INDEX

NAME: Adhesh. M STD.: _____ SEC.: _____ ROLL NO.: _____ SUB.: _____

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
1.	24-7-24	Python Excercises	10	
2.	7-8-24	Problem Statement and Domain	10	
3.	4-9-24	N-Queens Problem	10	
4.	4-9-24	DFS Search	10	
5.	11-9-24	A* Algorithm	10	
6.	18-9-24	AO* Algorithm	10	
7.	25-9-24	Implementation of KMeans Clustering Technique	10	
8	7-10-24	Artificial Neural Networks	10	
9.	9-10-24	Water Jug	10	
10.	16-10-24	Prolog Introduction	10	
11.	23-10-24	MinMax	10	
12.	23-10-24	Prolog Family Tree	10	
		Solver		
13.	25-10-24	Decision Tree	10	

completed

Exercise - 1

1) Factorial

```
def factorial(n):
    if n==0:
        return 1
    else:
        return n*factorial(n-1)
num = int(input("Enter num: "))
print(factorial(num))
```

Output:
Enter a num: 5
Factorial of 5 is 120

2) Sum of digits

```
def Sum_digits(n):
    Sum = 0
    while(n!=0):
        Sum = Sum + int(n%10)
        n = int(n/10)
    return Sum
num = int(input("Enter num: "))
print(Sum_digits(num))
```

Output:
Enter a num: 567
18

3) Palindrome

```
def isPalindrome(s):
    return s == s[::-1]
word = input("Enter a String: ")
ans = isPalindrome(word)
if ans:
    print("Yes")
else print("No")
```

4) Odd or Even

```
num = int(input())
if num%2 == 0:
    print("Even")
else: print("odd")
```

Output: 7
odd

5) Armstrong Number

```
def is_armstrong_num(num):
    Sum = 0
    while(num>0):
        digit = num%10
        Sum += digit**3
        num //= 10
    return Sum
num = int(input())
b = num
if is_armstrong_num(num) == b:
    print("Yes")
else:
    print("No")
```

Output:
23
No

6) Swapping of two integers

```
def swap(a,b)
    a = a-b
    b = b+a
    a = b-a
    return a,b
a = int(input())
b = int(input())
print(swap(a,b))
```

Output: 10,20
20,10

7) Area of a Rectangle

```
l = int(input())
b = int(input())
print(l*b)
```

Output:

```
10
20
200
```

8) Fibonacci Series

```
def fib(n):
    a, b = 0, 1
    for i in range(n):
        print(a, end=' ')
        a, b = b, a+b
n = int(input())
print(fib(n))
```

Output:

```
4
0 1 1 2 3
```

9) Prime or not

```
def is_prime(n):
if n<=1:
    print False
for i in range(2, int(n**0.5)+1)
    if n%i == 0:
        return False
return True
n = int(input())
print(is_prime(n))
```

Output:

```
7
True
```

10) Greatest common Divisor

```
def gcd(a,b):
while b:
    a, b = b, a%b
return a
n = int(input())
m = int(input())
print(gcd(n,m))
```

Output:

```
16
32
16
```

Stock Market Prediction

Domain:

Financial Analytics or Financial Forecasting.

Analyzing & predicting financial markets and stock prices. It involves techniques from quantitative finance, data science and machine learning and is a subset of business analytics and financial data analysis.

Problem Statement:

The objective of this project is to develop a predictive model so to forecast future stock values based on historical price data.

Target Audience:

The predictive capability of this system assists investors, analysts and common people involved in stock market investment.

Algorithm used : LSTM (Long - Short term Memory)

LSTM networks are a type of ~~recurrent~~ Recurrent Neural Network (RNN) architecture designed to learn and model temporal sequences and dependencies. They are effective for time series forecasting such as stock price prediction due to their ability to capture long-term dependencies and manage sequential data effectively.

Objectives

Core objectives of this project are

1. Data Collection and Preparation : Gather and preprocess historical stock price data
2. Model Development : Build an LSTM network to predict future prices.
3. Model training : Train LSTM model on historical
4. Model Evaluation : Assess models accuracy and performance

5. Model Optimization: Improve the model based on evaluation results.

6. Deployment & Prediction: Implement the model for real time predictions

7. Documentation and Reporting:

Document the project process and outcome

Domain

The Domain of a stock price prediction using LSTM networks falls within several intersecting fields.

1. Financial Analytics: Analysing historical stock price data
2. Quantitative Finance: Applying mathematical models to financial data for predictions
3. Time Series Analysis: Analyzing sequential data to identify patterns & forecast future values.
4. Machine Learning: Using algorithms, particularly LSTM networks, to predict stock prices based on historical data
5. Data Science: Collecting, preprocessing and analyzing financial data to develop predictive models.
6. Algorithmic Trading: Automating trading decisions

Exercise N-QUEEN

```
def print_solution(board):
    N = len(board)
    for i in range(N):
        row = ""
        for j in range(N):
            if board[i][j] == 1:
                row += "Q"
            else:
                row += "."
        print(row)
    print("\n")

def is_safe(board, row, col):
    n = len(board)
    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 = len(board)
    if col >= N:
        return True
    for i in range(N):
        if is_safe(board, i, col):
            board[i][col] = 1
            if solve_n_queens_util(board, col+1):
                return True
            board[i][col] = 0
    return False

def solve_n_queens(N):
    board = [[0] * N for _ in range(N)]
    if solve_n_queens_util(board, 0) is False:
        print("Solution don't exist")
        return False
    print_solution(board)
    return True

N = 8
solve_n_queens(N)
```

Result: The above program code has been executed successfully & output is verified

Depth First Search

Aim:

To implement Depth First Search Algorithm

Program code:

```
def dfs(graph, start, visited = None):
    if visited is None:
        visited = set()
    visited.add(start)
    print(start)
    for neighbor in graph[start]:
        if neighbor not in visited:
            dfs(graph, neighbor, visited)
    return visited
graph = {
    'A': ['B', 'C'],
    'B': ['A', 'D', 'E'],
    'C': ['A', 'F'],
    'D': ['B'],
    'E': ['B', 'F'],
```

```
    'F': ['C', 'F']
}
print ("DFS Starting from node 'A':")
dfs(graph, 'A')
```

Output

```
DFS starting from node 'A':
A
B
D
E
F
C
{'A', 'B', 'C', 'D', 'E', 'F'}
```

Result: The above code has been executed successfully & output is verified

A* Code

Aim
To implement A* algorithm

Program Code:

```
import heapq
def heuristic(a, b):
    return abs(a[0] - b[0]) + abs(a[1] - b[1])
def astar(grid, start, end):
    open_list = []
    heapq.heappush(open_list, (0 + heuristic(start, end), 0,
                               start))
    g_cost = {start: 0}
    came_from = {}
    directions = [(0,1), (1,0), (0,-1), (-1,0)]
    while open_list
        _, current_g, current = heapq.heappop(open_list)
        if current == end:
            path = []
            while current in came_from:
                path.append(current)
                current = came_from[current]
            path.append(start)
            return path[::-1]
```

```
for dx, dy in directions:
    neighbor = (current[0] + dx, current[1] + dy)
    if (0 <= neighbor[0] < len(grid) and
        0 <= neighbor[1] < len(grid[0]) and
        grid[neighbor[0]][neighbor[1]] == 0):
        tentative_g = current_g + 1
        if neighbor not in g_cost or tentative_g
            came_from[neighbor] = current
            g_cost[neighbor] = tentative_g
            g_cost = tentative_g + heuristic(neighbor, end
            heapq.heappush(open_list, [g_cost,
                                       tentative_g, neighbor))
return None

grid = [ [0, 0, 0, 0, 0],
         [0, 1, 1, 1, 0],
         [0, 0, 0, 1, 0
         [0, 1, 0, 0, 0],
         [0, 0, 0, 0, 0] ]
start = (0, 0)
end = (4, 4)
path = astar(grid, start, end)
print("Path found: ", path)
```

Result: The above code has been executed successfully and output is verified

AO* Code

Aim:

To implement AO* Algorithm

Program Code:

```
import heapq
def heuristic(a, b):
    return abs(a[0] - b[0]) + abs(a[1] - b[1])
def ao_star(grid, start, goals):
    directions = [(0, 1), (1, 0), (0, -1), (-1, 0)]
    open_list = [(0 + min(heuristic(start, goal) for
                goal in goals), 0, start)]
    g_cost, came_from = {start: 0}, {}

    while open_list:
        _, current_g, current = heapq.heappop(open_list)

    if any(current == goal for goal in goals):
        path = []
        while current in came_from:
            path = []
            path.append(current)
            current = came_from[current]
        return path + [start][::-1]

    for dx, dy in directions:
        neighbour = (current[0] + dx, current[1] + dy)
        if 0 <= neighbor[0] < len(grid)
            and 0 <= neighbor[1] < len(grid[0])
```

```
                    and grid[neighbor[0]][neighbor[1]] ==
                tentative_g = current_g + 1
                if neighbor not in g_cost or tentative_g <
                    g_cost[neighbor] = tentative_g
                    f_cost = tentative_g + min(abs(neighbor[0]
                                                   goal[
                    heapq.heappush(open_list, (f_cost, tentative_g, neighbor))
                    came_from[neighbor] = current

        return None

def get_position(prompt):
    while True:
        try:
            x, y = map(int, input(prompt).split())
            return (x, y)
        except ValueError:
            print("Please enter valid integer coordinates
                                Separated by a space.")
def main():
    grid = [
        [0, 0, 0, 0, 0],
        [0, 1, 1, 1, 0],
        [0, 0, 0, 1, 0],
        [0, 1, 0, 0, 0],
        [0, 0, 0, 0, 0]
    ]
```

```
start = get_position("Enter the starting position (x,y): ")
goal = get_position("Enter the goal position (x,y): ")
path = ao_star(grid, start, [goal])
if path:
    print("Path found:", path)
else:
    print("No path found")
if __name__ == "__main__":
    main()
```

Output:

```
Enter the starting position (x y): 1 1
Enter the goal position (x y): 3 4
Path found: [(3,4), (3,3), (3,2), (2,2),
             (2,1), (1,1)]
```

Result:

The above code has been executed successfully and output is verified.

Implementation of Decision Tree Classification Techniques

Aim:

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

Explanation:

- Import tree from sklearn
- Assign values for x & y
- Call the function predict for predicting on the basis of given random values for each given feature
- Display the output

Program Code:

```
from sklearn.tree import DecisionTreeClassifier
x = [[170, 65, 40], [160, 55, 38], [180, 80, 42],
     [175, 75, 74], [158, 52, 37]]
Y = [1, 0, 1, 1, 0]
clf = clf.fit(x, y)
Prediction = clf.predict([[160, 60, 39]])
```

Output: Predicted Gender Female

Result: Thus the python program to implement a decision tree classification technique for gender classification is executed successfully.

Implementation of K Means Clustering Technique

Aim:

To implement a K-Means clustering technique using python language

Explanation:

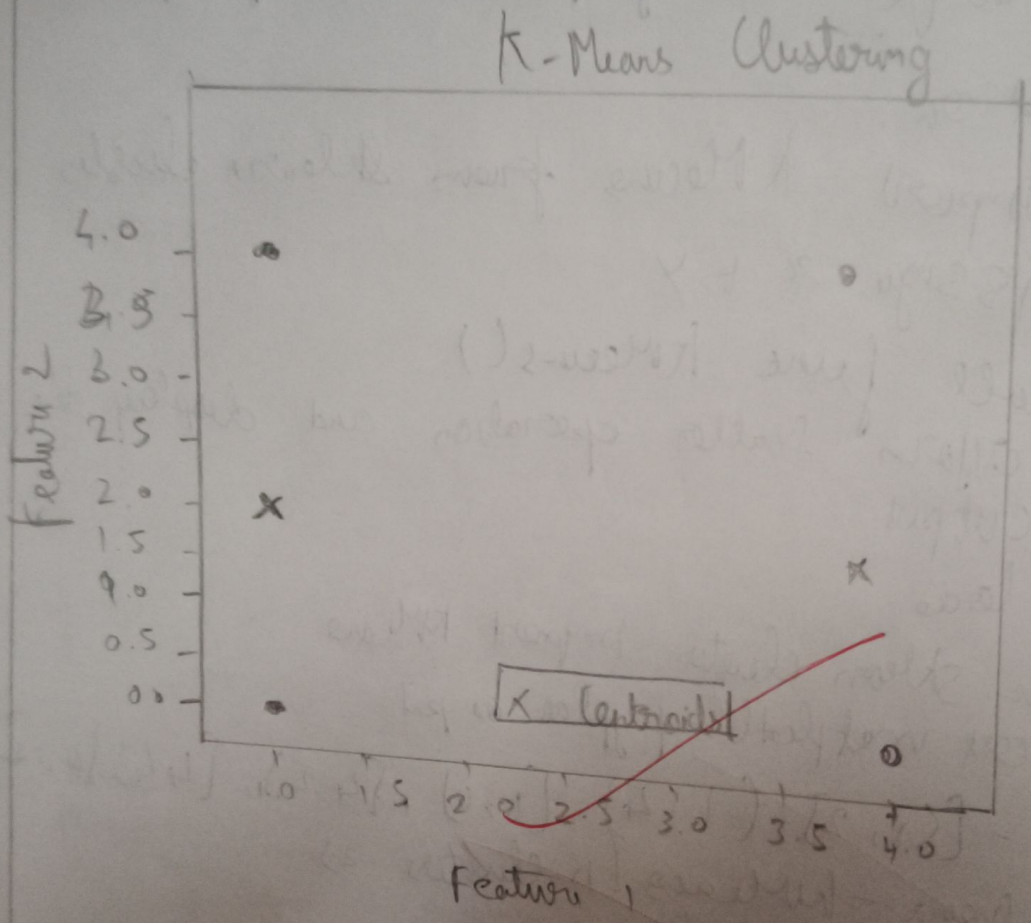
- Import KMeans from sklearn.cluster
- Assign x & y
- Call func KMeans()
- Perform scatter operation and display the output

Program code

```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
x = [(1, 2), (1, 4), (1, 0), (4, 2), (4, 4), (4, 0)]
kmeans = KMeans(n_clusters=2)
y = kmeans.labels
centers = kmeans.cluster_centers
for i, label in enumerate(y):
    plt.scatter(x[i][0], x[i][1], color='blue'
        if label==0 else 'green')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
```

```
plt.title('K-Means Clustering')
plt.legend()
plt.show()
```

Output:



Result:
Thus the Python program to implement K-Means Clustering Technique is executed successfully.

Implementation of Artificial Neural Networks for an Application in Regression

Aim:

To implement artificial neural network for an application in regression using python.

Program Code:

```
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split

x = np.array([[1200,3,20],[1500,4,15],[1800,3,5],
              [2000,5,10],[1600,4,25]])
y = np.array([200000,250000,270000,300000,
              240000])
x_train, x_test, y_train, y_test = train_test_split()
Scaler = standardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
model = Sequential()
model.add(Dense(1))
```

```
model.fit(x_train, y_train, epochs=10,
          batch_size=5, verbose=1)

predictions = model.predict(x_test)

for i in range(min(5, len(predictions))):
    print(f"Predicted: {predictions[i][0]:.2f},
          Actual: {y_test[i]}")
```

Output:

Predicted: 0.06, Actual: 250000

Result: Thus the python program to implement artificial neural networks for an application in regression is executed successfully

Water Jug

Program Code

```
def fill_4_gallon(x, y, x_max, y_max):
    return (x_max, y)

def fill_3_gallon(x, y, x_max, y_max):
    return (x, y_max)

def empty_4_gallon(x, y, x_max, y_max):
    return (0, y)

def empty_3_gallon(x, y, x_max, y_max):
    return (x, 0)

def pour_4_to_3(x, y, x_max, y_max):
    transfer = min(x, y_max - y)
    return (x - transfer, y + transfer)

def pour_3_to_4(x, y, x_max, y_max):
    transfer = min(y, x_max - x)
    return (x + transfer, y - transfer)

def dfs_water_jug(x_max, y_max, goal_x, visited=None):
    start = (0, 0)
    if visited is None:
        visited = set()
    stack = [start]
```

while stack,

```
state = stack.pop()
x, y = state
if state in visited:
    continue
visited.add(state)
print(f"visiting state: {state}")
if x == goal_x:
    print(f"Goal reached: {state}")
    return state
next_states = [fill_4_gallon(x, y, x_max, y_max),
    fill_3_gallon(x, y, x_max, y_max),
    empty_4_gallon(x, y, x_max, y_max),
    empty_3_gallon(x, y, x_max, y_max),
    pour_4_to_3(x, y, x_max, y_max),
    pour_3_to_4(x, y, x_max, y_max)]
for new_state in next_states:
    if new_state not in visited:
        stack.append(new_state)
return None
```

```
x_max = 4
y_max = 3
goal_x = 2
def water_jug(x_max, y_max, goal_x):
```

Output:

visiting state : (0, 0)
visiting state : (0, 3)
(3, 0)
(3, 3)
(4, 2)
(4, 0)
(1, 3)
(1, 0)
(0, 1)
(4, 1)
Goal reached : (2, 3)
(2, 3)

Result:
Thus, the python program to Solve water jug problem is executed and verified successfully

Prolog

Aim: To learn prolog terminologies and basic programs

Terminologies

Atomic Terms: Strings made up of lowercase, uppercase letters digits and underscore
Starts with a lowercase letter
dog ab_c_321

Compound Terms: are made up of PROLOG atom and a no of arguments enclosed in parantheses and seperated by commas.

is_bigger (elephant, x)
+ (9 (X, _), 7)

Facts: A fact is a predicate followed by a dot

bigger_animal (whale)
life_is_beautiful

Rules: A rule consists of a head (a predicate) and a body (a sequence of predicates seperated by commas) is_smaller (X, Y) :- is_bigger (Y, X)

CODE :

KB 1 :

Woman (mia)
Woman (jody)
Woman (yolanda)
Playsin (jody)
party

Query :

? Woman (mia)
- true
? Playsinguitar
- false
! party
- true
? concert
- Error

KB 2

Happy (yolanda)
listens2music (mia)
listens2music (yolanda) :- happy (yolanda)
playsAisguitar (mia) :- listens2music (mia)
PlaysAisGuitar (yolanda) :- listens2music (yolanda)

Output

? - playsAisguitar (mia)
- true
? playsAisguitar (yolanda)
- true

KBS

likes (dan, sally)

likes (sally, dan)

likes (john, brittney)

married (X, Y) - likes (X, Y) ; likes (Y, X)

friend (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

?. drink (sandwich)

false

KBS

owns (jack, car(bmw))

owns (john, car (chevy))

owns (olivia, car (civic))

owns (jane, car(chevy)).

Sedan (car(bmw))

Sedan (car (civic))

Output:

?. = owns (john, X)

X = car (chevy)

? owns (john, -)

true

?- owns (who car (chevy))

Result: Thus the experiment to learn about PROLOG and execute basic programs was done successfully.

Family Tree Using PROLOG

Code:

```
/* FACT:: */
male(peter)
male(john)
male(chris)
male(kevin)

female(betty)
female(jenny)
female(lisa)
female(ruby)

parent of(chris, peter)
parent of(chris, betty)
parent of(chris, peter)
parent of(ruby, betty)
parent of(kevin, chris)

/* RULE :: */
/* Son, grandparent */
father(x, y) :- male(y), parent(x, y)
mother(x, y) :- female(y), parent of(x, y)
brother(x, y) :- male(y), father(x, z), father(y, w).
Sister(x, y) :- female(y), father(x, z), father(y, w)
```

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

Output:

? parentOf(kevin, x)

x = chris

? father(x, chris)

x = kevin

? sister(x, chris)

false

Result: Thus the PROLOG program to implement and create family tree was successfully completed

Minmax Problem

Code:

```
def minmax (depth, nodeIndex, isMaximizingPlayer, scores,
                                        targetDepth):
    if depth == targetDepth:
        return scores [nodeIndex]
    if isMaximizingPlayer:
        return max (minimax (depth+1, nodeIndex
* 2, False, scores, targetDepth))
    else:
        return min (minimax (depth+1, nodeIndex * 2,
True, scores, targetDepth),
                minimax (depth+1, nodeIndex * 2+1,
True, scores, targetDepth))
if __name__ == "__main__":
    scores = [3, 5, 2, 9, 12, 5, 23, 23]
    targetDepth = 3
optimal value: minmax(0, 0, True, scores, targetDepth)
print ("The optimal value for you is: ", optimalvalue)
```

Output:

The optimal value for the game is : 12

Result: Thus the minimax program is implemented and output is verified successfully