INDEX

S.NO	TITLE	~~DATE~~ marks	SIGN
1.	PYTHON EXERCISES USING GOOGLE COLAB	10	
2.	PROBLEM STATEMENT AND DOMAIN	10	
3.	N-QUEENS PROBLEM	10	
4.	DFS SEARCH	10	
5.	A* Algorithm	10	
6	AO* Algorithm	10	
7.	Water Jug Problem	10	
8.	IMPLEMENTATION OF DECISION TREE CLASSIFICATION TECHNIQUES	10	
9.	IMPLEMENTATION OF K MEANS CLUSTERING TECHNIQUE	10	
10.	IMPLEMENTATION OF ARTIFICIAL NEURAL NETWORKS FOR AN APPLICATION IN REGRESSION	10	

11.	MINMAX ALGORITHM	10	
12.	INTRODUCING PROLOG	10	
13.	PROLOG - FAMILY 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 a number:"))
print("Factorial of", num, "is", factorial(num))
```

output:

Enter a number: 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 a number: "))
print(sum_digits(num))
```

output:

Enter a number: 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) ArmStrong Number

```
def is_armstrong_number(number):
    sum=0
    while(number>0):
        digit = number%10
        sum+=digit**3
        number//=10
    return sum

number=int(input())
b=number
if is_armstrong_number(number)==b:
    print("Yes")
else:
    print("No")
```

output:

23

No

5) 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

6) odd or even

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

output :

7

odd

7) Area of a rectangle

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

output:

10
20
200

8) 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

9) 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

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 SYSTEM

at is Financial Analysis ?

This domain encompasses various aspects related to analyzing and predicting financial markets and stock prices. It involves techiques 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 using Long-Short term memory (LSTM) networks to forecast future stock prices based on historical price data.

Intended Audience :-

This predictive capability can assist investors and analysts in making informed decisions about buying or selling stocks.

Algorithm used : LSTM (Long-Short Term Memory)

Type of Recurring Neural Network (RNN) architecture designed to learn and model temporal sequences and dependencies. They are particularly effective for stock price prediction, due to their ability to capture long-term dependencies and manage sequential data effectively.

Core Objectives :-

* Data collection and Preparation
* Model Development
* Model Training
* Model Evaluation
* Model optimization
* Deployment and Prediction
* Documentation and Reporting.

Domain

The Domain of a stock price prediction using LSTM networks falls within several interesting fields. Here's a detailed breakdown of the domain.

Financial Analytics: Analysing historical stock price data to forecast future trends.

Quantitative Finance: Applying Mathematical models to financial data for predictions.

Time Series Analysis: Analyzing sequential data to identify patterns and forecast future values.

Machine Learning: using algorithms, particularly LSTM networks, to predict stock prices based on historical data.

Data Science: Collecting, preprocessing, and analyzing financial data to develop predictive models.

Algorithmic Trading: Automating trading decisions based on stock price predictions.

EXERCISE - N-QUEENS

AIM:

To implement N-Queens problem

CODE:-

```
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(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(board, col+1):
            return true
return false

def solve(N):
    board = [[0]*N for _ in range(N)]
    if solve_n_queens(board, 0) is False:
        return False
    print_solution(board)
    return true

N=4
solve_n_queens(N)
```

OUTPUT:-

. . Q .
Q . . .
. . . Q
. Q . .

RESULT -

Thus the above code has been executed successfully & output is verified

EXERCISE - DFS

AIM:

To implement DFS Search Algorithm

CODE:

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

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

dfs(graph, 'A')
```

OUTPUT:

A
B
C
D
{'A', 'B', 'C', 'D'}

RESULT

Thus the above code has been executed successfully & output is verified.

A* Algorithm

Aim:- To implement A* Algorithm

```
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<=neighbo
            <len(grid[0]) and grid[neighbor[0]][neighbo
            ==0):
                tentative_g = current_g+1
                if neighbor not in g_cost or tentative_g<
                g_cost[neighbor]:
```

```
        g_cost[neighbor] = tentative_g
        f_cost = tentative_g + heuristic(neighbor, end)
        heapq.heappush(open_list, (f_cost, tentative_g, neighbor))
        came_from[neighbor] = current

    return None

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 = tuple(map(int, input("Enter start node: ").split()))
end = tuple(map(int, input("Enter end node: ").split()))

    path = astar(grid, start, end)
    if path:
        print("Path found:", path)
    else:
        print("No path found")

if __name__ == "__main__":
    main()
```

output:

Enter start node : 1 2

Enter end node : 3 4

Path found : [(1,2), (2,2), (3,2), (3,3), (3,4)]

Result :-

Thus the above code has been executed successfully and output is verified.

A* Algorithm

AIM:-

To implement A* Algorithm

CODE:-

```
import heapq
def heuristic(a,b):
    return abs(a[0]-b[0]) + abs(a[1]-b[1])

def a_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.append(current)
                current = came_from[current]
            return path + [start][::-1]

        for dx,dy in directions:
            neighbour = (current[0] + dx, current[1] + dy)
            if 0<= neighbour[0] < len(grid) and 0 <= neighbour[1]
            < len(grid[0]) and grid[neighbour[0]][neighbour[1]]
            == 0:
                tentative_g = current_g + 1
                if neighbour not in g_cost or tentative_g < g_cost
                [neighbour]:
                    g_cost[neighbour] = tentative_g
                    f_cost[n] = tentative_g + min(heuristic(neighbor, g
```

```
        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("Invalid")

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]]

    start = get_position("Enter the starting position (x,y)
    goal = get_position("Enter the goal position (x,y)

    path = a_star(grid, start, [goal])
    if path:
        print("Path found:", path)
    else:
        print("No path found")

if __name__ == "__main__":
    main()
```

Output :-

Enter the starting position : 1 1

Enter the goal position : 3 4

Path found : [(3,4), (3,3), (3,2), (2,2), (2,1), (1,1)]

Result :-

Thus the above code has been executed successfully & output is verified.

Group 1

9/10/24 (Team 8)

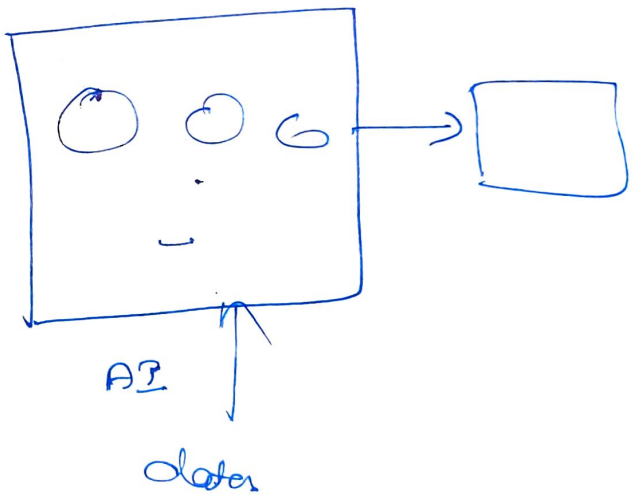
Stock price prediction

- Add keywords in abstract.
  Add more content.

- Introduction.
  - Statistics.
- Literature survey.

SIM
- Architecture ~~diag~~.

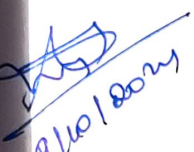
i/p Process o/p



Explanation

block dig.

- Implementation (Alg.)
  - mathematical calculation
  - formula.
  - Block diagram.
- Results.
  - Graphs.
  - Tables



Water Jug Problem

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_3to_4 (x, y, x_max, y_max)
    transfer = min (x, y_max - y)
    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

dfs_water_jug(x_max, y_max, goal_x)
```

Output:

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

Result:

Thus, the python program to solve water jug problem is executed and verified successfully

IMPLEMENTATION OF DECISION TREE CLASSIFICATION TECHNIQUES

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.

Program Code:

```
from sklearn.tree import decisiontreeclassifier
x = [[170,65,40], [160,55,38], [180,80,42], [175,75,41]
     [158,52,37]]
Y = [1,0,1,1,0]
clf = clf.fit(x,Y)
prediction = clf.predict([[165,60,39]])
print("predicted Gender:", "Male" if prediction[0]
      == 1 else "Female")
```

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 k-Means from sklearn.cluster
- Assign x and y.
- Call the function kmeans()
- Perform scatter operation and display the output.

Program Code:

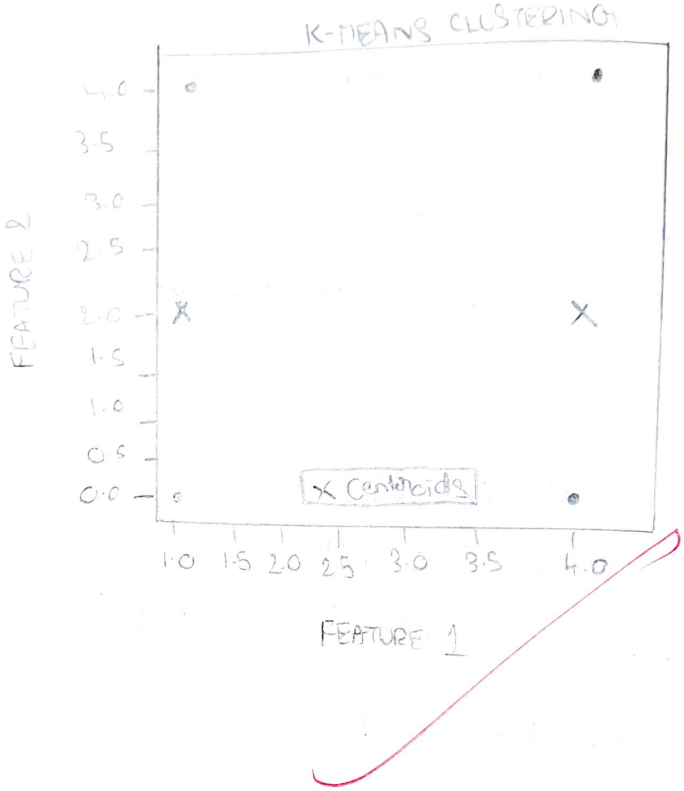
```
from sklearn.cluster import kMeans
import matplotlib.pyplot as plt
x = [[1,2], [1,4], [1,0], [4,2], [4,4], [4,0]]

kmeans = kmeans (n_clusters=2)
kmeans.fit(x)
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.scatter (centers[:,0], centers[:,1],
        color = 'red', marker = 'x', s=100,
        label = 'centroids')

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 networks 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
from sklearn.preprocessing import Standard Scaler

X = np.array ([[1200, 3, 25], [1500, 4, 15], [1800, 3, 30],
               [2000, 5, 16], [1600, 4, 25]])

Y = np.array ([200000, 250000, 270000, 300000, 240000])

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 (32, input_dim = X.shape[1],
                  activation = 'relu'))

model.add (Dense (1))
model.compile (optimizer = 'adam', loss = 'mean_squared_error')
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.

MINMAX ALGORITHM

AIM:

To implement MINMAX ALGORITHM

PROGRAM:-

```
import math
def minmax (depth, node_index, is_maximizes, scores,
 height):
if depth == height:
    return scores [node_index]

 if is_maximizes:
    return max (minmax (depth+1, node_index*2, false,
                scores, height), minmax (depth+1, node_index*
                2+1, false, scores, height))

  else:
    return min (minmax (depth+1, node_index*2, true,
                scores, height), minmax (depth+1, node_index*2+1,
                true, scores, height))

def calculate_tree_height (num_leaves):
    return math.ceil (math.log 2 (num_leaves))

scores = [3,5,6,9,1,2,0,-1]
tree_height = calculate_tree_height (len (scores))
optimal_score = minmax (0,0, true, scores, tree_height)
print (f"The optimal score is : {optimal_score}")
```

output:-

The optimal score is :5

Result:-

Thus the minmax algorithm is implemented successfully.

PROLOG INTRODUCTION

Aim:

To learn PROLOG terminologies and write basic programs.

TERMINOLOGIES:

Atomic terms: Strings made up of lowercase, uppercase letters and underscore. Starts with a lowercase letter

dog ab_C_321

Variables: Strings starting with capital letters

Dog Apple_420

compound terms: are made up of PROLOG atom and a no. of arguement enclosed in paranthesis and separated by commans

is_bigger(elephant, X)

f(g(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 separated by commas)

is_smaller(X, Y) :- is_bigger(Y, X).

aunt(Aunt, Child) :- sister(Aunt

CODE:

KB1:

woman (mia)

woman (jody)

woman (yolanda)

playsAirGuitar (jody)

party

Query:

? woman(mia)

-true

? playsAirGuitar(mia)

-false

? party

-true

? concert

-error

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, birthday)

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

friends (X, Y) :- likes (X, Y), likes (Y, X)

output:-

? listens (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) :- 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 (jan, car (chevy)).

sedan (car (bmw)).

sedan (car (civic)).

truck (car (chevy)).

output:

```
? owns(john, X)
X = car(cheng)
? owns(john, _)
true
? - owns(who, car(cheng))
who - john
```

Result:

Thus the experiments to learn about PROLOG and execute basic programs was done successfully.

PROLOG - FAMILY TREE

AIM:

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

CODE:

```
/* FACT :: */
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)

/* RULE :: */

/* 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,
grandmother(X,Y) :- female(Y), parent of (X,Z), parent of(Z,
brother(X,Y) :- male(Y), father(X,Z), father(Y,Y), Z
sister(X,Y) :- female(Y), father(X,Z), father(Y,Y), Z
```

output:

? parent of (kevin, x)

x = chris

? father (x, chris)

x = kevin

? sister (x, chris)

false

RESULT:

Thus the PROLOG program to implement and execute family tree was successfully completed.