding: A

Best path for A = ['A', 'C'] with cost 3

Expanding B

Best path for B: ['E'] with cost 1

Expanding: E

Expanding: C

Best path C: ['G']

Expanding: G

Solution = $\{A: [B, C], 'B' = [E], 'C' = [G]\}$

Result: This algorithm implemented
Successfully.

Aim: To implement a decision tree classification 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 pandas as pd
from sklearn.tree import
DecisionTreeClassifier.
data = {'Height': [152, 158, 172, 185, 167, 180, 157, 190,
                  164, 177],
        'Weight': [48, 55, 72, 83, 68, 78, 52, 90, 66, 80],
        'Gender': ['Female', 'Female', 'male', 'Male', 'Female', 'Male',
                  'Female', 'Male', 'Female', 'Male']}

df = pd.DataFrame(data)
x = df[['Height', 'Weight']]
y = df['Gender']
classifier = DecisionTreeClassifier()
classifier.fit(x, y)
height = float(input("Enter height (in cm) for prediction: "))
weight = float(input("Enter weight (in kg) for prediction: "))
random_values = pd.DataFrame([height, weight],
                              columns=['Height', 'Weight'])
predicted_gender = classifier.predict(random_values)
print(f"Predicted gender for height {height} cm and weight {weight} kg is {predicted_gender}")
```


output:

Enter height (in cm) for prediction: 169

Enter weight (in kg) for prediction: 61

Predicted gender for height 169.0cm and
weight = 61.0kg - female.



Result: Thus the above Python code executes
successfully.

Aim:

To implement of K-Mean clustering technique using python language.

Explanation:

- import KMean from sklearn cluster.
- Assign X and Y.
- Call the function K-Mean()
- Perform scatter operation and display the output.

Code:

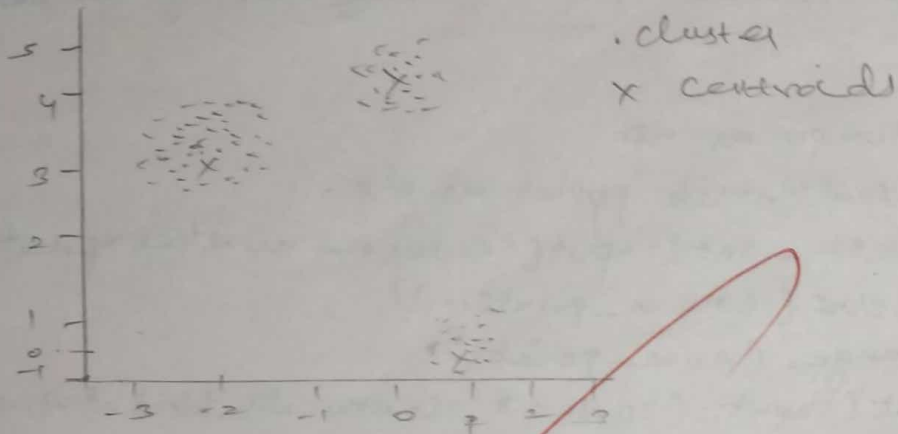
```
import numpy as np
import matplotlib.pyplot as plt
num_points = int(input("Enter the number of data points:"))
x = np.zeros((num_points, 2))
for i in range(num_points):
    x = float(input("Enter x-coordinates for datapoint {i+1}:"))
    y = float(input("Enter y-coordinates for datapoint {i+1}:"))
    x[i] = [x, y]
num_clusters = int(input("Enter the number of clusters:"))
def kmeans(x, num_clusters, max_iter=100):
    centroids = x[np.random.choice(x.shape[0], num_clusters, replace=False)]
    for _ in range(max_iter):
        distances = np.linalg.norm(x[:, np.newaxis] - centroids, axis=2)
        labels = np.argmin(distances, axis=1)
        new_centroids = np.array([x[labels == k].mean(axis=0) for k in range(num_clusters)])
        if np.all(centroids == new_centroids):
            break
        centroids = new_centroids
    labels, centroids = kmeans(x, num_clusters)
    print("cluster labels:\n", labels)
    print("Centroids:\n", centroids)
    plt.figure(figsize=(8,6))
    plt.scatter(x[:, 0], x[:, 1], c=labels, cmap='viridis', marker='o', label='Data points')
    plt.scatter(centroids[:, 0], centroids[:, 1], c='red',
```

```

s=200, marker='x', label='centroid')
plt.title('K-means clustering (from scratch)')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend(['F'])
plt.grid(True)
plt.show()

```

output:



~~Result:~~ Thus the K-means clusters algorithm for
 technique has been implemented
 an application using python successfully.

Aim: To implement of Artificial Neural Networks for an Application using PYTHON REGRESSION.

Algorithm:

Generate data: create example data for training

Prepare data: Normalize or scale the data.

split data: divide data into training and testing sets.

Build Model: Define the neural network structure

Compile Model: select option and loss function.

Train Model: Fit the model to the training data over Multiple epochs

Evaluate Model: ^{Use} Test model performance using testing set.

Predict: Use the Model to make Predict as new data.

Visualize: Compare Predictions with actual results.

Code:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
import matplotlib.pyplot as plt

x = np.random.rand(1000, 3) # 1000 samples, 3 features.
y = 3 * x[:, 0] + 2 * x[:, 1] ** 2 + 1.5 * np.sin(x[:, 2] * np.pi) + np.random.normal(0, 0.1, 1000) # Nonlinear relationship
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)

model = Sequential()
model.add(Dense(10, input_dim=x_train.shape[1], activation='relu'))
model.add(Dense(1, activation='linear'))
model.compile(optimizer=Adam(learning_rate=0.01), loss='mean_squared_error')
```

```

history = model.fit(x_train, y_train, epochs=100, batch_size=32,
                    validation_split=0.2, verbose=1)
y_pred = model.predict(x_test)
mse = np.mean((y_test - y_pred.flatten())**2)
print("mean squared error: ", mse)
plt.figure(figsize=(12,6))
plt.plot(history.history['loss'], label='training loss')
plt.plot(history.history['val_loss'], label='validation loss')
plt.title("Training and validation loss")
plt.xlabel('Epoch')
plt.ylabel('loss')
plt.legend()
plt.show()

```

output:

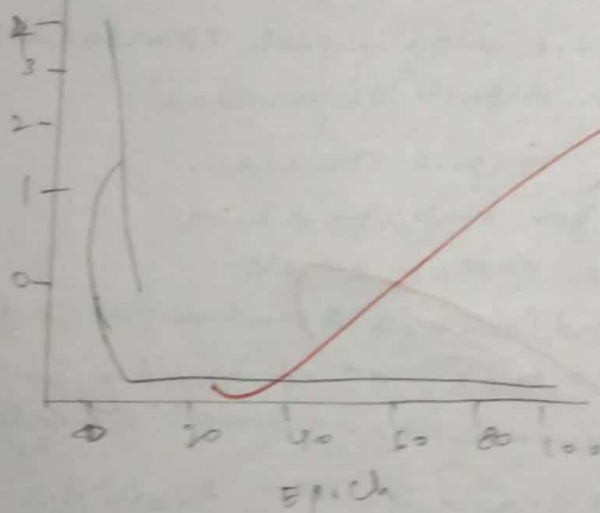
Epoch 1/100

20/20

...

Epoch 100/100

20/20



~~Output:~~

Result: Thus the artificial neural networks for regression has been implemented successfully.

Aim:

To implement minimax algorithm.

PROGRAM:

```
import math
def minimax(depth, node, index, is_maximizes, scores, height):
    if depth == height:
        return scores[node_index]
    if is_maximizes:
        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(num_leaves):
    return math.ceil(math.log2(num_leaves))
scores = [3, 5, 6, 9, 1, 2, 0, -1]
tree_height = calculate_tree_height(len(scores))
optimal_score = minimax(0, 0, True, scores, tree_height)
print("The optimal score is : {optimal_score}")
```

Output: The optimal score is 5

Result:

Thus the minimax algorithm is executed successfully.

Aim: To learn Prolog terminologies and write basic program.

TERMINOLOGIES:

- ① Atomic terms: They are usually string made up of lower & uppercase letters, digits and the underscore, starting with a lowercase letter. eg: dog, ab_c-321
- ② variables: They are strings of letters, digits and the underscore, starting with a capital letter or underscore.
- ③ compound terms: compound terms are made up of a Prolog atom and a number of arguments enclosed in parentheses and separated by commas.
eg: is-bigger(elephant, x). f(g(x, -), 7)
- ④ Fact: A fact is a predicate followed by a dot.
eg: bigger-animal(whale). life-a-beautiful
- ⑤ Rules: A rule consists of a head and body.
eg: 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(golanda)
playsAirGuitar(jody)
party
query 1: ?- woman(mia)
query 2: ?- playsAirGuitar(mia)
query 3: ?- party
query 4: ?- concert
```

Output:

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

```
?- playsAirGuitar(mia)
false
```

```
?- concert
```

Error = Unknown procedure. Concert/0

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, britney)

married(x, y) \equiv likes(x, y), likes(y, x)

friends(x, y) \equiv -likes(x, y) \wedge likes(y, x)

output:

? - likes(dan, x)

x = sally

? - married(dan, sally)

true

? - married(john, britney)

false.

KB4:

food(burger)

food(sandwich)

food(pizza)

lunch(sandwich)

dinner(pizza)

meal(x) \equiv -food(x)

output:

? - food(pizza)

true

? - meal(x), lunch(x)

x = sandwich

PROG:

owns(jack, car(bmw))

owns(john, car(chery))

owns(olivia, car(civic))

owns(jane, car(hrv))

~~owns~~ sedan(car(bmw))

sedan(car(civic))

truck(car(hrv))

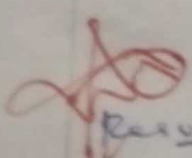
output:

? - owns(john, x)

x = car(chery)

? - owns(john, -)

-true.

 Result: Thus the basic prolog programs has been executed successfully.

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

Source code:

Knowledge base:

male(peter).

male(john).

male(chris).

male(kevin).

female(betty).

female(jenny).

female(lisa).

female(helen).

parent of (chris, peter)

parent of (chris, betty)

parent of (helen, peter)

parent of (helen, betty)

parent of (kevin, chris)

parent of (kevin, lisa)

parent of (jenny, john)

parent of (jenny, helen).

/ * Rules. * /

/ * son, parent. /

* son, grandparent *

father(x,y) :- male(y), parent of (x,y)

mother(x,y) :- female(y), parent of (x,y)

grandfather(x,y) :- male(y), parent of (x,z), parent of (z,y)

grandmother(x,y) :- female(y), parent of (x,z), parent of (z,y)

brother(x,y) :- male(y), father(x,z), father(w,z), w \= y

sister(x,y) :- female(y), father(x,z), father(w,z), w \= y

Output:

male (Peter)

true

father (Chris, Peter)

true

father (Chris, Betty)

false

mother (Chris, x)

x = Betty

brother (Chris, Helen)

false.

Key: The Prolog for family tree program
has been executed successfully.