

**MALAD KANDIVALI EDUCATION SOCIETY’S**

**NAGINDAS KHANDWALA COLLEGE OF COMMERCE, ARTS & MANAGEMENT STUDIES & SHANTABEN NAGINDAS KHANDWALA COLLEGE OF SCIENCE**

**MALAD [W], MUMBAI – 64**

AUTONOMOUS INSTITUTION

(Affiliated To University Of Mumbai)

Reaccredited ‘A’ Grade by NAAC | ISO 9001:2015 Certified

**CERTIFICATE**

**Name: Ms. Zeenat Hafeez Fazale Rab**

**Roll No: 578 Programme: BSc IT Semester: V**

This is certified to be a bonafide record of practical works done by the above student in the college laboratory for the course **PRINCIPLES OF ARTIFICIAL INTELLIGENCE PRACTICAL (Course Code: 2151UITPA)** for the partial fulfilment of Fifth Semester of BSc IT during the academic year 2021-22.

The journal work is the original study work that has been duly approved in the year 2021- 22 by the undersigned.

|  |  |  |
| --- | --- | --- |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  External Examiner |  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Mrs. Elizabeth Leah George (Subject-In-Charge) |
| Date of Examination: | (College Stamp) |  |

**Name: Zeenat**

**Class: T.Y. B.Sc. IT Sem- V Roll No: 578**

**Subject: PRINCIPLES OF ARTIFICIAL INTELLIGENCE PRACTICAL[Course Code: 2151UITPA]**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr No** | **Name** | **Date** | **Sign** |
| 1 | Implement Breadth First Search and Depth First Search algorithms. | 30/06/2021 |  |
| 2 | To implement Hill Climbing Algorithm in Travelling Salesman Problem (TSP) | 07/07/2021 |  |
| 3a | To implement Constraint Satisfaction Problem. (CSP). | 14/07/2021 |  |
| 3b | To implement Minimax algorithm | 22/07/2021 |  |
| 4a | To implement the working of decision tree based ID3 algorithm. | 28/07/2021 |  |
| 4b | To implement the working of Naive Bayes Algorithm | 04/08/2021 |  |
| 5 | To implement the Bayesian Classifier using the Medical data. | 11/08/2021 |  |
| 6 | To apply Markov Property to generate Donald’s Trump’s speech by considering each word used in the speech and for each word, create a dictionary of words that are used next. | 14/08/2021 |  |
| 7 | Prediction Algorithm - Use of different packages on dataset of Cat and Non-Cat images | 18/08/2021 |  |
| 8 | Neural Representation of AND and OR Logic Gates Perceptron. | 23/08/2021 |  |
| 9 | Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. | 25/08/2021 |  |
| 10 | Implementation of basic neural network model with 4 activation functions on Pima Indians onset of diabetes dataset. | 27/08/2021 |  |

**INDEX**

**A.I Journal**

**Google Colab Link:** [**https://colab.research.google.com/drive/1kx2fQ-NvedFtb05ITteRVziiPhTm8OGM?usp=sharing**](https://colab.research.google.com/drive/1kx2fQ-NvedFtb05ITteRVziiPhTm8OGM?usp=sharing)

**Practical 1**

**Colab link:** [**https://colab.research.google.com/drive/1kx2fQ-NvedFtb05ITteRVziiPhTm8OGM#scrollTo=Ui7dmws2OF7R**](https://colab.research.google.com/drive/1kx2fQ-NvedFtb05ITteRVziiPhTm8OGM#scrollTo=Ui7dmws2OF7R)

**Aim:** Implement Breadth first search and Depth first search

**Theory:**

**Breadth First Search(BFS)**

BFS is a traversing algorithm where you should start traversing from a selected node (source or starting node) and traverse the graph layerwise thus exploring the neighbour nodes (nodes which are directly connected to source node). You must then move towards the next-level neighbour nodes.

As the name BFS suggests, you are required to traverse the graph breadthwise as follows:

1. First move horizontally and visit all the nodes of the current layer

2. Move to the next layer

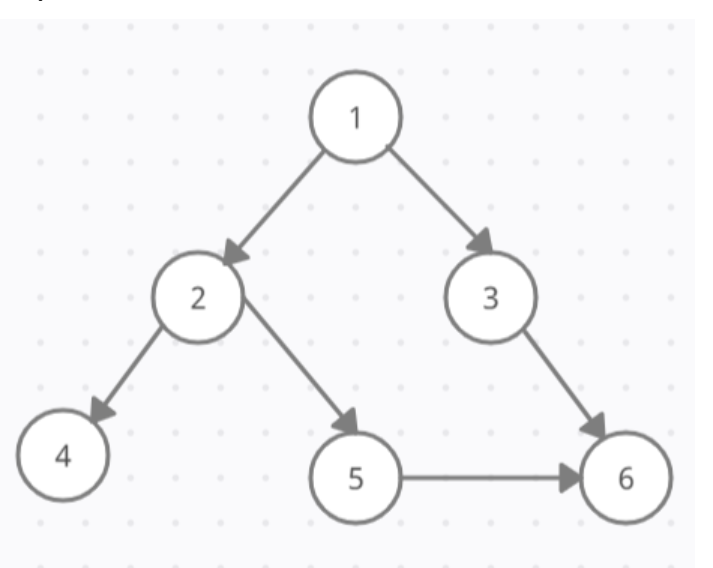
implemented Queue (FIFO)

Give always optimal solution

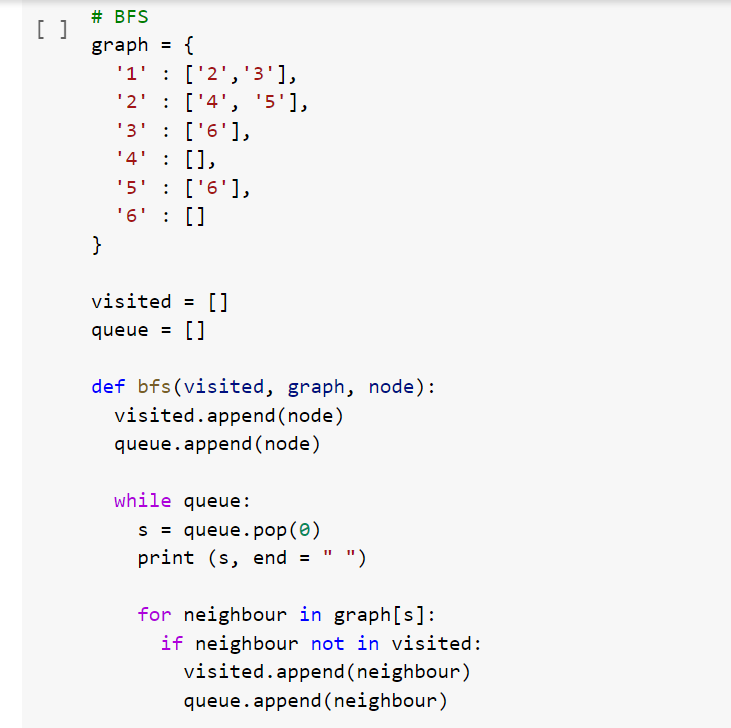
provide shallowest path

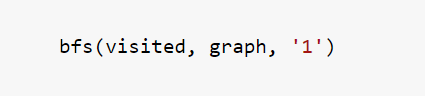
no backtracking required

**Graph:**



**Code:**





**Output:**



**Colab Link:** [**https://colab.research.google.com/drive/1kx2fQ-NvedFtb05ITteRVziiPhTm8OGM#scrollTo=3FY2sfcuGbl-**](https://colab.research.google.com/drive/1kx2fQ-NvedFtb05ITteRVziiPhTm8OGM#scrollTo=3FY2sfcuGbl-)

**Theory:**

### Depth First Search (DFS)

The DFS algorithm is a recursive algorithm that uses the idea of backtracking. It involves exhaustive searches of all the nodes by going ahead, if possible, else by backtracking. This recursive nature of DFS can be implemented using stacks.Pick a starting node and push all its adjacent nodes into a stack. Pop a node from stack to select the next node to visit and push all its adjacent nodes into a stack. Repeat this process until the stack is empty. However, ensure that the nodes that are visited are marked. This will prevent you from visiting the same node more than once. If you do not mark the nodes that are visited and you visit the same node more than once, you may end up in an infinite loop.

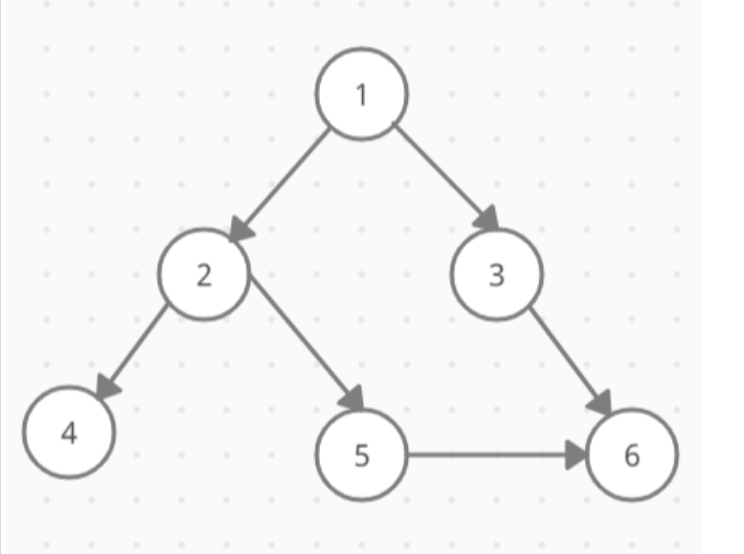
implemented using Stack(LIFO)

provide Deeptest Node

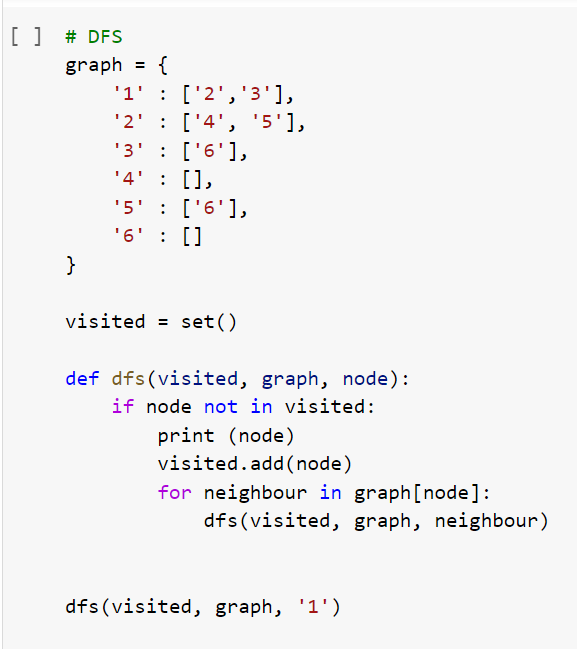
some time not give complete ans

backtracking required.

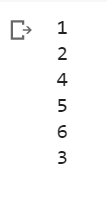
**Graph:**



**Code:**



**Output:**



**Practical 2**

**Colab Link:** [**https://colab.research.google.com/drive/1kx2fQ-NvedFtb05ITteRVziiPhTm8OGM#scrollTo=OArpTnZZvRJE&line=1&uniqifier=1**](https://colab.research.google.com/drive/1kx2fQ-NvedFtb05ITteRVziiPhTm8OGM#scrollTo=OArpTnZZvRJE&line=1&uniqifier=1)

**Aim:** Hill climbing algorithm implemented on tree of nodes

**Theory:**

Hill Climbing is a heuristic search used for mathematical optimisation problems in the field of Artificial Intelligence. It is one such Algorithm is one that will find the best possible solution to your problem in the most reasonable period of time.

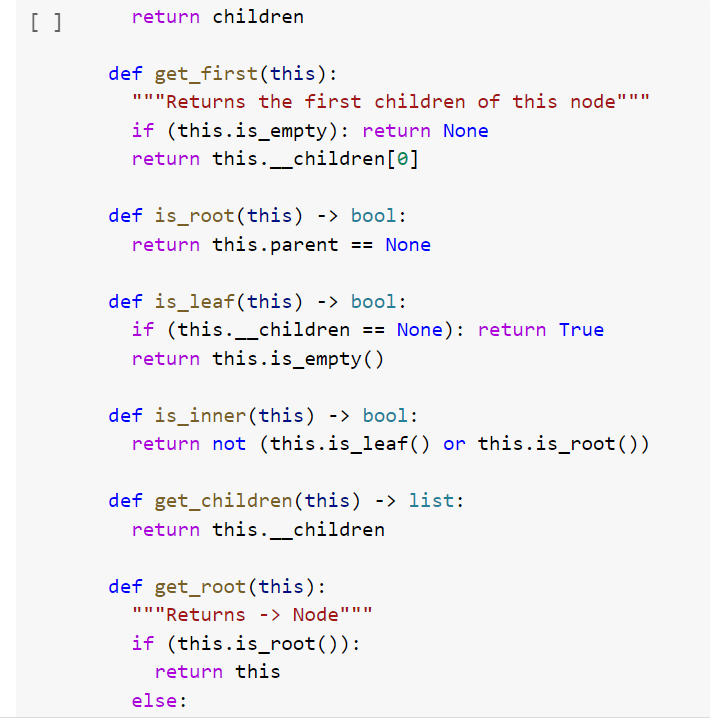
Given a large set of inputs and a good heuristic function, the algorithm tries to find the best possible solution to the problem in the most reasonable time period. This solution may not be the absolute best but it is sufficiently good considering the time allotted.

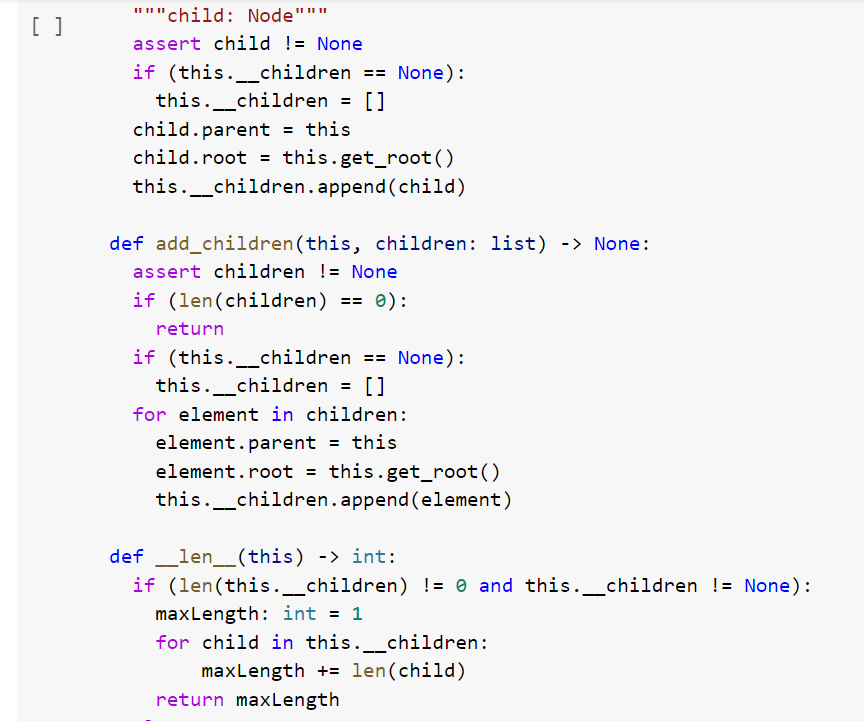
The definition above implies that hill-climbing solves the problems where we need to maximise or minimise a given real function by selecting values from the given inputs.

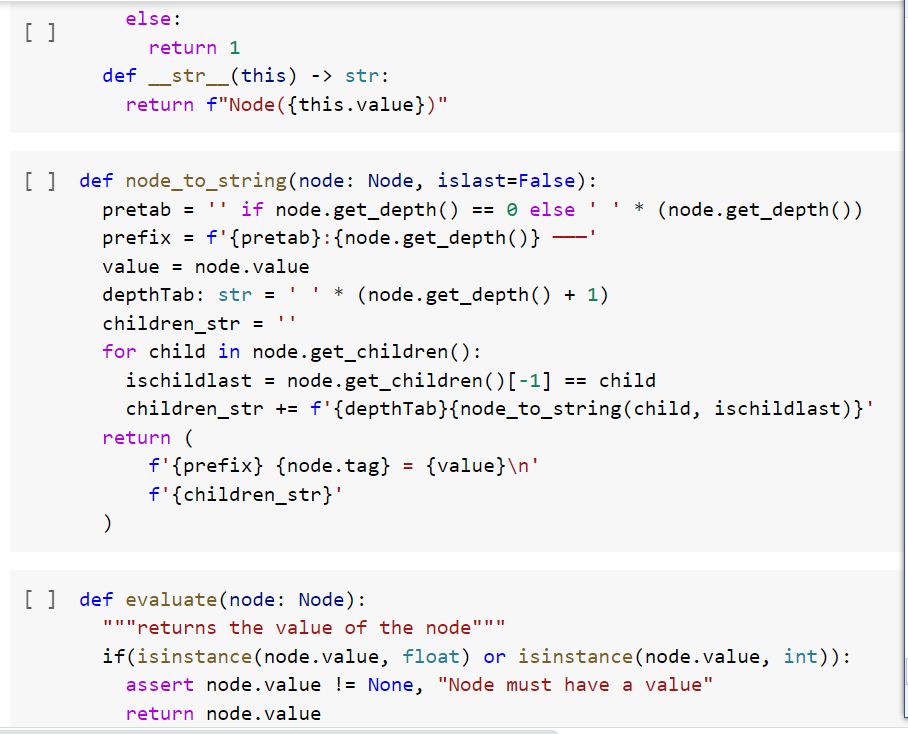
example of this is the Travelling Salesman Problem where we need to minimise the distance travelled by the salesman.

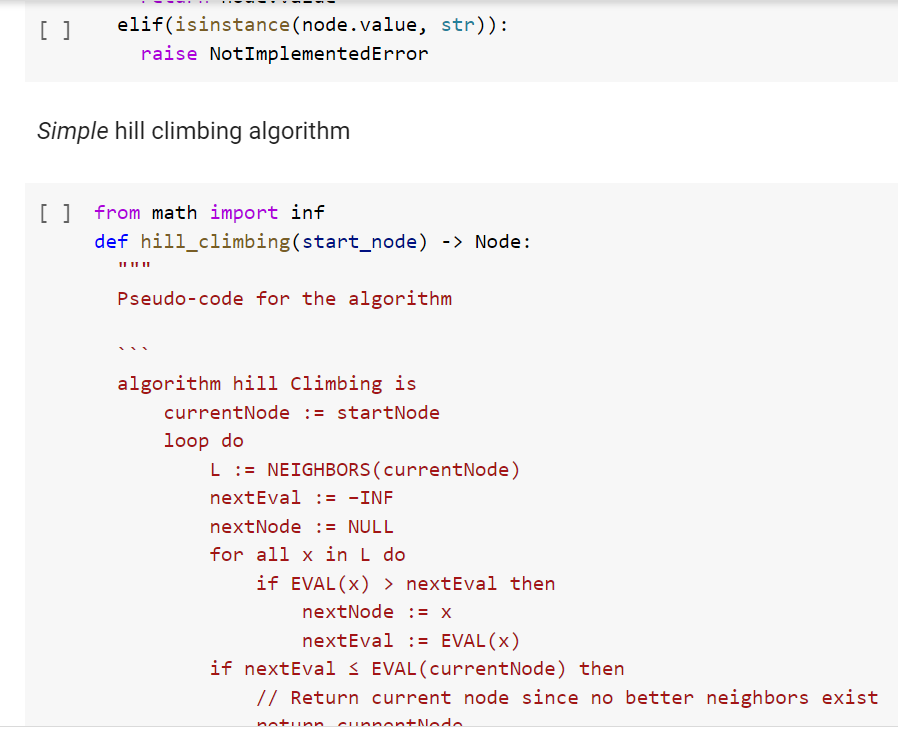
**Code:**

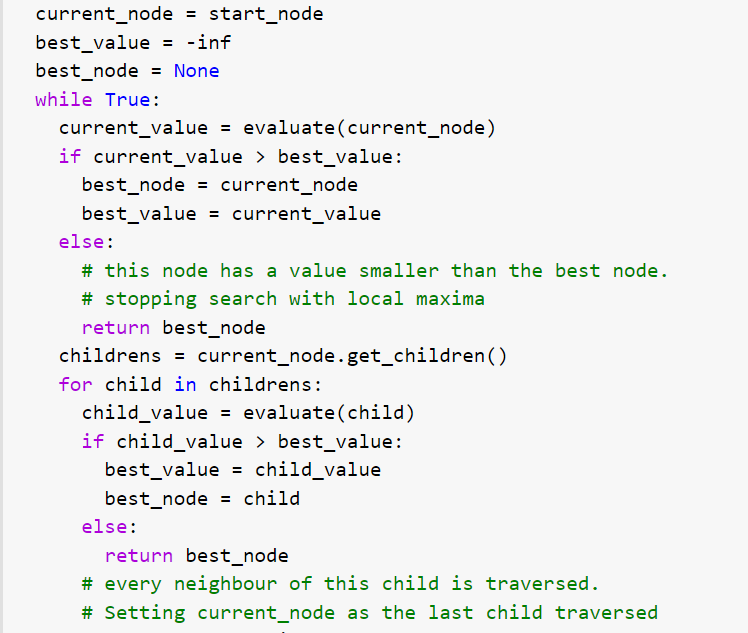


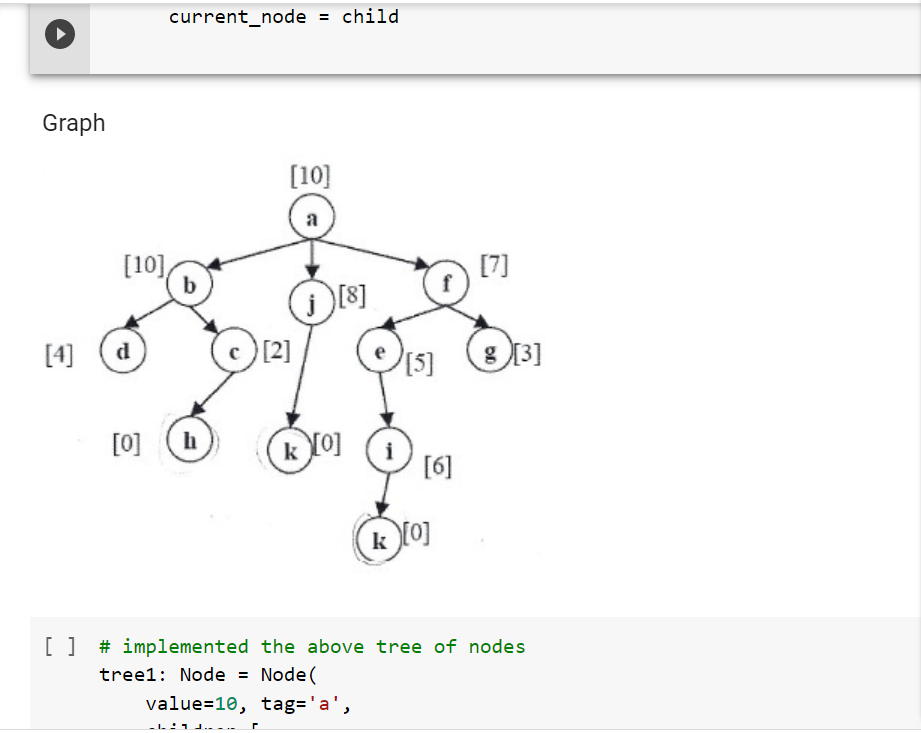






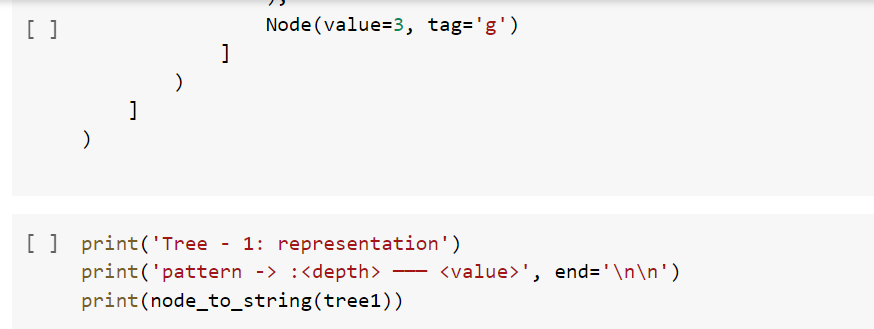




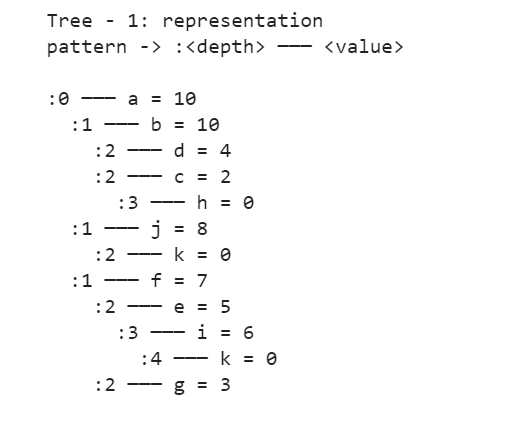


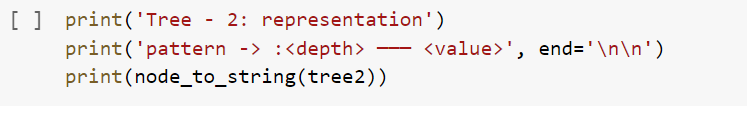




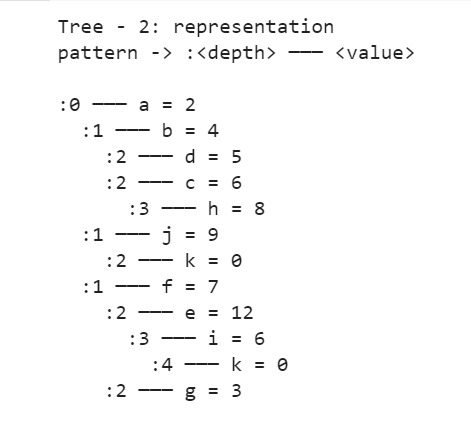


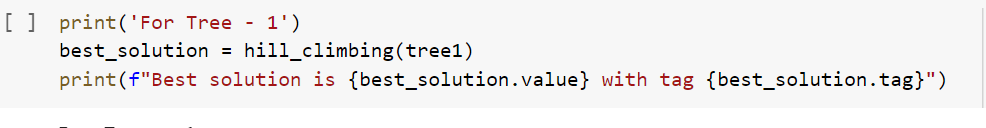
**Output:**



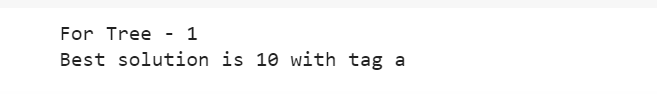


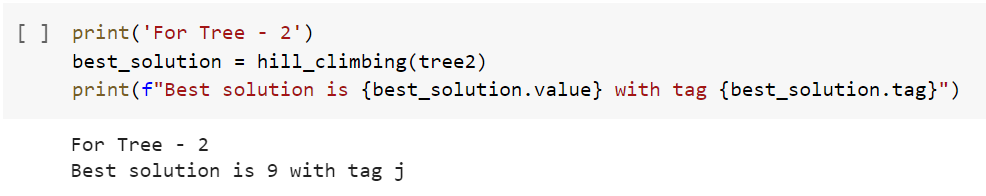
**Output:**





Output:





**Practical 3**

**Colab link:** [**https://colab.research.google.com/drive/1kx2fQ-NvedFtb05ITteRVziiPhTm8OGM#scrollTo=77sRqSE6cBMR**](https://colab.research.google.com/drive/1kx2fQ-NvedFtb05ITteRVziiPhTm8OGM#scrollTo=77sRqSE6cBMR)

### A) Aim: Backtracking with CSP

### Theory:

The constraint is the process to find the solution to the constraint. It is part of Artificial Intelligence. It defines that a set of objects whose state must satisfy a number of constraints. It is also called the Constraint Satisfaction Problem. The objectives are to assign a value for each variable such that all constraints are satisfied. Backtracking can be defined as a general algorithmic technique that considers searching every possible combination in order to solve a computational problem. Every constraint satisfaction problem which has clear and well-defined constraints on any objective solution, that incrementally builds candidate to the solution and abandons a candidate (“backtracks”) as soon as it determines that the candidate cannot possibly be completed to a valid solution, can be solved by Backtracking.

Constraint Satisfaction Problem

1 Set of variables {X1, X2, ..., Xn}

2 Set of domains for each variable {D1, D2, ..., Dn}

3 Set of constraints C

CSPs as Search Problems:

• initial state: empty assignment (no variables)

• actions: add a {variable = value} to assignment

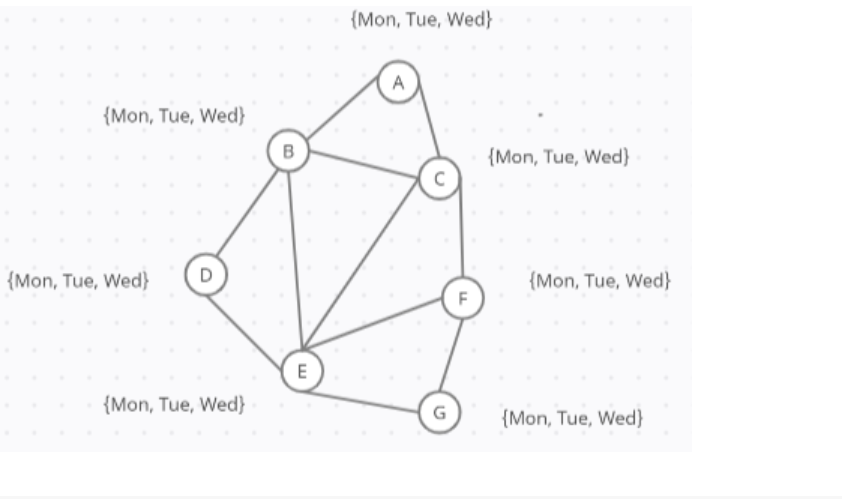
• transition model: shows how adding an assignment changes the assignment

• goal test: check if all variables assigned and constraints all satisfied

• path cost function: all paths have same cost

In this problem, we are implementing to constraint satisfaction rule and backtracking rule to solve this problem.

**Graph:**



**Code:**

"""

Naive backtracking search without any heuristics or inference.

"""

VARIABLES = ["A", "B", "C", "D", "E", "F", "G"]

CONSTRAINTS = [

    ("A", "B"),

    ("A", "C"),

    ("B", "C"),

    ("B", "D"),

    ("B", "E"),

    ("C", "E"),

    ("C", "F"),

    ("D", "E"),

    ("E", "F"),

    ("E", "G"),

    ("F", "G")

]

def backtrack(assignment):

    """Runs backtracking search to find an assignment."""

    # Check if assignment is complete

    if len(assignment) == len(VARIABLES):

        return assignment

    # Try a new variable

    var = select\_unassigned\_variable(assignment)

    for value in ["Monday", "Tuesday", "Wednesday"]:

        new\_assignment = assignment.copy()

        new\_assignment[var] = value

        if consistent(new\_assignment):

            result = backtrack(new\_assignment)

            if result is not None:

                return result

    return None

def select\_unassigned\_variable(assignment):

    """Chooses a variable not yet assigned, in order."""

    for variable in VARIABLES:

        if variable not in assignment:

            return variable

    return None

def consistent(assignment):

    """Checks to see if an assignment is consistent."""

    for (x, y) in CONSTRAINTS:

        # Only consider arcs where both are assigned

        if x not in assignment or y not in assignment:

            continue

        # If both have same value, then not consistent

        if assignment[x] == assignment[y]:

            return False

    # If nothing inconsistent, then assignment is consistent

    return True

solution = backtrack(dict())

print(solution)

**Output:**

{'A': 'Monday', 'B': 'Tuesday', 'C': 'Wednesday', 'D': 'Wednesday', 'E': 'Monday', 'F': 'Tuesday', 'G': 'Wednesday'}

### 3B Aim: MiniMax Alphabeta Pruning

### Colab Link: <https://colab.research.google.com/drive/1kx2fQ-NvedFtb05ITteRVziiPhTm8OGM#scrollTo=UE1ecnoc8Xt6&line=1&uniqifier=1>

### Theory:

Alpha beta pruning is an optimisation technique for the minimax algorithm. Alpha-beta pruning is nothing but the pruning of useless branches in decision trees. This alpha-beta pruning algorithm was discovered independently by researchers in the 1900s.The need for pruning came from the fact that in some cases decision trees become very complex. In that tree, some useless branches increase the complexity of the model. So, to avoid this, Alpha-Beta pruning comes to play so that the computer does not have to look at the entire tree.

Minimax is a classic depth-first search technique for a sequential two-player game. The two players are called MAX and MIN. The minimax algorithm is designed for finding the optimal move for MAX, the player at the root node. The search tree is created by recursively expanding all nodes from the root in a depth-first manner until either the end of the game or the maximum search depth is reached.

Alpha: Alpha is the best choice or the highest value that we have found at any instance along the path of Maximizer. The initial value for alpha is – ∞.

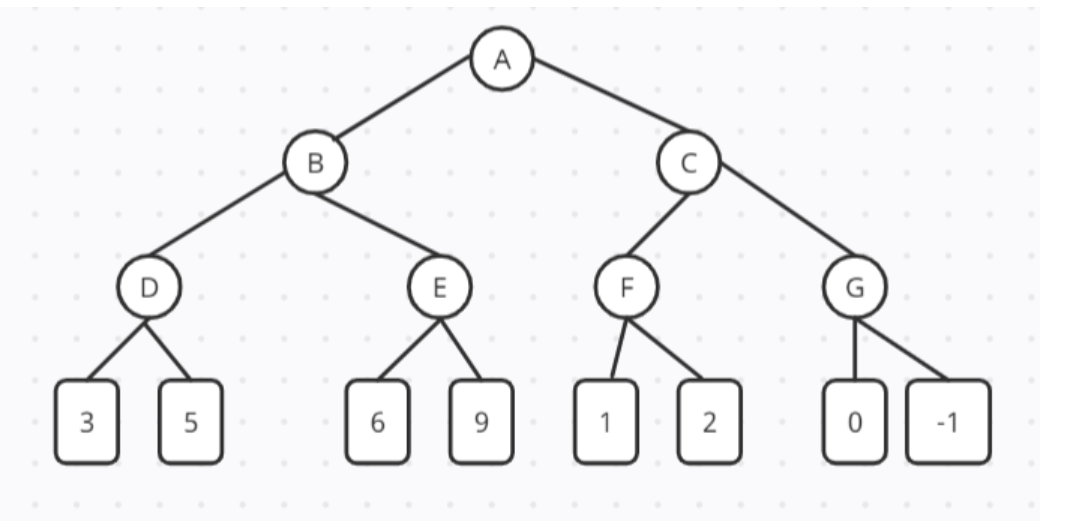
Beta: Beta is the best choice or the lowest value that we have found at any instance along the path of Minimizer. The initial value for alpha is + ∞.

The condition for Alpha-beta Pruning is that α >= β.

MAX will update only alpha values and MIN player will update only beta values

Alpha and Beta values only be passed to child nodes.

**Graph:**



**Code:**

# working of Alpha-Beta Pruning

# Initial values of Aplha and Beta

MAX, MIN = 1000, -1000

def minimax(depth, nodeIndex, maximizingPlayer,

            values, alpha, beta):

    if depth == 3:

        return values[nodeIndex]

    if maximizingPlayer:

        best = MIN

        # Recur for left and right children

        for i in range(0, 2):

            val = minimax(depth + 1, nodeIndex \* 2 + i,

                          False, values, alpha, beta)

            best = max(best, val)

            alpha = max(alpha, best)

            # Alpha Beta Pruning

            if beta <= alpha:

                break

        return best

    else:

        best = MAX

        for i in range(0, 2):

            val = minimax(depth + 1, nodeIndex \* 2 + i,

                            True, values, alpha, beta)

            best = min(best, val)

            beta = min(beta, best)

            # Alpha Beta Pruning

            if beta <= alpha:

                break

        return best

if \_\_name\_\_ == "\_\_main\_\_":

    values = [3, 5, 6, 9, 1, 2, 0, -1]

    print("The optimal value is :", minimax(0, 0, True, values, MIN, MAX))

**Output**:

The optimal value is : 5

**Practical 4**

**Colab** **link**:[**https://colab.research.google.com/drive/1kx2fQ-NvedFtb05ITteRVziiPhTm8OGM#scrollTo=Xbb2SZUcC3vH**](https://colab.research.google.com/drive/1kx2fQ-NvedFtb05ITteRVziiPhTm8OGM#scrollTo=Xbb2SZUcC3vH)

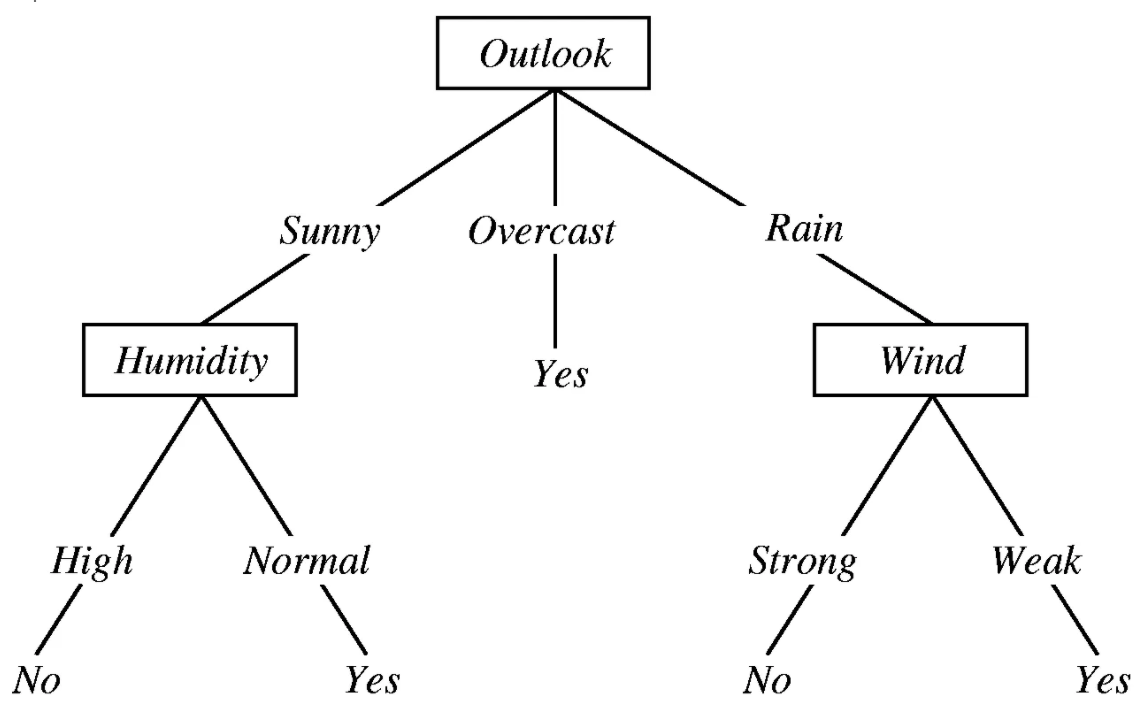
**Aim:**Decision Tree - ID3 Algorithm , Naive Bayes  Theorem

**Theory:**

## **4A Decision Tree**

Decision trees are assigned to the information based learning algorithms which use different measures of information gain for learning it is the most powerful and popular tool for classification and prediction. We can use decision trees for issues where we have continuous but also categorical input and target features. Decision tree is a flowchart like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class labe

**Graph:**



**Code:**

# code for decision tree

import pandas as pd

import numpy as np

from sklearn.datasets import load\_iris

from sklearn.tree import DecisionTreeClassifier

from sklearn.model\_selection import train\_test\_split

data = load\_iris()

print('Classes to predict: ', data.target\_names)

X = data.data

y = data.target

print('Number of examples in the data:', X.shape[0])

X[:4]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state = 47, test\_size = 0.25)

from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier(criterion = 'entropy')

clf.fit(X\_train, y\_train)

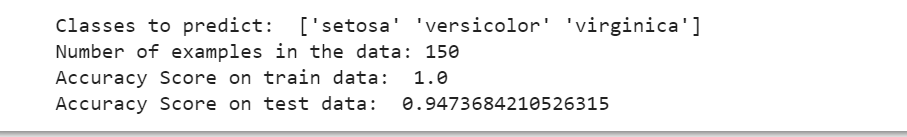
y\_pred =  clf.predict(X\_test)

from sklearn.metrics import accuracy\_score

print('Accuracy Score on train data: ', accuracy\_score(y\_true=y\_train, y\_pred=clf.predict(X\_train)))

print('Accuracy Score on test data: ', accuracy\_score(y\_true=y\_test, y\_pred=y\_pred))

**Output:**

:

## **4B) Aim: Naive Bayes Algorithm**

**Colab link:** [**https://colab.research.google.com/drive/1kx2fQ-NvedFtb05ITteRVziiPhTm8OGM#scrollTo=xP5JkxC0\_bxO&line=1&uniqifier=1**](https://colab.research.google.com/drive/1kx2fQ-NvedFtb05ITteRVziiPhTm8OGM#scrollTo=xP5JkxC0_bxO&line=1&uniqifier=1)

**Theory:**

Naive Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems. It is mainly used in text classification that includes a high-dimensional training dataset. Naive bayes does quite well when the training data doesn't contain all possibilities so it can be very good with low amounts of data. Decision trees work better with lots of data compared to Naive Bayes. Naive Bayes is used a lot in robotics and computer vision, and does quite well with those tasks.

**Code:**

from sklearn.datasets import load\_breast\_cancer

data\_loaded = load\_breast\_cancer()

X = data\_loaded.data

y = data\_loaded.target

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data\_loaded.data, data\_loaded.target, test\_size=0.2,random\_state=20)

from sklearn.naive\_bayes import GaussianNB

naive\_bayes = GaussianNB()

naive\_bayes.fit(X\_train , y\_train)

y\_predicted = naive\_bayes.predict(X\_test)

from sklearn import metrics

metrics.accuracy\_score(y\_predicted , y\_test)

**Output:**



### Practical 5

### Colab link: <https://colab.research.google.com/drive/1kx2fQ-NvedFtb05ITteRVziiPhTm8OGM#scrollTo=cpb_f_2SDYCt>

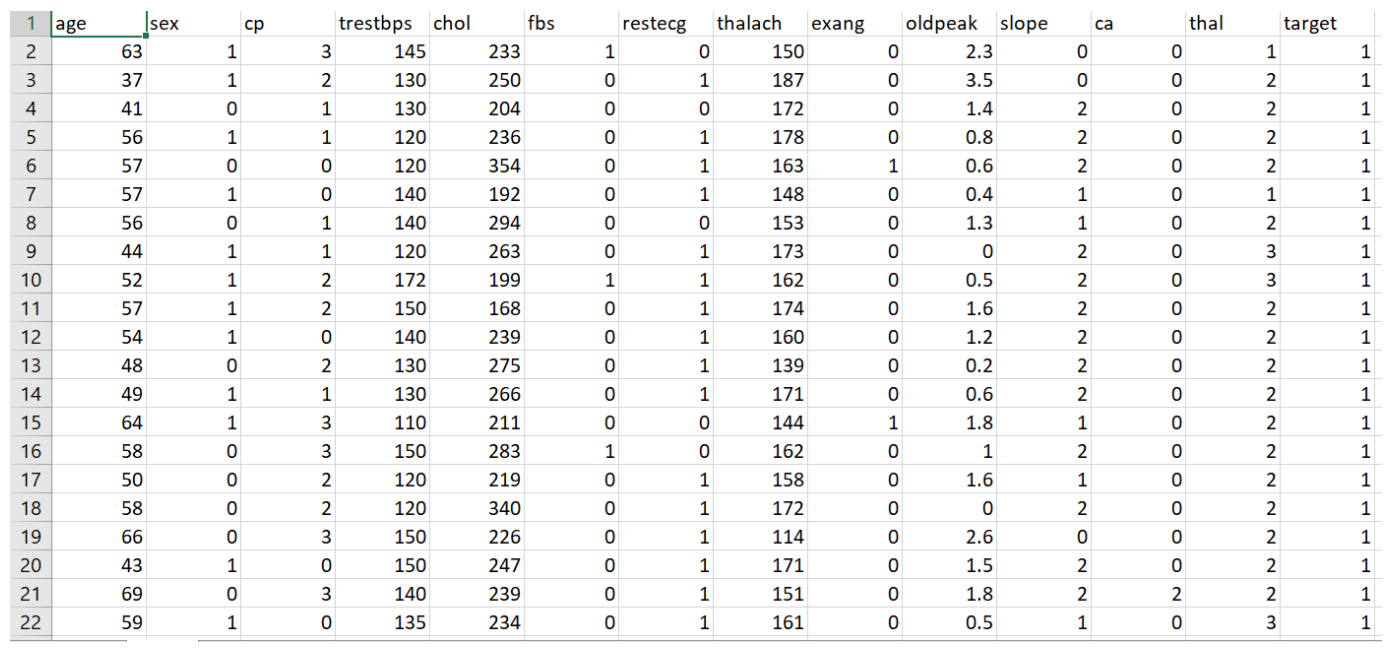
**Aim:** Bayesian Classifier heart.csv dataset

**Theory:**

Bayesian network theory can be thought of as a fusion of incidence diagrams and Bayes’ theorem. Bayesian belief network is key computer technology for dealing with probabilistic events and to solve a problem which has uncertainty.A Bayesian network is a probabilistic graphical model which represents a set of variables and their conditional dependencies using a directed acyclic graph.

It is also called a Bayes network, belief network, decision network, or Bayesian model. The probability of an event occurring given that another event has already occurred is called a conditional probability.

**Dataset:**



**Code:**

from google.colab import files

uploaded = files.upload()

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

import sklearn as skl

heart = pd.read\_csv('heart.csv')

import io

!pip install pgmpy

from pgmpy.models import BayesianModel

from pgmpy.estimators import MaximumLikelihoodEstimator, BayesianEstimator

heart.columns

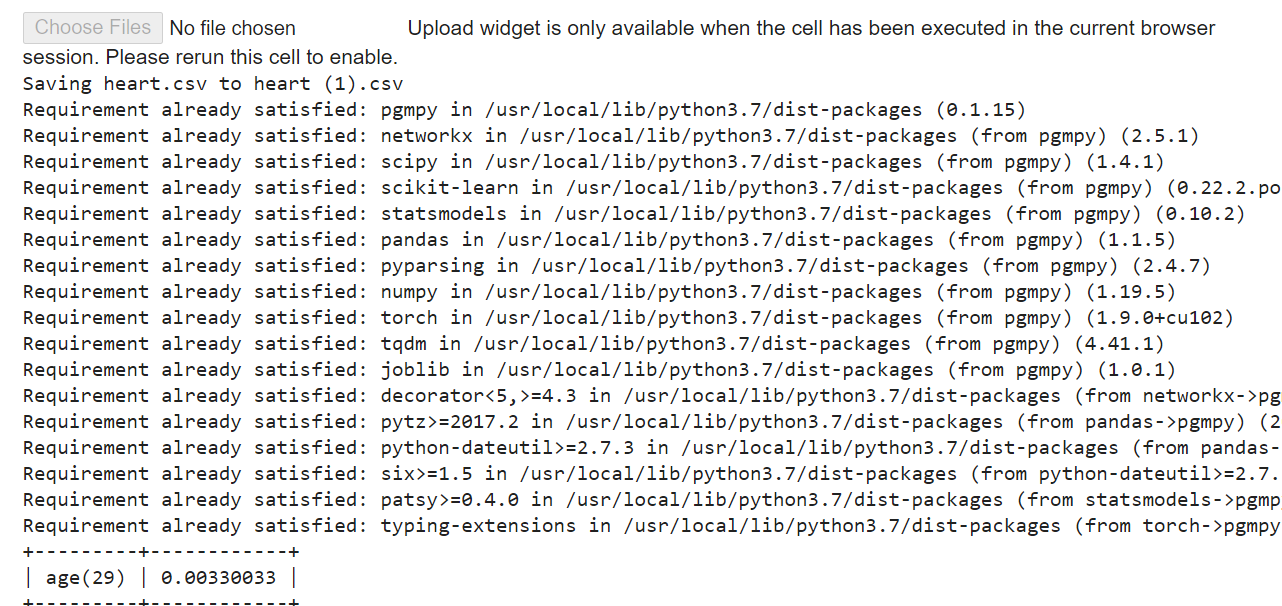
model = BayesianModel([('age','trestbps'),('age','fbs'),('sex','trestbps'),('exang','trestbps'),('trestbps','target'),('fbs','target'),

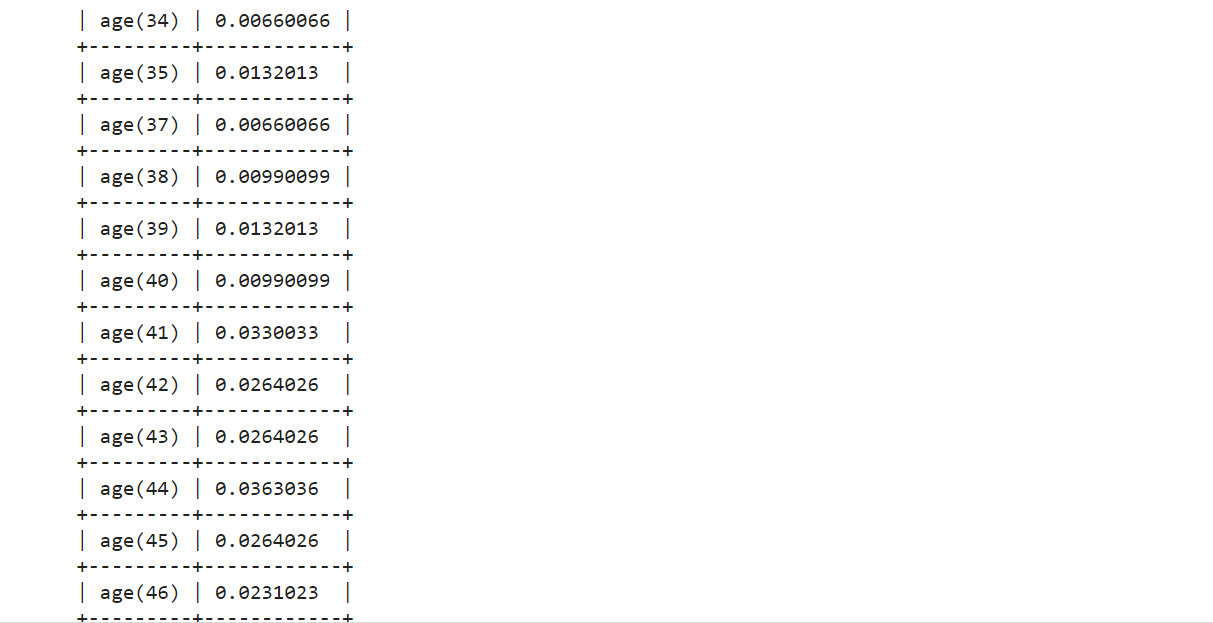
('target','restecg'),('target','thalach'),('target','chol')])

model.fit(heart,estimator=MaximumLikelihoodEstimator)

print(model.get\_cpds('age'))

**Output:**





**Practical 6**

**Colab link:** [**https://colab.research.google.com/drive/1kx2fQ-NvedFtb05ITteRVziiPhTm8OGM#scrollTo=PkOnDo-s444h**](https://colab.research.google.com/drive/1kx2fQ-NvedFtb05ITteRVziiPhTm8OGM#scrollTo=PkOnDo-s444h)

**Markov Model Speech Analysis**

**Aim:** Apply Markov Property to generate Donald’s Trump’s speech by considering each word used in the speech and for each word, create a dictionary of words that are used next.

**Theory:**

A Markov model is a stochastic method for randomly changing systems where it is assumed that future states do not depend on past states. These models show all possible states as well as the transitions, rate of transitions and probabilities between them.

For instance, Hidden Markov Models are similar to Markov chains, but they have a few hidden states. Since they’re hidden, you can’t be see them directly in the chain, only through the observation of another process that depends on it.

A Markov chain is a model that tells us something about the probabilities of sequences of random variables, states, each of which can take on values from some set. These sets can be words, or tags, or symbols representing anything, like the weather. A Markov chain makes a very strong assumption that if we want to predict the future in the sequence, all that matters is the current state.

**Code:**

import numpy as np

from google.colab import files

files.upload()

trump = open('DonaldTs.txt', encoding='utf8').read()

#display the data

print(trump)

corpus = trump.split()

#Display the corpus

print(corpus)

def make\_pairs(corpus):

  for i in range(len(corpus) - 1):

    yield (corpus[i], corpus[i + 1])

pairs = make\_pairs(corpus)

word\_dict={}

for word\_1, word\_2 in pairs:

  if word\_1 in word\_dict.keys():

    word\_dict[word\_1].append(word\_2)

  else:

    word\_dict[word\_1] = [word\_2]

#randomly pick the first word

first\_word = np.random.choice(corpus)

#Pick the first word as a capitalized word so that the picked word is not taken from in between a sentence

while first\_word.islower():

#Start the chain from the picked word

   chain = [first\_word]

#Initialize the number of stimulated words

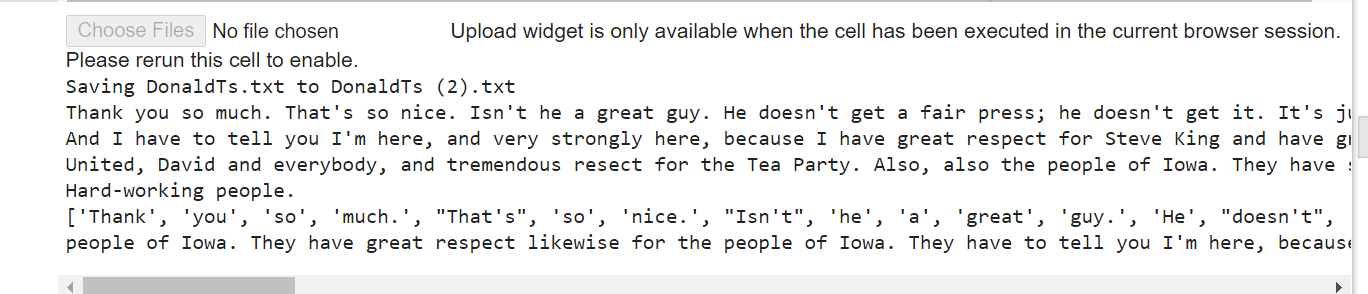
   n\_words = 60

   for i in range(n\_words):

     chain.append(np.random.choice(word\_dict[chain[-1]]))

print(" ".join(chain))

**Output:**



**Practical 7**

**Colab link:** [**https://colab.research.google.com/drive/1kx2fQ-NvedFtb05ITteRVziiPhTm8OGM#scrollTo=yUMLRIoWD0hb**](https://colab.research.google.com/drive/1kx2fQ-NvedFtb05ITteRVziiPhTm8OGM#scrollTo=yUMLRIoWD0hb)

**Prediction Algorithm - Cat and Non-Cat images**

**Aim:** Prediction Algorithm - Use of different packages on dataset of Cat and Non-Cat images

**Theory:**

Predictive analytics in the Trendskout AI Platform Prediction through machine learning or deep learning can be done in a number of different ways, depending on the underlying algorithm that is used. As the name suggests, predictive models are designed to predict unknown values, properties or events. If the predictions from the algorithm proved to be accurate, the algorithm could theoretically be used in the future to identify people at high risk of suicide, and deliver targeted programs to them. That would be a very good thing. Predictive algorithms are everywhere.

**Code:**

import numpy as np

import matplotlib.pyplot as plt

import h5py

import scipy

from PIL import Image

from scipy import ndimage

%matplotlib inline

def load\_dataset():

    train\_dataset = h5py.File('train\_catvnoncat.h5', "r")

    train\_set\_x\_orig = np.array(train\_dataset["train\_set\_x"][:]) # your train set features

    train\_set\_y\_orig = np.array(train\_dataset["train\_set\_y"][:]) # your train set labels

    test\_dataset = h5py.File('test\_catvnoncat.h5', "r")

    test\_set\_x\_orig = np.array(test\_dataset["test\_set\_x"][:]) # your test set features

    test\_set\_y\_orig = np.array(test\_dataset["test\_set\_y"][:]) # your test set labels

    classes = np.array(test\_dataset["list\_classes"][:]) # the list of classes

    train\_set\_y\_orig = train\_set\_y\_orig.reshape((1, train\_set\_y\_orig.shape[0]))

    test\_set\_y\_orig = test\_set\_y\_orig.reshape((1, test\_set\_y\_orig.shape[0]))

    return train\_set\_x\_orig, train\_set\_y\_orig, test\_set\_x\_orig, test\_set\_y\_orig, classes

train\_set\_x\_orig, train\_set\_y, test\_set\_x\_orig, test\_set\_y\_orig, classes = load\_dataset()

#index = 26

#plt.imshow(train\_set\_x\_orig[index])

#print ("y = " + str(train\_set\_y[:, index]) + ", it's a '" + classes[np.squeeze(train\_set\_y[:, index])].decode("utf-8") +  "' picture.")

index= 25

plt.imshow(test\_set\_x\_orig[index])

print ("y = " + str(test\_set\_y[:, index]) + ", it's a '" + classes[np.squeeze(test\_set\_y[:, index])].decode("utf-8") +  "' picture.")

#plt.imshow(train\_set\_y\_orig[index])

#print ("y = " + str(train\_set\_y[:, index]) + ", it's a '" + classes[np.squeeze(train\_set\_y[:, index])].decode("utf-8") +  "' picture.")

import numpy as np

import matplotlib.pyplot as plt

import h5py

import scipy

from PIL import Image

from scipy import ndimage

#from lr\_utils import load\_dataset

%matplotlib inline

def load\_dataset():

    train\_dataset = h5py.File('train\_catvnoncat.h5', "r")

    train\_set\_x\_orig = np.array(train\_dataset["train\_set\_x"][:]) # your train set features

    train\_set\_y\_orig = np.array(train\_dataset["train\_set\_y"][:]) # your train set labels

    test\_dataset = h5py.File('test\_catvnoncat.h5', "r")

    test\_set\_x\_orig = np.array(test\_dataset["test\_set\_x"][:]) # your test set features

    test\_set\_y\_orig = np.array(test\_dataset["test\_set\_y"][:]) # your test set labels

    classes = np.array(test\_dataset["list\_classes"][:]) # the list of classes

    train\_set\_y\_orig = train\_set\_y\_orig.reshape((1, train\_set\_y\_orig.shape[0]))

    test\_set\_y\_orig = test\_set\_y\_orig.reshape((1, test\_set\_y\_orig.shape[0]))

    return train\_set\_x\_orig, train\_set\_y\_orig, test\_set\_x\_orig, test\_set\_y\_orig, classes

#Data will be loaded from the test\_catvnoncat.h5 and train\_catvnoncat.h5 files

#The load\_dataset function below is responsebile for loading the above mentioned data files.

#lr\_utils file includes the function load\_dataset()

# Loading the data (cat/non-cat)

train\_set\_x\_orig, train\_set\_y, test\_set\_x\_orig, test\_set\_y\_orig, classes = load\_dataset()

# We added "\_orig" at the end of image datasets (train and test) because we are going to preprocess them. After preprocessing, we will end up with train\_set\_x and test\_set\_x (the labels train\_set\_y and test\_set\_y don't need any preprocessing).

# Each line of your train\_set\_x\_orig and test\_set\_x\_orig is an array representing an image. You can visualize an example by running the following code. Feel free also to change the `index` value and re-run to see other images.

# Example of a picture

#change the index value below to check if the image at that particular index is cat or non cat

index = 26

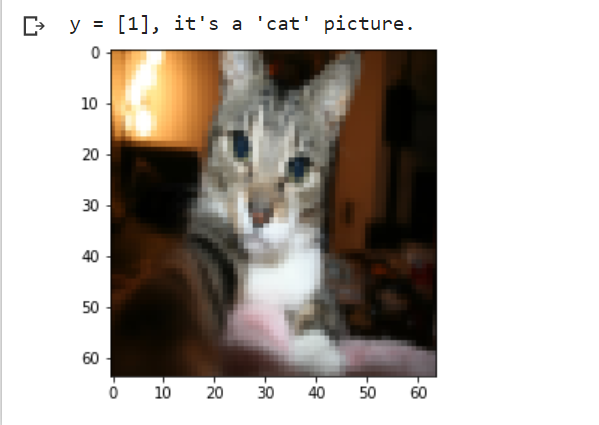
plt.imshow(train\_set\_x\_orig[index])

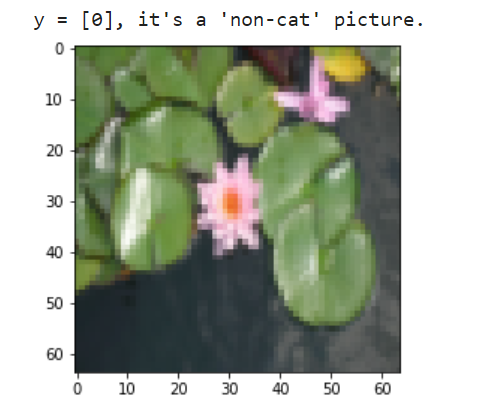
print ("y = " + str(train\_set\_y[:, index]) + ", it's a '" + classes[np.squeeze(train\_set\_y[:, index])].decode("utf-8") +  "' picture.")

#plt.imshow(train\_set\_y\_orig[index])

#print ("y = " + str(train\_set\_y[:, index]) + ", it's a '" + classes[np.squeeze(train\_set\_y[:, index])].decode("utf-8") +  "' picture.")

**Output**:





import numpy as np

import h5py

    # Loading the data (cat/non-cat)

train\_dataset = h5py.File('train\_catvnoncat.h5', "r")

train\_set\_x\_orig = np.array(train\_dataset["train\_set\_x"][:]) # train set features

train\_set\_y\_orig = np.array(train\_dataset["train\_set\_y"][:]) # train set labels

test\_dataset = h5py.File('test\_catvnoncat.h5', "r")

test\_set\_x\_orig = np.array(test\_dataset["test\_set\_x"][:]) # test set features

test\_set\_y\_orig = np.array(test\_dataset["test\_set\_y"][:]) # test set labels

classes = np.array(test\_dataset["list\_classes"][:]) # the list of classes

train\_set\_y = train\_set\_y\_orig.reshape((1, train\_set\_y\_orig.shape[0]))

test\_set\_y = test\_set\_y\_orig.reshape((1, test\_set\_y\_orig.shape[0]))

m\_train = train\_set\_x\_orig.shape[0]

m\_test = test\_set\_x\_orig.shape[0]

num\_px = train\_set\_x\_orig.shape[1]

print ("Dataset dimensions:")

print ("Number of training examples: m\_train = " + str(m\_train))

print ("Number of testing examples: m\_test = " + str(m\_test))

print ("Height/Width of each image: num\_px = " + str(num\_px))

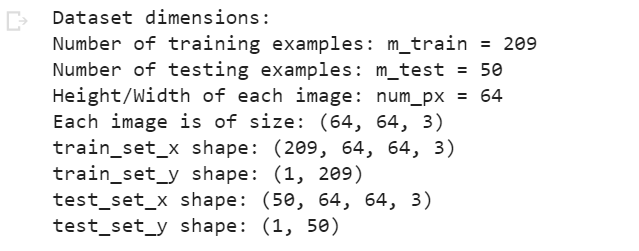
print ("Each image is of size: (" + str(num\_px) + ", " + str(num\_px) + ", 3)")

print ("train\_set\_x shape: " + str(train\_set\_x\_orig.shape))

print ("train\_set\_y shape: " + str(train\_set\_y.shape))

print ("test\_set\_x shape: " + str(test\_set\_x\_orig.shape))

print ("test\_set\_y shape: " + str(test\_set\_y.shape))



# Reshape the training and test examples

train\_set\_x\_flatten = train\_set\_x\_orig.reshape(train\_set\_x\_orig.shape[0], -1).T

test\_set\_x\_flatten = test\_set\_x\_orig.reshape(test\_set\_x\_orig.shape[0], -1).T

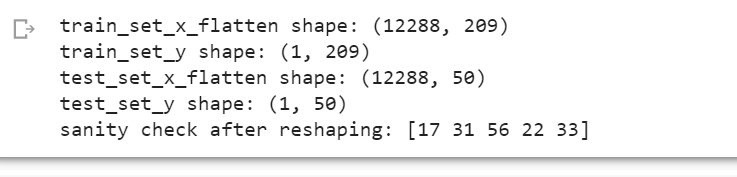
print ("train\_set\_x\_flatten shape: " + str(train\_set\_x\_flatten.shape))

print ("train\_set\_y shape: " + str(train\_set\_y.shape))

print ("test\_set\_x\_flatten shape: " + str(test\_set\_x\_flatten.shape))

print ("test\_set\_y shape: " + str(test\_set\_y.shape))

print ("sanity check after reshaping: " + str(train\_set\_x\_flatten[0:5,0]))



train\_set\_x = train\_set\_x\_flatten/255.

test\_set\_x = test\_set\_x\_flatten/255.

from sklearn.linear\_model import LogisticRegression

lr = LogisticRegression(C=1000.0, random\_state=0)

lr.fit(train\_set\_x.T, train\_set\_y.T.ravel())

lr.coef\_.shape

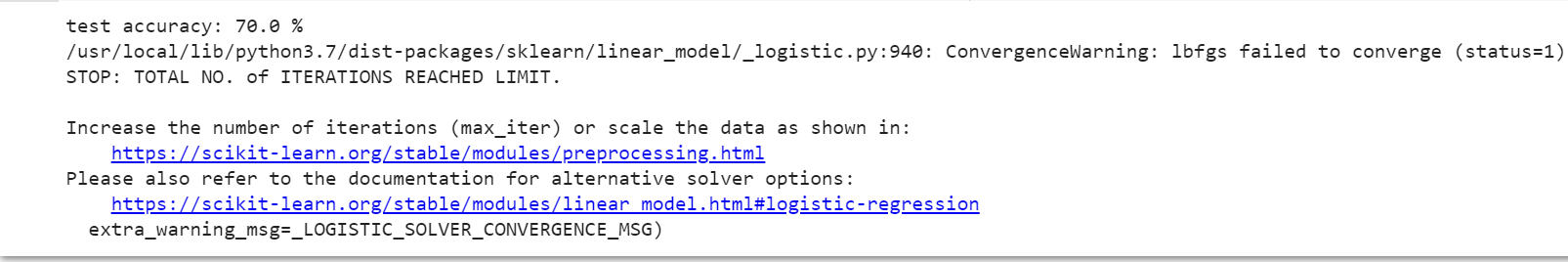
lr.coef\_

lr.intercept\_

Y\_prediction = lr.predict(test\_set\_x.T)

Y\_prediction.shape

print("test accuracy: {} %".format(100 - np.mean(np.abs(Y\_prediction - test\_set\_y)) \* 100))





**Practical 8**

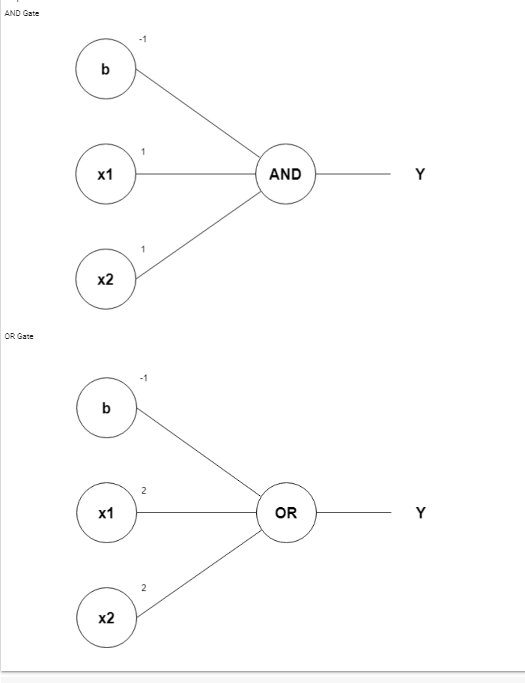
**Colab link:** [**https://colab.research.google.com/drive/1kx2fQ-NvedFtb05ITteRVziiPhTm8OGM#scrollTo=3TiXAChqOwMy&line=1&uniqifier=1**](https://colab.research.google.com/drive/1kx2fQ-NvedFtb05ITteRVziiPhTm8OGM#scrollTo=3TiXAChqOwMy&line=1&uniqifier=1)

**Aim:** AND AND OR PERCEPTRON

**Theory:**

A Perceptron is an Artificial Neuron. It is the simplest possible Neural Network. Neural Networks are the building blocks of Artificial Intelligence. Perceptron is one of the earliest—and most elementary—artificial neural network models. The perceptron is extremely simple by modern deep learning model standards. However the concepts utilised in its design apply more broadly to sophisticated deep network architectures. The perceptron is a supervised learning binary classification algorithm, originally developed by Frank Rosenblatt in 1957. It categorises input data into one of two separate states based a training procedure carried out on prior input data.

**Graph:** AND AND OR Gate



**Code:**

import numpy as np

def unitStep(v):

  if v>=0:

    return 1

  else:

    return 0

def perceptronModel(x,w,b):

  v=np.dot(w,x)+b

  y=unitStep(v)

  return y

def OR\_logicfunction(x):

  w=np.array([1,1])

  b=-0.5

  return perceptronModel(x,w,b)

def AND\_logicFunction(x):

 w = np.array([1, 1])

 bAND = -1.5

 return perceptronModel(x, w, bAND)

 test1=np.array([0,0])

test2=np.array([0,1])

test3=np.array([1,0])

test4=np.array([1,1])

print("OR({},{})={}".format(0,0,OR\_logicfunction(test1)))

print("OR({},{})={}".format(0,1,OR\_logicfunction(test2)))

print("OR({},{})={}".format(1,0,OR\_logicfunction(test3)))

print("OR({},{})={}".format(1,1,OR\_logicfunction(test4)))

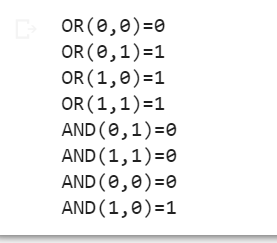
print("AND({},{})={}".format(0,1,AND\_logicFunction(test1)))

print("AND({},{})={}".format(1,1,AND\_logicFunction(test2)))

print("AND({},{})={}".format(0,0,AND\_logicFunction(test3)))

print("AND({},{})={}".format(1,0,AND\_logicFunction(test4)))

**Output:**



**Practical 9**

**Colab link:** [**https://colab.research.google.com/drive/1kx2fQ-NvedFtb05ITteRVziiPhTm8OGM#scrollTo=0xLyGk88Djli**](https://colab.research.google.com/drive/1kx2fQ-NvedFtb05ITteRVziiPhTm8OGM#scrollTo=0xLyGk88Djli)

**Classify the iris data**

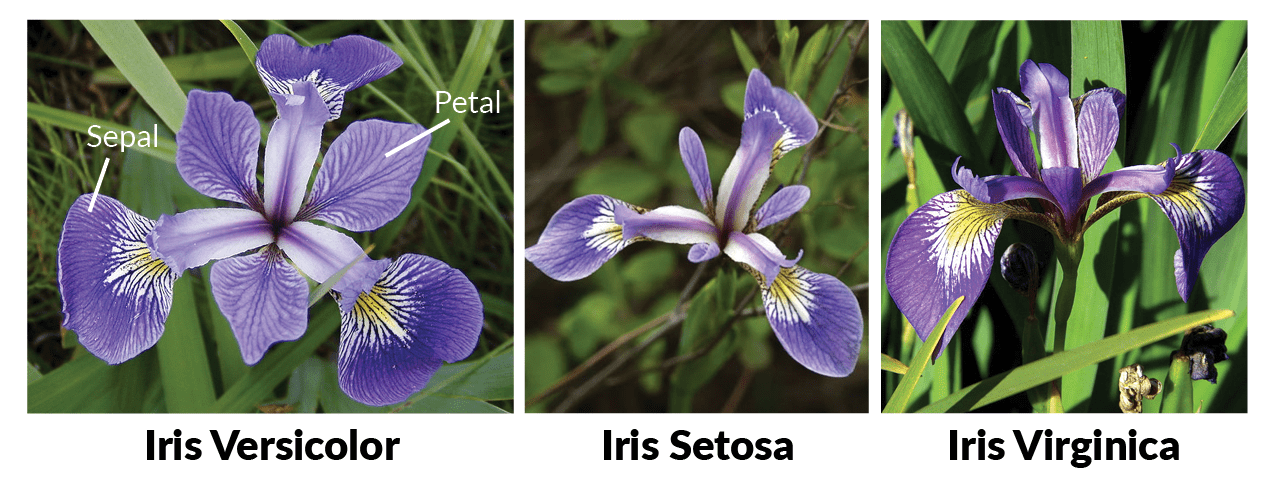
**Aim:** Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.

**Theory:**

The k-nearest neighbors (KNN) algorithm is a simple, supervised machine learning algorithm that can be used to solve both classification and regression problems. It's easy to implement and understand, but has a major drawback of becoming significantly slows as the size of that data in use grows.

It is also flexible because the number of K neighbors and the distance between them are chosen for what is appropriate for the data being analyzed. It is widely disposable in real-life scenarios since it is non-parametric, meaning, it does not make any underlying assumptions about the distribution of data We are given some prior data (also called training data), which classifies coordinates into groups identified by an attribute.

The Iris Dataset contains four features (length and width of sepals and petals) of 50 samples of three species of Iris (Iris setosa, Iris virginica and Iris versicolor). These measures were used to create a linear discriminant model to classify the species. The dataset is often used in data mining, classification and clustering examples and to test algorithms.



**Code:**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

import seaborn as sns

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

# Assign colum names to the dataset

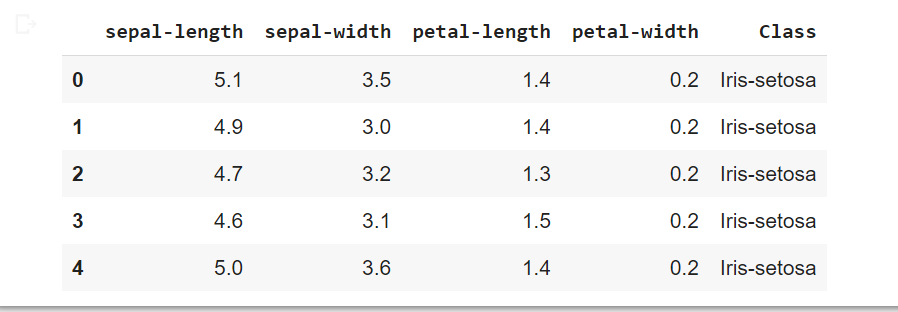
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']

# Read dataset to pandas dataframe

dataset = pd.read\_csv(url, names=names)

dataset.head()

**Output:**



X = dataset.iloc[:, :-1].values

y = dataset.iloc[:, 4].values

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.20)

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

scaler.fit(X\_train)

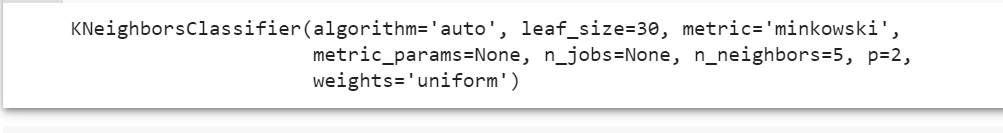
X\_train = scaler.transform(X\_train)

X\_test = scaler.transform(X\_test)

from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n\_neighbors=5)

classifier.fit(X\_train, y\_train)



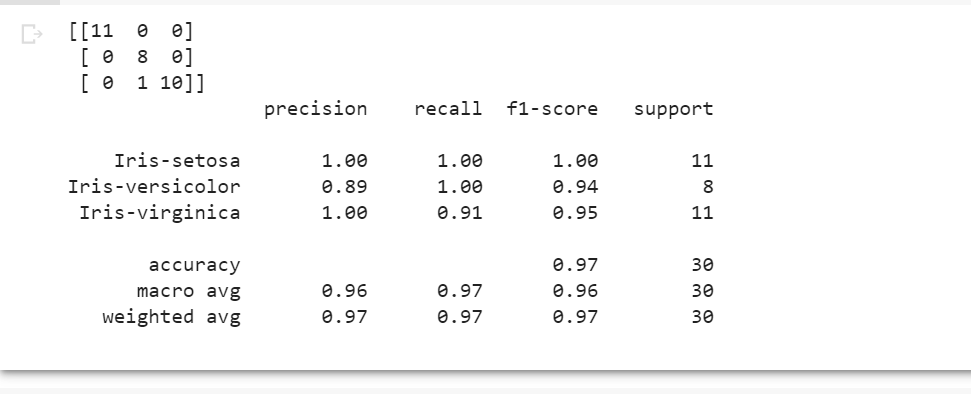
y\_pred = classifier.predict(X\_test)

from sklearn.metrics import classification\_report, confusion\_matrix

print(confusion\_matrix(y\_test, y\_pred))

print(classification\_report(y\_test, y\_pred))

**Output:**



### 

### 

### Practical 10

### Colab link: <https://colab.research.google.com/drive/1kx2fQ-NvedFtb05ITteRVziiPhTm8OGM#scrollTo=J4ePRdp2PFJ6>

### Aim: Implementation of basic neural network model with 4 activation functions on Pima Indians onset of diabetes dataset.

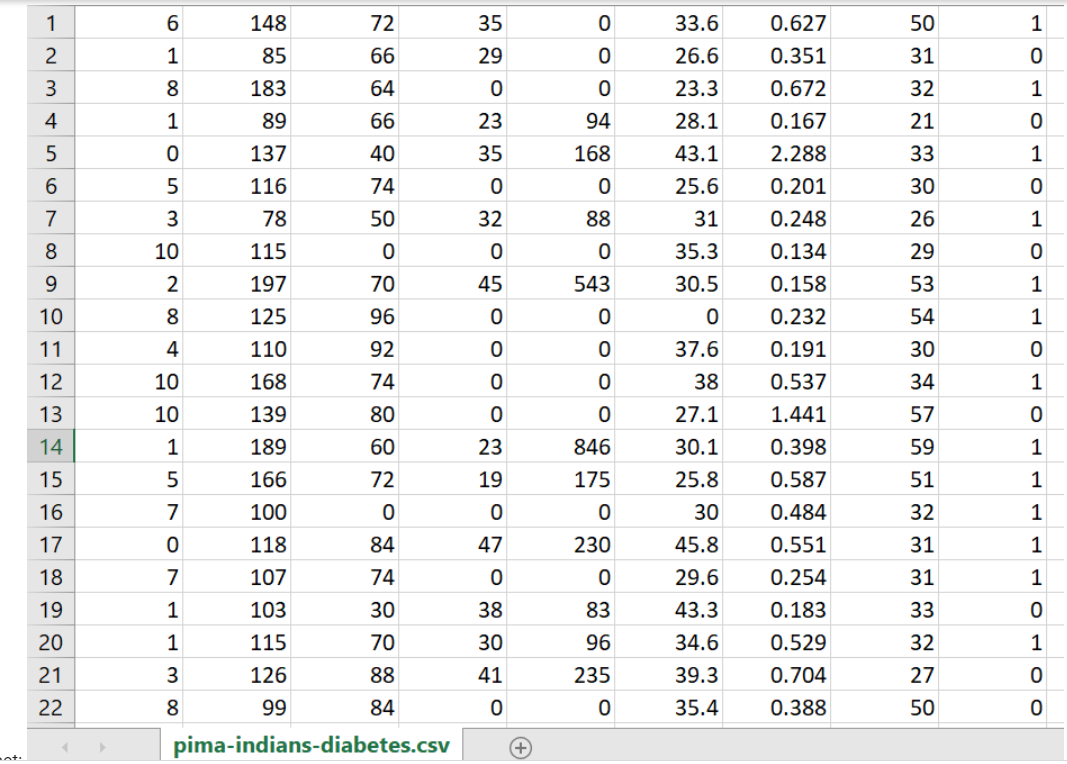
**Theory:**

The Role of Artificial Intelligence made it possible for machines to learn from experience to perform tasks more efficiently. The Artificial neural network is one of its advancements which is inspired by the structure of the human brain that helps computers and machines more like a human.

This is a standard machine learning dataset from the UCI Machine Learning repository. It describes patient medical record data for Pima Indians and whether they had an onset of diabetes within five years.

As such, it is a binary classification problem (onset of diabetes as 1 or not as 0). All of the input variables that describe each patient are numerical. This makes it easy to use directly with neural networks that expect numerical input and output values, and ideal for our first neural network in Keras.

**Dataset:**



**Code:**

from google.colab import files

files.upload()

from numpy import loadtxt

from keras.models import Sequential

from keras.layers import Dense

# load the dataset

dataset = loadtxt('pima-indians-diabetes.csv.csv', delimiter=',')

# split into input (X) and output (y) variables

X = dataset[:,0:8]

y = dataset[:,8]

# define the keras model

model = Sequential()

model.add(Dense(12, input\_dim=8, activation='relu'))

model.add(Dense(8, activation='relu'))

model.add(Dense(1, activation='sigmoid'))

# compile the keras model

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

...

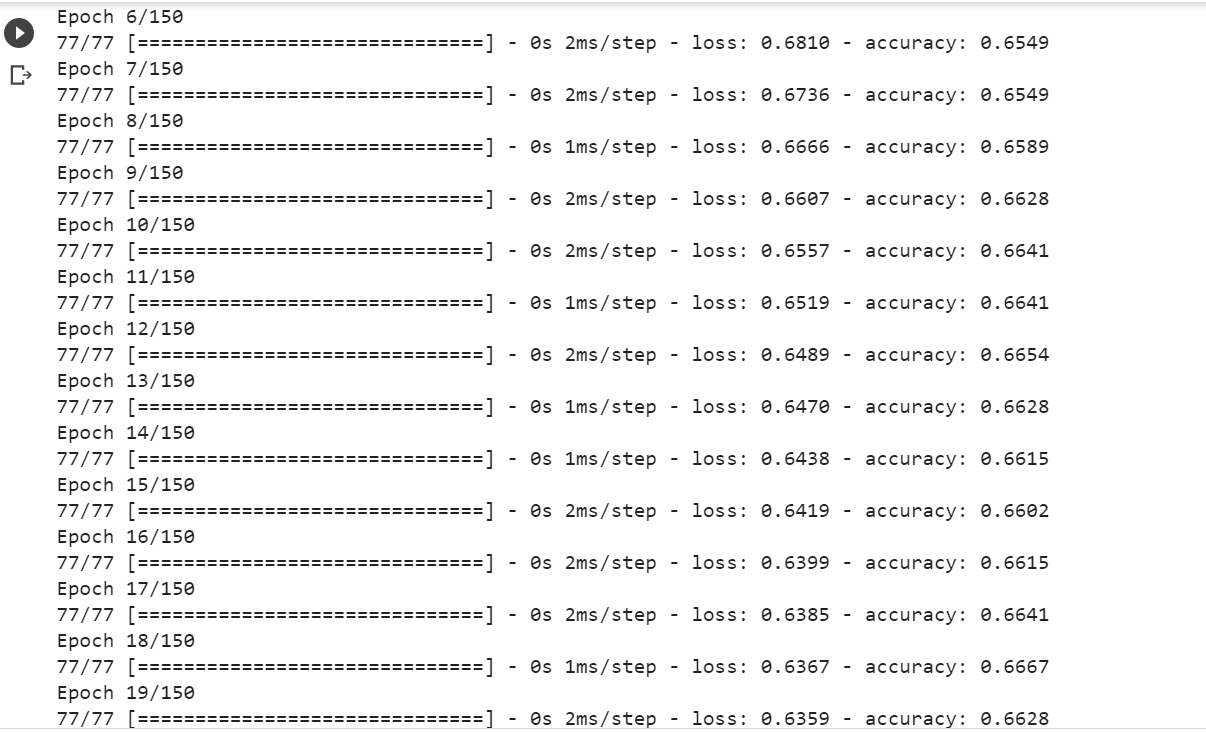
# fit the keras model on the dataset

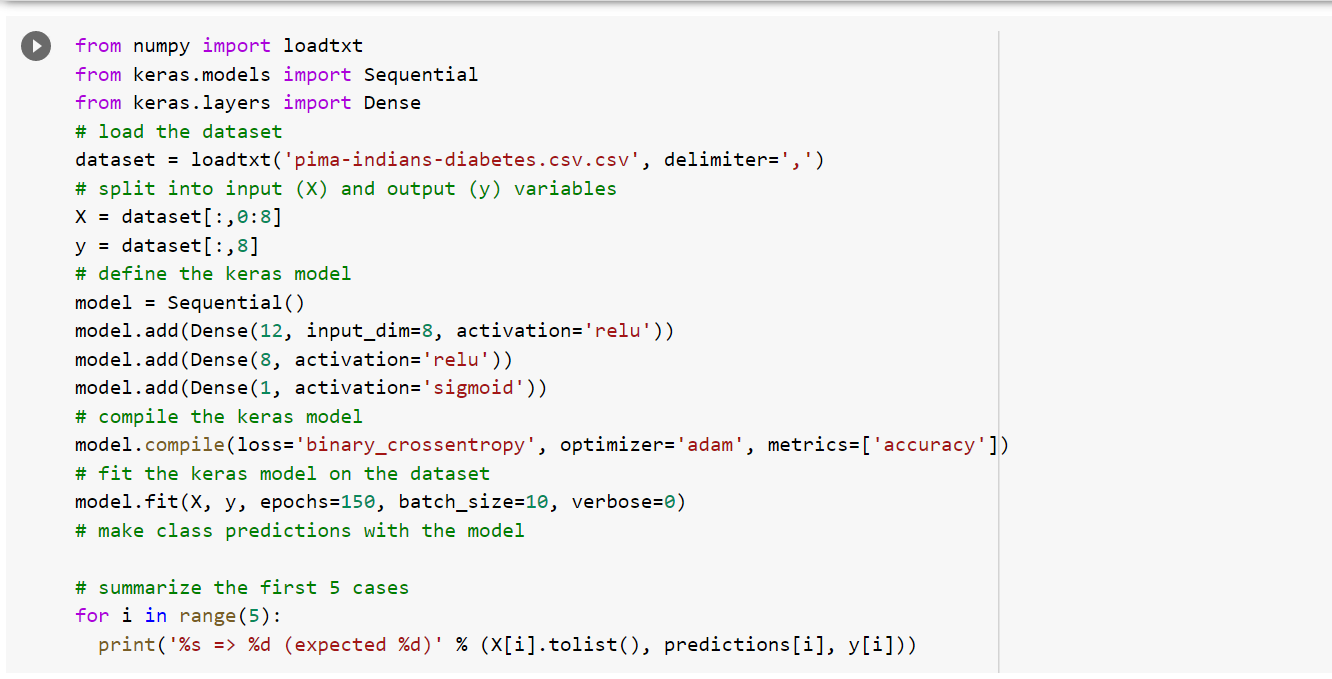
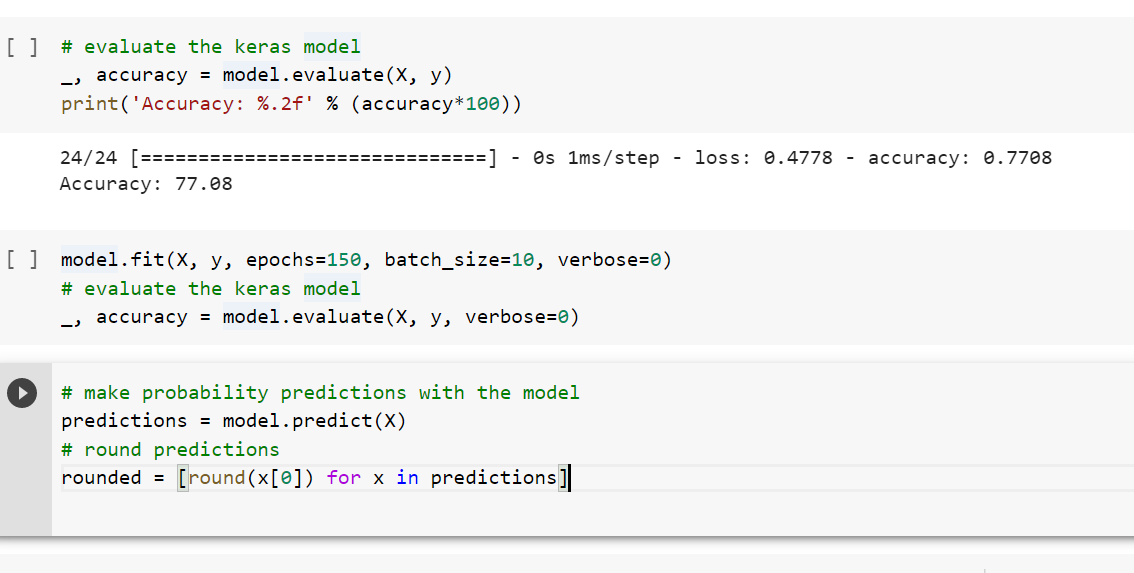
model.fit(X, y, epochs=150, batch\_size=10)

# fit the keras model on the dataset

model.fit(X, y, epochs=150, batch\_size=10)

**Output:**





**Output:**

