1)

```
HT - list = (1,2,3,4,5)

first - element - my - list [0]

last - element - my - list [-1]

my list [2] = 10

my - list - remove (10)

for element in my - list
    print (element)
```

output:-

```
1
2
4
5
0
```

2)

```
Import numpy as np

my - array = np - array ([1,2,3,4,5])

first - element - my - array (0)

last - element = my array [-1]

my - array [2] = 10

my . array = np . append (my . array)

my . array - np . delete (my - array , n.p (array

for element in array - print element
```

output:-

[5, 4, 3, 2, 1]

3) Finding second smallest element in an array program:-

```
list 1 = [10, 20, 4, 70, 64, 7]
list 2 = [list (set (list 1))]
list 2. sort ()
print ["second smallest element is :" list2[1]
```

output.

second smallest element is : 7

4) Finding second largest

program :

```
list 1 = [10, 20, 4, 70, 64, 7]
list 2 = list (set (list 1))
list 2. sort ()
print ("second large element is : list 2 [-2]
```

Output :

second largest element is : 64

5) list of square :

```
def square . list (n)
    return [i ** 2 for i in range (n)]
num = int (input ("Enter a number :"))
print ("list of squares ; square. list(nu
```

Output :

Enter a no : 5

list of squar : 1, 4, 9, 16, 25.

College Recommendation system for admissions.

Problem statement:

After completing the school students are unaware about the college and which college they should select for cutoff and cast category.

domain: college admit guide.

End users: school students.

Abstract: students and their family would receive tailored college recommendation under the proposed based on their academic, career objective and 12th grade and cutoff scores, admission trends, and other relavent data. complete about the collaborating colleges and offers as well as streamlined methods for data analysis and retrieval and user-friendly interfaces. the major objectives of the approch are to decrease the number of less-than-idea

college selection admission made by TNEA counseling program participants, enhance admission process openness and encourage data-driven decision-making. the system uses complex algorithm and a large quantity of data its help students and their family make decisions that support their academy and career aspirations. the technology integrates advanced analytics with intuitive user interface to empower student its make informed decision about their future education path. the system seeks to enhance both the college selection process and the result for student participating in the TNEA counseling program. In conclusion, the proposed approach is a ~~major~~ improvement in the college admission process since it makes use of data-driven insights to help students make decisions.

Objectives :-

Our prediction technology provides a comprehensive solution to assist Tamil Nadu students in making educated judgments regarding their college admission. Our research seamlessly integrates recommendation algorithm and regression modeling to address the challenges of anticipating current-year cutoffs and offering individualized college options based on a student's subjects of interest. Our solution user evaluation, stringent data protection policies, and an easy-to-use interface to improve decision-making for both students and education institutions. This study represents a major step toward data-driven and effective college admission, with the potential for further breakthroughs and broader applications in the education sector.

4/9/24

N Queens Problem :-

Aim: To write a python program to place the Queen on safe def is safe (board, row, do column

Program:

```
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, n):
    print("\n solution :")
    for row in board:
        for col == 1:
            print("Q", end=" ")
        else:
            print(".", end=" ")
    print()

def solve_n-queens(n):
    board = [[0 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: 8.

Some Solution:

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

Enter the value of N: 4

Solutions

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

Result:

Thus the program has been executed successfully.

1/9/24 Depth first search.

Aim:

To find Depth first search algorithm.

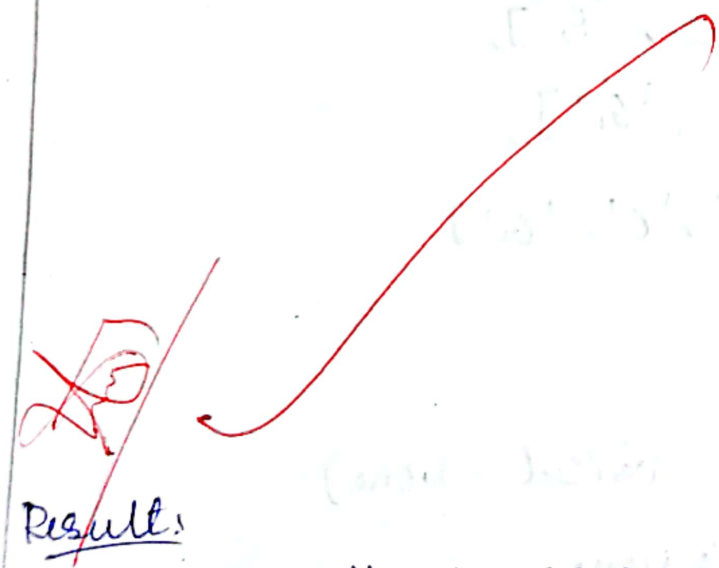
program:

```
graph = {
    'A' : ['B', 'S'],
    'B' : ['A'],
    'C' : ['D', 'E', 'F', 'S'],
    'D' : ['C']
    'E' : ['C', 'H'],
    'F' : ['C', 'G'],
    'G' : ['F', 'S'],
    'H' : ['E', 'G'],
    'S' : ['A', 'C', 'G']
}

def dfs(graph, node, visited = None):
    if visited is None:
        visited = []
if node not in visited:
```

```
        visited.append(node)
        for neighbor in graph[node]:
            if neighbor in graph
            dfs(graph, neighbor, visited)
    return visited

visited_nodes = dfs(graph1, 'A')
print(visited_nodes)
```

Output:

```
['A', 'B', 'S', 'C', 'D', 'E', 'H', 'G', 'F']
```



Result:

Thus the program has been executed successfully.

11/9/24 A* Algorithm.

Aim:

To implement A* Algorithm

program:-

```
from heap import heappop, heappush

class node:
    def __init__(self, position, parent = none);
    self.position = position.
    self.parent = parent
    self.g = 0
    self.h = 0
    self.f = 0
    def __eq__(self, other)
        return self.position == other.position.
def a.star (start, goal, grid)
start.node = Node (start)
goal_node = node (goal)
open_list = [ ]
closed_list = set()
while open_list:
    current_node = happen (open_list)
    if current_node = goal_node:
```

```
path = []
while current_node:
    path.append(current_node.position)
    current_node = current_node.parent
return path[::-1]
neighbours = [(0,-1),(0,1),(-1,0),(1,0)]
for n in neighbours:
    neighbour_position = (current_node.position[0]+n[0],
                          position[1]+n[1])
if 0 <= neighbour_position[0] < len(grid)
continue.
neighbour_node.g = current_node.g + 1
neighbour_node.h = abs(neighbour_node.position[0]-
                       node_position[1] - goal_node.
if all(neighbour_node != open_node for open_n
    in open_list)
heap.push(open_list, neighbour_node)
return None.
grid = [
    [0,1,0,0,0]
    [0,1,0,1,0]
    [0,0,0,1,0]
    [0,1,1,1,0]
```

```
start - (0,0)
goal - (4,4)
path = a - star (start, goal, grid)
print ('path found:', path
```

output:

path found = [(0,0), (1,0), (2,0), (3,0), (4,0), (4,1)
(4,2), (4,3), (4,4)]

Result:

thus the A* algorithm has been implemented and executed successfully.

output:-

Expanding : A

Best path for A : ['B', 'C'] with cost:

Expanding : B.

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

Expanding : E

Expanding : C

Best path C : ['Gr']

expanding : Gr

Solution . {A = [B, C], 'B', ['E'], 'C', ['Gr']}

Result:

thus AO* algorithm implemented successfully.

Implementation of Decision tree classification

Aim:

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

Program:-

```
Import pandas as pal
from skill as. tree import decision tree
data: {' Hlight': [152, 155, 172, 185, 167, 180,
                   164, 177]

      'weight'. [45, 57, 72, 85, 68, 78, 22, 88]

Gender: ['Female', 'male', 'Male', Female
        , 'Mal', 'Femal', 'mal', 'Female'].

    df = pataframe (data)
    Y = df ['Hlight', 'weight']
    Y = df ['Gender']

classifier - Decision tree classifier
classifier = fit (X, Y)
height = float (input ("Enter height (in kg)
                       for prediction");
weight = float (input ("Enter weight (input) for
                       prediction: "))
```

```
print(f'predicted gender for height {
cm and weight {weight} kg :
{predicted_gender[0]}')
```

output:

predicted Gender : Female

Result:

Thus the Python program to implement a decision tree classification technique gender classification is executed succesfully

Aim:

To implement artificial neural networks for an application in regression using Python.

program:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_spli
from keras.layers import Dense
import matplotlib.python as plt
# Generating synthetic dataset for demo
x = np.random.rand(1000,3)
y = 3*np.pi+np.random.normal(0,0.1,1000)
x_train, x_test, y_train, y_test = train_test_
split(x,y, test_size=0.2, random_state=42)
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
model = Sequential()
model = Sequential()
model.add(Dense('input_dim= y_train
                slope[1] activation='relu'
```

```
g.pred = model.predict(x_test)
plt.figure(figsize=(12,6))
plt.plot(history.history['loss'], label=
                'training loss')
plt.plot(history.history['val_loss'], label=
                'validation loss')
plt.xlabel('Epoch')
plt.xlabel('Epoch')
plt.xlabel('loss')
plt.legend()
plt.show()
```

output:

predicted : 0.06 _ Actual : 25000

Result:

Thus the python program to implement Artificial neural network for an application is regression is executed successfully.

Aim:

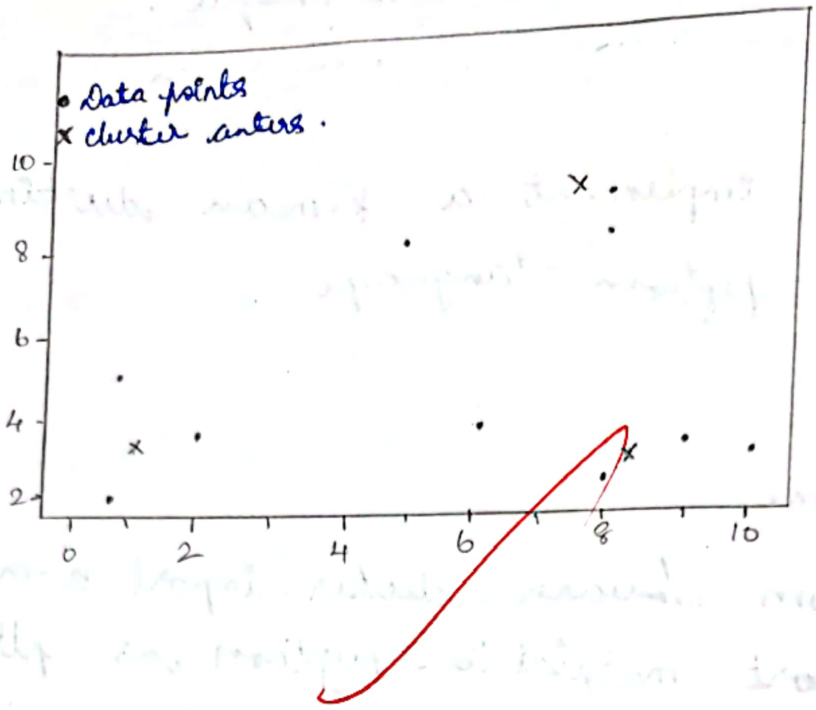
To implement a k-mean clustering technique using python language.

Algo

program:-

```
From sklearn.cluster import k-means
import matplotlib.pyplot as plt
x = [[1,2]], [1,4], [1,0], [4,2], [4,4]
k means: k mean.clusters.centers
for i, bal in enumsal(y):
plt.scatter(x[i][0] x[i][1] colour='blue'
 if label.==0 else='green']
plt.scatter(centa[::0], centa,[1]
 colour = 'red; marker='x'[:,0], centa[:,1],
                                    S=200
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.legend()
plt.show()
K means = k means(n-cluster=3)
k means.fit(x)
centers = k means.cluster - center
plt.legend()
plt.show()
```

output:



Result:

The program has been executed and output was verified successfully.

Ex No:
Date: 15/10/24

Aim:

To implement artificial Neural Network for an application in Regression using Python

program:

```
import numpy as np.
from tensorflow_keras.models import sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import standard
from sklearn.metrics import mean_squared_error
np.random.seed (42)
x = np.random.rand (1000, 3)*10
y = x[:,0] * 300 + x[:,1]*500 + x[:,2] * 100
x train, x_test, x_train, y_test = train_test
split (x, y, test_size = 0.2 random_state = 42)
scaler = standard scaler();
x_train = scaler fit transform x_train
x_test = scaler. transform (x_test)
model = sequential();
model.add (Dense (64, input dim = 'mse'))
model .add (Dense (1))
model.compile (optimizer 'adam', loss = 'mse',
metrics = ['mae'],
loss, mae, = model .evaluate (x_test, y_test,
print=('Mean squared error on test
```

outputs:

Mean Absolute error on test data: 717.3563842771437

Mean squared error on test data: 729649.3500195

Result.

the program has been executed and output was verified successfully.

23/10/24

Aim:

To learn prolog terminologies and write basic programs.

Terminologies:

1. Atomic terms:-

atomic terms are usually string made up of lower- and uppercase letter, digit and the underscore, starting with a lowercase letter.

eg: dog
ab_c_321

2. variable

variables are string of letters, digits, and the underscore starting with a capital letter or a underscore.

eg:
Dog
Apple_420

3. compound terms:-

compond terms made up of a prolog atom and a number of argument (PROLOG terms atoms, number variable, or ther compound terms) enclosed in parathesis and separated by commas

eg: is_bigger (elephant, x)
f(g(x, _), 7)

4. Facts:-

A fact is a predicate followed by a dot

eg:

bigger_animal (whale)

dif_is_beautiful.

5. Rules:

A rule consist of head (a predicate) and a body (a sequence of predicates by comma)

eg:

is_smaller (x, y) is_bigger (y, x)

aunt (Aunt, child)

Source code:-

KB1:

woman(mia)

woman (Jody)

woman (yolanda)

plays_air_guitar (Jody)

party

Query 1: ?= woman (mia)

Query 2: ?- plays_air_guitar (mia)

Query 3: ? - party

Query 4: ?- concert

Output:

?- woman (mia)

true

? - plays_air_guitar(mia)

False

? - party

true.

KB2:

happy(yolanda)

listens2music(mia)

listens2music(yolanda) :- happy(yolanda)

plays_air_guitar(mia) :- listens2music(mia)

plays_air_guitar(yolanda) :- listens2music(yolanda)

output:

?- plays_air_guitar(mia)

True

?- play_air_guitar(yolanda)

True.

?- ■

KB3:

likes(dan, sally)

likes(sally, dan)

likes(John, britney)

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

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

output:

?- likes(dan, X)

X = sally

?- married(dan, sally)

true

?- married(John, britney)

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))
own (John, car (chevy))
own (olivia, car (chevy))
sedan (car (bmw))
sedan (car (civic))
truck (car (chevy))

output:

? - own (John, x)
x = car (chevy).

? - owns (John)
true
? - owns (who, car (chevy))
false

?- owns(Jane,X), truck(X)
X = car (cheevy)

Result

the program has been executed and output was verified succesfully.

Prolog - Family

Aim:

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

Source code:

```
male (peter).
male (John)
male (chris)
mal (Kevin)

female (betty)
female (Jeny)
female ( lisa)
female ( helen)

parent of (chris, peter)
parent of (chris, betty)
parents of (helen, betty)
parent of ( Kevin, chris)
parent of (Jeny, John)
parent of (Jeny, helen)
/* Rules :*/
/* Son, parent
* Son. grand parent */
father (X,Y) :- male (X), parent of (X,Y)
mother (X,Y) :- female (Y), parent of (X,Y)
```

grand mothr (x,y) :- male (y), parent of (x,z),
parent of (z,y)

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

brothr (x,y) :- male (y), fathr (x,z), fathr (y,w), z ==

sister (x,y) :- femal (y), fathr (x,z) fathr (x,w), z == w

Result:-

Thus the Prolog problem its implem
ented family tree was successfully completed.

Exp No:

Date: 10/11/24

Unification and Resolution

Aim:

To execute program based on resolution.

Ex:1

Let us see for below the prolog progra instruction take after squring.

Facts : likes (John, Jen)

likes (Jane, John)

upon asking the query first prolog to search matching term in facts in Top enmes for Likes predicte with the argume and it can match this (John)

Ex 2: at the prolog query prompt where you write follow query.

ans (x, car (bmw)) = own (y, car(c)

will get answer : x, y, c = bmw

Here ans (x, car (bmw) and own(y,c)

2nd argument with car predicate inside the brent are same both side a(x,y) and c in initials, written as {bmw / c} and called with location

i) Select trees that contain containing terms

ii) scan those trees clauses

iii) come out the containing terms.

for example we have following statements.

1, it is a pleasant dog you will statement picture.

2) if you are doing strabery picking you are happy.

3) strawbery-picking < pleasant

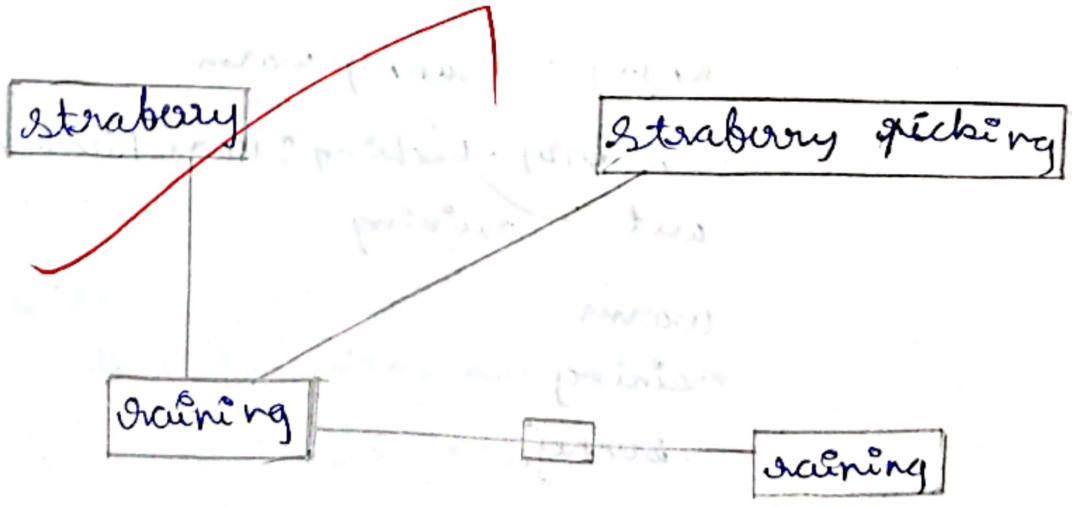
4) happy < strabery - picking)

5) pleasent happy.

How → see the high on right.

when you write above new clause in order on inputs from. we have pleasent shyapp on

happy = pleasant.



Part (I) English sentence

1) if it is sunny and warm day you will enjoy.

2) if it is warm and pleasant day you will enjoy.

strawbury picking:

3) if it is raining then no strawbury picking

4) if it is raining then we will get

5) it is warm day

6) if is sunny.

part (II) propositional statement

1) enjoy < sunny < warm
2) strawbury, picking < warm
3) warm
4) raining
5) sunny.

source code:-

enjoy :- sunny warm

strawbary - picking : warm pleasend.

ant : raining

warm

raining

sunny.

output:

? not strawbary picking
tree

? injoy

? - wet
tree

Result:

thus the program was base one verification and resolution has been executed successfully.