	DATE	TITLE		SIGNATURE
1)	31/7/24	Basic Python Programs	9	
2)	7/8/24	Domain: Sentiment Analysis	9	
3)	4/9/24	N Queens	9	
4)	4/9	Depth first Search	10	
5)	11/9	A* Algorithm	10	
6)	18/9	Ao* Algorithm	10	
7)	25/9	Decision Tree	10	
8)	9/10	k-means Clustering	10	
9)	16/10	Artificial Neural Network	10	
10)	23/10	MinMax	10	
11)	30/10	Introduction to Prolog	10	
12)	6/11	Prolog Family Tree	10	

Completed

Ex - 3 N Queens Date: 4/9/24

Write a program to solve 8 Queens problems

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

def col_dashes (bs, c):
    for i in range (c):
        if share_diagnol (1, bs[i], c, bs[c]):
            return True;
        return false

def has_dashes (the_board):
    for col in range (1, len (the_board)):
        if col_dashes (board, col):
            return True;
    return False

def main (E):
    import random
    rng = random. Random ()
    bd = list (range(8))
    num_found = 0
    tries = 0
    result = []
    while num_found <10:
        rng. shuffled
        tries += 1
        if not has_dashed (bd) and bd not in result:
            print ("Found Soln {0} in {1} tries", format(bd, tries))
            tries = 0
            num_found += 1
            result. append (list (bd))
      print (result).

main ()
```

Output:

```
4
Found solution [1,3,0,2] in 4 tries
Found solution [2,0,3,1] in 47 tries

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

8
8
Found solution [2,5,1,4,7,0,6,3] in 115 tries
Found solution [4,1,3,5,7,2,0,6] in 2 tries
Found solution [3,1,6,4,0,7,5,2], in 430 tries
Found solution [5,2,0,7,4,1,3,6] in 317 tries
Found solution [2,6,1,7,5,3,0,4] in 1849 tries
Found solution [4,2,0,6,1,7,5,3] in 56 tries

Q . . . . . . .
. . . . . . Q .
. . . . Q . . .
. . . . . . . Q
. Q . . . . . .
. . . Q . . . .
. . . . . Q . .
. . Q . . . . .
```

RESULT:

The 8 Queens problems has been excecuted Succenfully.

Ex-4 DFS Date:- 4/9/24

Solve any problem using Depth first search

CODE:

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

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

dfs (graph, 'A')
```

OUTPUT:

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

RESULT:

Thus DFS program is executed succenfully.

Water Jug using dfs:

```
def fill_4.gallon(a,y,x_max, y_max):
      return (x_max, y);
def fill_3_gallon(x,y, xmax, y_max):
      return (x, y_max)
def empty_4_gallon (x,y, x_max, y_max):
      return (a,y)

def empty_3_gallon (x,y, xmax, ymax):
      return (x,0)

def   pour_4_to_3 (x,y, xmax, ymax):
      trarsfer = min (x, y_max_y)
      return (x-trarsfer, y+trarsfer)

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

def dfs_water_jug(xmax,ymax, goal_x, visited=None, start=(0,0)):
    if visited is None:
        visit = set()
        Stack = [start]
        while stack:
            state = stack.pop()
            x,y = state
            if state in visited:
                 continue
            visited.add(state)
            print(f"visited state : {state}")
            if x==goal_x:
                 print(f"goat reached. {state}")
                 return state
```

```
next states = [fill_4_gallon(x, y, xmax, ymax), fill_3_gallon(x, y),
               empty_4 gallon (x, y, xmax, ymax),
               empty_3_gallon (x, y, xmax, ymax),
               pour_4 to_3 (x, y, xmax, ymax),
               pour_3_to_4 (x, y, xmax, ymax),]

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_waterjug (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)
  Goal reached : (2,3)
      (2,3)
```

RESULT:

Thus the water Jug problem using DFS is executed successfully.

ate:- 11/9/24

EX-5- Implement A* Algorithm

CODE:

```
def asta(start_node, stop_node):
    open_set = set(start_node)
    closed_set = set()
    g = {}
    parents = {}
    g[start_node] = 0
    parent [startnode] = startnode
    while len[openset] > 0:
        n = none
        for v in openset:
            if n=none or g[v] + heuristic(v) < g[m] +
                                          heuristic(m)
        n = v
        if(n==stopnode or graph_nodes == none:
            pass
        else
            if g[m] > g[n] + weight:
                g[m] = g[n] + weight
                parents [m] = n
            if min closedset;
                closed_set_remove (m)
                open_set. add (m)
            if n== None:
                print ("path does not exist")
                return None
        if n== stopnode:
            path []
            while parents [n] ! =n :
                path. append (n)
                n= parents[n]
            path. append (start_node)
            path. reverse ()
            print( "path found:{}".format(path))
            return path
        open_set.remove(n)
        closed_set.add(n)
        print("Path doesnot exist")
        return none

def get_neighbours(v):
    if v in graphnodes:
        return graph_nodes[v]
    else
        return node

def heuristic (n):
    dist = { 'A': 11, 'B': 6, 'C': 99, 'D': 1, 'E': 7, 'G': 0}
     return h_dist[n]


graphNodes = {
        'A' : [('B',2), ('E', 3)], 'B' : [('C', 1', ('G', 9)],
        'C' : None , 'E' : [('D', 6)], 'D' : [('G', 1)]}

astaa('A',G)
```

OUTPUT :

path found : ['A', 'E', 'D', 'G']

RESULT :

Thus the A* algorithm is executed successfully.

Date: 16/10/24 Ex-9 ARTIFICIAL NEURAL NETWORKS

Aim

To implement artificial neutral networks for an application in regression using python

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. layer import Dense
from keras optimizes import Adam
import matplotlib pyplot as plt.

np random. seed (42)
x = np. random. rand (1000, 3)
y = 3* x [:,0] + 2*x [:,1]**2 + 1.5 + np.sin [x [:,2].pi]
x_train = scaler. fit_transform (x-train)
y_test = scaler. transform (x-test)
model = sequential()
model. add (Dense (10, input_dim = x_train_shape[1], activation = 'relu')
model.add (Dense (10, activation = 'relu'))
model. add (Dense (1, activation = 'linear'))
modelcompile ( optimizer = Adam (learning_rate = 0.01), loss =
                                'mean_squared error')

history = model.fit (x_train, y_train, epochs = 100,
                        batchsize = 32, validation_split = 0.2)

y_pred = model.predict (x.test)
mse = np.mean (y-test - y-pred. flatten())**2)
plt. figure ( figsize = (12, 6))
plt.plot (history.history ['loss'], label = 'Training Loss')
plt.plot (history. history ['val_loss'], label = 'Validation Loss')
```

Date: 25/9/24 EX-7-DECISION TREE

AIM

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

CODE:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import traintest-split
from sklearn_tree import Decisiontree classifier
from sklearn.metrics import accuracy-score,
      classification_report, confusion_matrix,
import matplotlib.pyplot as plt
from sklearn import tree

data = {
  'Height' : [150,160,170,180,155,165,175,185, 140,145],
  'weight' : [50, 60,70, 80, 53, 65, 75, 85, 45, 50],
  'Age' : [25, 30, 35, 40, 28, 32, 37, 42, 24, 29],
  'Gender' : ['Fenale', 'Female', 'Male', 'Male', 'Female', 'Female',
           'male', 'Male', 'Female', 'Female'] }
 pf = pd. Dataframe(data)
 df ['Gender'] = df ['Gender'] map({female': 0, 'Male': 1})

 x = df [['Height', 'weight', 'Age']]
 y = df ['Gender']
 x_train, x_test, y_train, y_test = traintest_split
                (x, y, test_size=0.3, random_state=42)

 clf = Decisiontree Classifier ()
 clf. fit ( k_train, y_train)
 y_pred = clf. predict (x_test)
 accuracy = accuracy_score(y_test, y_pred)
 conf_matrix = confusion_matrix (y_test, y_pred)
  class_report = classification_report (y_test, y_pred, zerodiv=0)
```

```
printf(f"Accuracy: {accuracy}')
print("confusion matrix : \n", conf_matrix)
print('Classification Report : \n'', class_report)
plt.figure(figsize=(12,8))
tree.plot tree (clf, feature_names = x.colums,
                  classNames = ['Female', 'male'], filled = true)
plt.title ("DecisionTree for Gender classification)
plt show()
```

RESULT:

Thus the decision tree is executed sucessfully.

Date:-9/10/24 EX-8 - K-MEANS

AIM:

to implement a k-means clustering thechique using python

PROGRAM.

```
import numpy as np
Inport matplotlib pyplot as plt
from sklearn, cluster import k Means
from sklearn. datasets import make_blobs
 x, y_true = mak_blobs (n_samples=300, centers = 3 , cluster_std=0.60,
             random_ state =0)
 k=3
 k_means = k_means (n_clusters =k, random_state =0)
 y_means = kmeans. fit_predict (x)
 plt . figure (figsize= (8,6))
 plt. scatter (x[ :,0], [:, 1], c=y_kmeans, S=30, Cmap='viridis',
               label= 'clusters' )
 Center_ ers = kmeans. cluster_ centers.
 plt. scatter (centers [:,0), centers [:, 1], c='red', S=200,
             alpha= 0.75 , marker= 'x', label='centroids')
```

RESULT.

Thus the k-means algorithm is executed successfully

Date: 30/10/24 11) INTRODUCTION TO PROLOG

Aim:

To learn PROLOG terminologies and write basic programs

Terminologies

1) Atomic Terms

usually strings made up of lower_end and uppercaseletters digits and the undersore starting with a lower case

Eg: dog
abc_321

2) Variables

Strings of letter, digits, and undersscore, starting with capital letter or an underscore

Eg: Dog
Apple_420

3) Compound Terms

Made up of a Prolog atom and a number of arguments enclosed in parenthsis and separated by comma

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

4) Facts:

predicate. followed by a dot

Eg: bigger_animal (whale)
life_is_beautiful

5) Rules

consist of head and a 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 (yolanda)
playsAirGuitar(Jody)
Porty

Query1 :?- woman(mia) - True
Query2 :?- playsAirGuitar( mia) - False
Query 3: ? - party - True
Query 4: ?- concert. procedure concert doesnt exist.

KB2
happy (yolanda)
listens2misi (mia)
listens2music(yolanda) :- happy (yolanda)
playsAirGuitar(mia) :- listens2music (mia)
playsAirGuitar(yolanda) :- listens2music(yolanda)
Query1 : ? plays AirGuitar (mia) True
Query 2: ? plays AirGuitar (yolanda) False

KB3
like (dan, sally)
likes( sally, dan)
likes ( john, brithney)
married (x,y) :- likes (x,y), likes(y,x)
friends (x,y) :- likes (x,y) ; likes(y,y)
Query1 : ? likes (dan, x)
    x - sally
Query 2: ? married (dans, sally) false
Query 3: ? married (john, brittney) false
```

KB4

```
food (burger)
food (sandwhich
food (pizza)
lunch (sandwich)
 dinner (pizza)
 meal (x) :- food (x)
 Query 1? meal (x) lunch (x)
      x= sandwich
 Query2:? food (pizza) =true
 Query 3:? dinner(sandwich) false
```

KB5

```
owns (jack, car (bmw))
 owns (john, car (chevy))
 owns (olivia, car (civil))
 owns (jane, car (chevy))
sedan (car (bmw))
 sedan (car (civic))
 truck (car (chevy))
  Query 1:? own (john, x)
      x= car (chevy)
  Query 2:? owns (john_ _)
      true
  Query 3:? owns (who, car (chevy))
      who = john
  Query 4:? owns((jane, x), sedan (x)) false
```

RESULT:-

Thus the prolog programs are executed succesfully

Date:- 6/11/24 EX-12-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 (petch)
male ( john)
male (chris)
male (kevin)

fmale (betty)
female (jeny)
female (lisa)
female (helen)

parent-of (chris, peter)
parent of (chris, betty)
parent of ( helen, peter)
parent of ( helan, betty)
parentof ( kevin, chris)
parent of (kevin, liso)
parentof (jeny, john)
parentof (jeny, helen)

/* Rules :: */
/* Son, parent
 * Son, grandparent */
father (x, v) :- male (v), parentof (x, y)
```

OUTPUT :-

```
X = chris
Y = pear
mother (x, y) :- female ( y) , parent (x,y)
```

OUTPUT :-

```
X = chris.
Y = betty
grandfather (x, Y) :- male (Y) , parent of (x, z) , parentof (z, y)
```

OUTPUT :-

```
X = kevin.
Y = peter
Z = chris
grandmother (x, y) :- female (Y), parent of (X, Z) , parentof (Z, Y)
```

RESULT :-

Thus the program was executed succesfully and output is rectified.