QUESTION 1

Write a Python Program to Solve N-Queen Problem without using Recursion.

SOURCE CODE

```
def is_safe(board, row, col, N):
    # Check if there is a queen in the same column
    for i in range(row):
        if board[i][col] == 1:
            return False

    # Check if there is a queen in the left diagonal
    for i, j in zip(range(row, -1, -1), range(col, -1, -1)):
        if board[i][j] == 1:
            return False

    # Check if there is a queen in the right diagonal
    for i, j in zip(range(row, -1, -1), range(col, N)):
        if board[i][j] == 1:
            return False

    return True

def print_solution(board, N):
    for i in range(N):
        for j in range(N):
            print(board[i][j], end=" ")
        print()

def solve_n_queens(N):
    board = [[0 for _ in range(N)] for _ in range(N)]
    stack = []
    row, col = 0, 0

    while row < N:
        while col < N:
            if is_safe(board, row, col, N):
                board[row][col] = 1
                stack.append(col)
                break
            col += 1

        if col == N:
            if not stack:
                break
            col = stack.pop()
```

```
            row -= 1
            board[row][col] = 0

    col += 1
        elif row == N - 1:
            # A solution is found
            print_solution(board, N)
            col = stack.pop()
            row -= 1
            board[row][col] = 0
            col += 1
        else:
            row += 1
            col = 0

# Example usage for N = 8
solve_n_queens(8)
```

OUTPUT

```
1  0  0  0  0  0  0  0
0  0  0  0  1  0  0  0
0  0  0  0  0  0  0  1
0  0  0  0  0  1  0  0
0  0  1  0  0  0  0  0
0  0  0  0  0  0  1  0
0  1  0  0  0  0  0  0
0  0  0  1  0  0  0  0
```

QUESTION 2

Write a Python Program to implement the Backtracking approach to solve N Queen's problem

SOURCE CODE

```
def print_solution(board):
    for row in board:
        print(" ".join(row))

def is_safe(board, row, col, N):
    # Check if there is a queen in the same column
    for i in range(row):
        if board[i][col] == 'Q':
            return False

    # Check upper left diagonal
    for i, j in zip(range(row, -1, -1), range(col, -1, -1)):
        if board[i][j] == 'Q':
            return False

    # Check upper right diagonal
    for i, j in zip(range(row, -1, -1), range(col, N)):
        if board[i][j] == 'Q':
            return False

    return True

def solve_n_queens_util(board, row, N):
    if row == N:
        # All queens are placed successfully, print the
solution
        print_solution(board)
        print()
        return

    for col in range(N):
        if is_safe(board, row, col, N):
            # Place the queen
            board[row][col] = 'Q'

            # Recur to place queens in the remaining rows
            solve_n_queens_util(board, row + 1, N)

            # Backtrack: remove the queen from the current
            # position
```

```
                board[row][col] = '.'


def solve_n_queens(N):
    # Initialize an empty chessboard
    board = [['.' for _ in range(N)] for _ in range(N)]

    solve_n_queens_util(board, 0, N)

# Example usage for N = 4
solve_n_queens(4)
```

OUTPUT

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

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

QUESTION 3

Write a Python Program to implement Min-Max Algorithm

SOURCE CODE

```
import math
def fun_minmax(cd,node,maxt,scr,td):
    if cd==td:
        return scr[node]
    if (maxt):

        return
max(fun_minmax(cd+1,node*2,False,scr,td),fun_minmax(cd+1,nod
e*2+1,False,scr,td))

    else:

        return
min(fun_minmax(cd+1,node*2,True,scr,td),fun_minmax(cd+1,node
*2+1,True,scr,td))
scr=[]
x=int(input("enter the total no. of leaf node"))
for i in range(x):
    y=int(input("enter the leaf value"))
    scr.append(y)
td=math.log(len(scr),2)
cd=int(input("enter the current depth value"))
nodev=int(input("enter the node value"))
maxt=True
print("The answer is")
answer=fun_minmax(cd,nodev,maxt,scr,td)
print(answer)
```

OUTPUT

```
Enter the total no. of leaf node 8
enter the leaf value 1
enter the leaf value 2
enter the leaf value 3
enter the leaf value 4
enter the leaf value 5
enter the leaf value 6
enter the leaf value 7
enter the leaf value 8
enter the current depth value 0
enter the node value 0

The answer is 6
```

QUESTION 4

Write a Python Program to implement Alpha-Beta Pruning Algorithm

SOURCE CODE

```
maximum,minimum=1000,-1000

def fun_alphabeta(d,node,maxt,v,A,B):
    if d==3:
        return v[node]
    if maxt:
        best=minimum
        for i in range(0,2):
            value=fun_alphabeta(d+1,node*2+i,False,v,A,B)
            best=max(best,value)
            A=max(A,best)
            if B<=A:
                break
        return best
    else:
        best=maximum
        for i in range(0,2):
            value=fun_alphabeta(d+1,node*2+i,True,v,A,B)
            best=min(best,value)
            A=min(A,best)
            if B<=A:
                break
        return best
scr=[]
x=int(input("enter the total no. of leaf node"))
for i in range(x):
    y=int(input("enter the leaf value"))
    scr.append(y)

d=int(input("enter the depth value"))
node=int(input("enter the node value"))


answer=fun_alphabeta(d,node,True,scr,minimum, maximum)
print(answer)
```

OUTPUT

```
enter the total no. of leaf node8
enter the leaf value1
enter the leaf value2
enter the leaf value3
enter the leaf value4
enter the leaf value5
enter the leaf value6
enter the leaf value7
enter the leaf value8
enter the depth value0
enter the node value0
6
```

QUESTION 5

Write a Python Program to implement Breadth First Search

SOURCE CODE

```
from collections import deque

class Graph:
    def __init__(self):
        self.graph = {}

    def add_edge(self, start, end):
        if start not in self.graph:
            self.graph[start] = []
        self.graph[start].append(end)

    def bfs(self, start):
        visited = set()
        queue = deque([start])

        while queue:
            current_node = queue.popleft()
            if current_node not in visited:
                print("Visiting node:", current_node)
                visited.add(current_node)

                neighbors = self.graph.get(current_node, [])
                queue.extend(neighbor for neighbor in
neighbors if neighbor not in visited)

# Example usage:
graph = Graph()
graph.add_edge('A', 'B')
graph.add_edge('A', 'C')
graph.add_edge('B', 'D')
graph.add_edge('C', 'E')
graph.add_edge('D', 'E')

start_node = 'A'

print(f"Breadth-First Search from {start_node}:")
graph.bfs(start_node)
```

OUTPUT

```
Breadth-First Search from A:
Visiting node: A
Visiting node: B
Visiting node: C
Visiting node: D
Visiting node: E
```

QUESTION 6

Write a Python Program to implement Depth First Search

SOURCE CODE

```
graph={
    'A':['B','C'],
    'B':['D','E'],
    'C':['F'],
    'D':[],
    'E':['F'],
    'F':[]
}
visited=set()      #set to keep track of visited nodes

def dfs(visited,graph,node):          #function for dfs
    if node not in visited:
        print(node)
        visited.add(node)
        for neighbour in graph[node]:
            #print("neighbour",neighbour)
            dfs(visited,graph,neighbour)

#driver code
print("The following is the depth first search")
dfs(visited,graph,'A')               #function calling
```

OUTPUT

```
The following is the depth first search
A
B
D
E
F
C
```

QUESTION 7

Write a Python Program to implement Iterative Deepening Depth First search (IDDFS)

SOURCE CODE

```
graph={
    'A':['B','C'],
    'B':['D','E'],
    'C':['G'],
    'D':[],
    'E':['F'],
    'F':[],
    'G':[]
}
def DFS(currentnode,destination,graph,maxdepth):
    print("checking for destination",currentnode)
    if currentnode==destination:
        return True
    if maxdepth<=0:
         return False
    for node in graph[currentnode]:
        if DFS(node,destination,graph,maxdepth-1):
            return True
    return False

def iterativeDDFS(currentnode,destination,graph,maxdepth):
    for i in range(maxdepth):
        if DFS(currentnode,destination,graph,i):
            return True
    return False
if not iterativeDDFS('A','E',graph,4):
    print("path is not avilable")
else:
    print("path exist")
```

OUTPUT

```
checking for destination A
checking for destination A
checking for destination B
checking for destination C
checking for destination A
checking for destination B
checking for destination D
checking for destination E
path exist
```

QUESTION 8

Write a Python Program to implement Best First Search

SOURCE CODE

```
from queue import PriorityQueue

class Graph:
    def __init__(self):
        self.graph = {}

    def add_edge(self, start, end, cost):
        if start not in self.graph:
            self.graph[start] = []
        self.graph[start].append((end, cost))

    def best_first_search(self, start, goal):
        visited = set()
        priority_queue = PriorityQueue()
        priority_queue.put((0, start))

        while not priority_queue.empty():
            current_cost, current_node =
                 priority_queue.get()

            if current_node in visited:
                continue

            print("Visiting node:", current_node)

            visited.add(current_node)

            if current_node == goal:
                print("Goal reached!")
                break

            neighbors = self.graph.get(current_node, [])
            for neighbor, cost in neighbors:
                if neighbor not in visited:
                    priority_queue.put((cost, neighbor))

# Example usage:
graph = Graph()
graph.add_edge('A', 'B', 5)
graph.add_edge('A', 'C', 10)
graph.add_edge('B', 'D', 3)
```

```
graph.add_edge('C', 'E', 7)


graph.add_edge('D', 'E', 2)

start_node = 'A'
goal_node = 'E'

print(f"Best-First Search from {start_node} to
{goal_node}:")
graph.best_first_search(start_node, goal_node)
```

OUTPUT

```
Best-First Search from A to E:
Visiting node: A
Visiting node: B
Visiting node: D
```

QUESTION 9

Write a Python Program to implement A* Algorithm

SOURCE CODE

```
import heapq

class Node:
    def __init__(self, row, col, cost, parent=None):
        self.row = row
        self.col = col
        self.cost = cost
        self.parent = parent

    def __lt__(self, other):
        return self.cost < other.cost

def heuristic(node, goal):
    # Example: Manhattan distance heuristic
    return abs(node.row - goal.row) + abs(node.col -
goal.col)

def astar(grid, start, goal):
    rows, cols = len(grid), len(grid[0])
    visited = set()
    pq = []
    heapq.heappush(pq, start)

    while pq:
        current = heapq.heappop(pq)

        if (current.row, current.col) == (goal.row,
goal.col):
            path = []
            while current:
                path.append((current.row, current.col))
                current = current.parent
            return path[::-1]

        visited.add((current.row, current.col))

        neighbors = [
            (current.row - 1, current.col),
            (current.row + 1, current.col),
            (current.row, current.col - 1),
            (current.row, current.col + 1)
```

```
        ]

        for neighbor_row, neighbor_col in neighbors:
            if 0 <= neighbor_row < rows and 0 <=
neighbor_col < cols and grid[neighbor_row][neighbor_col] ==
0 and (neighbor_row, neighbor_col) not in visited:
                neighbor = Node(neighbor_row, neighbor_col,
current.cost + 1, current)
                neighbor_cost = neighbor.cost +
heuristic(neighbor, goal)
                heapq.heappush(pq, neighbor)

    return None

# Example usage
grid = [
    [0, 0, 0, 0, 0],
    [0, 1, 0, 1, 0],
    [0, 1, 0, 1, 0],
    [0, 0, 0, 0, 0],
]

start = Node(0, 0, 0)
goal = Node(3, 4, 0)

path = astar(grid, start, goal)
print("A* Path:", path)
```

OUTPUT

```
A* Path: [(0, 0), (1, 0), (2, 0), (3, 0), (3, 1), (3, 2),
(3, 3), (3, 4)]
```

QUESTION 10

Write a Python Program to implement AO* Algorithm

SOURCE CODE

```
import heapq

class Node:
    def __init__(self, row, col, cost, parent=None):
        self.row = row
        self.col = col
        self.cost = cost
        self.parent = parent

    def __lt__(self, other):
        return self.cost < other.cost

def heuristic(node, goal):
    # Example: Manhattan distance heuristic
    return abs(node.row - goal.row) + abs(node.col -
goal.col)

def a_star(grid, start, goal):
    rows, cols = len(grid), len(grid[0])
    visited = set()
    pq = []
    heapq.heappush(pq, start)

    while pq:
        current = heapq.heappop(pq)

        if (current.row, current.col) == (goal.row,
goal.col):
            path = []
            while current:
                path.append((current.row, current.col))
                current = current.parent
            return path[::-1]

        visited.add((current.row, current.col))

        neighbors = [
            (current.row - 1, current.col),
            (current.row + 1, current.col),
            (current.row, current.col - 1),
            (current.row, current.col + 1)
```

```
        ]

        for neighbor_row, neighbor_col in neighbors:
            if 0 <= neighbor_row < rows and 0 <=
neighbor_col < cols and grid[neighbor_row][neighbor_col] ==
0 and (neighbor_row, neighbor_col) not in visited:
                neighbor = Node(neighbor_row, neighbor_col,
current.cost + 1, current)
                neighbor_cost = neighbor.cost +
heuristic(neighbor, goal)
                heapq.heappush(pq, neighbor)

    return None

def ao_star(grid, start, goal, epsilon):
    best_path = a_star(grid, start, goal)
    while True:
        cost_bound = best_path[-1][0] + epsilon if best_path
else float('inf')
        suboptimal_path = a_star_with_bound(grid, start,
goal, cost_bound)
        if not suboptimal_path or suboptimal_path[-1][0] >=
cost_bound:
            break
        best_path = suboptimal_path
    return best_path

def a_star_with_bound(grid, start, goal, cost_bound):
    rows, cols = len(grid), len(grid[0])
    visited = set()
    pq = []
    heapq.heappush(pq, start)

    while pq:
        current = heapq.heappop(pq)

        if (current.row, current.col) == (goal.row,
goal.col):
            path = []
            while current:
                path.append((current.row, current.col))
                current = current.parent
            return path[::-1]

        visited.add((current.row, current.col))

        neighbors = [
            (current.row - 1, current.col),
```

```
            (current.row + 1, current.col),
            (current.row, current.col - 1),
            (current.row, current.col + 1)
        ]

        for neighbor_row, neighbor_col in neighbors:
            if 0 <= neighbor_row < rows and 0 <= 
neighbor_col < cols and grid[neighbor_row][neighbor_col] == 
0 and (neighbor_row, neighbor_col) not in visited:
                neighbor = Node(neighbor_row, neighbor_col, 
current.cost + 1, current)
                neighbor_cost = neighbor.cost + 
heuristic(neighbor, goal)
                if neighbor_cost < cost_bound:
                    heapq.heappush(pq, neighbor)

    return None

# Example usage
grid = [
    [0, 0, 0, 0, 0],
    [0, 1, 0, 1, 0],
    [0, 1, 0, 1, 0],
    [0, 0, 0, 0, 0],
]

start = Node(0, 0, 0)
goal = Node(3, 4, 0)

epsilon = 2
result = ao_star(grid, start, goal, epsilon)
print("AO* Path:", result)
```

OUTPUT

```
AO* Path: [(0, 0), (1, 0), (2, 0), (3, 0), (3, 1), (3, 2), 
(3, 3), (3, 4)]
```

QUESTION 11

Write a Python Program to implement K-Nearest Neighbor Algorithm for data classification, choose dataset of your own choice

SOURCE CODE

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Load the Iris dataset
iris = datasets.load_iris()
X = iris.data
y = iris.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Initialize the K-Nearest Neighbors classifier with k=3
knn_classifier = KNeighborsClassifier(n_neighbors=3)

# Train the classifier
knn_classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred = knn_classifier.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Example: Make a prediction for a new data point
new_data_point = [[5.0, 3.5, 1.5, 0.2]]
prediction = knn_classifier.predict(new_data_point)
print("Prediction for the new data point:",
iris.target_names[prediction[0]])
```

OUTPUT

```
Accuracy: 1.0
Prediction for the new data point: setosa
```

QUESTION 12

Write a Python Program to implement Naïve Bayes Algorithm for data classification, choose dataset of your own choice

SOURCE CODE

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score,
classification_report

# Load the Iris dataset
iris = datasets.load_iris()
X = iris.data
y = iris.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Initialize the Naive Bayes classifier
naive_bayes_classifier = GaussianNB()

# Train the classifier
naive_bayes_classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred = naive_bayes_classifier.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Display classification report
print("Classification Report:\n",
classification_report(y_test, y_pred))

# Example: Make a prediction for a new data point
new_data_point = [[5.0, 3.5, 1.5, 0.2]]
prediction = naive_bayes_classifier.predict(new_data_point)
print("Prediction for the new data point:",
iris.target_names[prediction[0]])
```

OUTPUT

```
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30

Prediction for the new data point: setosa
```

QUESTION 13

Write a Python Program to implement Decision Trees for data classification, choose data set of your own choice

SOURCE CODE

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score,
classification_report
from sklearn.tree import export_text

# Load the Iris dataset
iris = datasets.load_iris()
X = iris.data
y = iris.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Initialize the Decision Tree classifier
decision_tree_classifier =
DecisionTreeClassifier(random_state=42)

# Train the classifier
decision_tree_classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred = decision_tree_classifier.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Display classification report
print("Classification Report:\n",
classification_report(y_test, y_pred))

# Display the Decision Tree rules
tree_rules = export_text(decision_tree_classifier,
feature_names=iris.feature_names)
print("Decision Tree Rules:\n", tree_rules)

# Example: Make a prediction for a new data point
new_data_point = [[5.0, 3.5, 1.5, 0.2]]
```

```
prediction =
decision_tree_classifier.predict(new_data_point)
print("Prediction for the new data point:",
iris.target_names[prediction[0]])
```

OUTPUT

```
Accuracy: 1.0
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30

Decision Tree Rules:
 |--- petal length (cm) <= 2.45
|   |--- class: 0
|--- petal length (cm) >  2.45
|   |--- petal length (cm) <= 4.75
|   |   |--- petal width (cm) <= 1.65
|   |   |   |--- class: 1
|   |   |--- petal width (cm) >  1.65
|   |   |   |--- class: 2
|   |--- petal length (cm) >  4.75
|   |   |--- petal width (cm) <= 1.75
|   |   |   |--- petal length (cm) <= 4.95
|   |   |   |   |--- class: 1
|   |   |   |--- petal length (cm) >  4.95
|   |   |   |   |--- petal width (cm) <= 1.55
|   |   |   |   |   |--- class: 2
|   |   |   |   |--- petal width (cm) >  1.55
|   |   |   |   |   |--- petal length (cm) <= 5.45
|   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |--- petal length (cm) >  5.45
|   |   |   |   |   |   |--- class: 2
|   |   |--- petal width (cm) >  1.75
|   |   |   |--- petal length (cm) <= 4.85
|   |   |   |   |--- sepal width (cm) <= 3.10
|   |   |   |   |   |--- class: 2
|   |   |   |   |--- sepal width (cm) >  3.10
|   |   |   |   |   |--- class: 1
|   |   |   |--- petal length (cm) >  4.85
|   |   |   |   |--- class: 2

Prediction for the new data point: setosa
```

QUESTION 14

Write a Python Program to implement Logistic Regression for data classification, choose dataset of your own choice

SOURCE CODE

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,
classification_report

# Load the Breast Cancer Wisconsin dataset
breast_cancer = datasets.load_breast_cancer()
X = breast_cancer.data
y = breast_cancer.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Initialize the Logistic Regression classifier
logistic_regression_classifier =
LogisticRegression(random_state=42)

# Train the classifier
logistic_regression_classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred = logistic_regression_classifier.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Display classification report
print("Classification Report:\n",
classification_report(y_test, y_pred))

# Example: Make a prediction for a new data point
new_data_point = [X_test[0]]  # Using the first data point
from the test set as an example
prediction =
logistic_regression_classifier.predict(new_data_point)
print("Prediction for the new data point:", "Malignant" if
prediction[0] == 1 else "Benign")
```

OUTPUT

```
Accuracy: 0.956140350877193
Classification Report:
               precision    recall  f1-score   support

           0       0.97      0.91      0.94        43
           1       0.95      0.99      0.97        71

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114

Prediction for the new data point: Malignant

/usr/local/lib/python3.8/dist-
packages/sklearn/linear_model/_logistic.py:460:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the
data as shown in:
    https://scikit-
learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative
solver options:
    https://scikit-
learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

QUESTION 15

Write a Python Program to implement Support Vector Machines for data classification, choose dataset of your own choice

SOURCE CODE

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score,
classification_report

# Load the Iris dataset
iris = datasets.load_iris()
X = iris.data
y = iris.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Initialize the Support Vector Machines classifier
svm_classifier = SVC(kernel='linear', C=1.0,
random_state=42)

# Train the classifier
svm_classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred = svm_classifier.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Display classification report
print("Classification Report:\n",
classification_report(y_test, y_pred))

# Example: Make a prediction for a new data point
new_data_point = [[5.0, 3.5, 1.5, 0.2]]
prediction = svm_classifier.predict(new_data_point)
print("Prediction for the new data point:",
iris.target_names[prediction[0]])
```

OUTPUT

```
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30

Prediction for the new data point: setosa
```

QUESTION 16

Write a Python Program to implement Linear regression on a dataset of your own choice

SOURCE CODE

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# Load the Boston Housing dataset
boston = datasets.load_boston()
X = boston.data
y = boston.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Initialize the Linear Regression model
linear_regression_model = LinearRegression()

# Train the model
linear_regression_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = linear_regression_model.predict(X_test)

# Calculate mean squared error and R^2 score
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)
print("R^2 Score:", r2)

# Example: Make a prediction for a new data point
new_data_point = np.array([0.00632, 18.0, 2.31, 0, 0.538,
6.575, 65.2, 4.09, 1, 296, 15.3, 396.9, 4.98]).reshape(1, -
1)
prediction = linear_regression_model.predict(new_data_point)
print("Predicted Price for the new data point:",
prediction[0])
```

OUTPUT

```
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30

Prediction for the new data point: setosa
```

QUESTION 17

Write a Python Program to implement Polynomial Regression on a dataset of your own choice

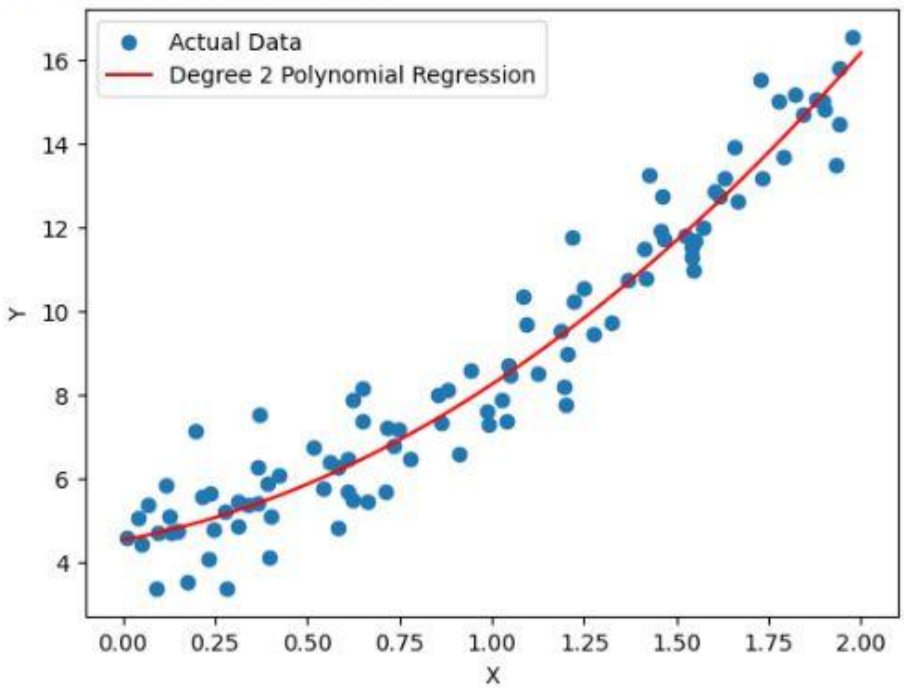
SOURCE CODE

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Generate a hypothetical dataset
np.random.seed(42)
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + 1.5 * X**2 + np.random.randn(100, 1)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Apply Polynomial Regression
degree = 2  # Degree of the polynomial
poly_features = PolynomialFeatures(degree=degree,
include_bias=False)
X_train_poly = poly_features.fit_transform(X_train)

# Initialize and train the Polynomial Regression model
poly_regression_model = LinearRegression()
poly_regression_model.fit(X_train_poly, y_train)

# Make predictions on the test set
X_test_poly = poly_features.transform(X_test)
y_pred = poly_regression_model.predict(X_test_poly)

# Calculate mean squared error and R^2 score
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)
print("R^2 Score:", r2)

# Plot the results
plt.scatter(X, y, label='Actual Data')
x_plot = np.linspace(0, 2, 100).reshape(-1, 1)
```

```
x_plot_poly = poly_features.transform(x_plot)
y_plot = poly_regression_model.predict(x_plot_poly)
plt.plot(x_plot, y_plot, color='red', label=f'Degree
{degree} Polynomial Regression')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.show()
```

OUTPUT

```
Mean Squared Error: 0.6358406072820804
R^2 Score: 0.9511063423293336
```



QUESTION 18

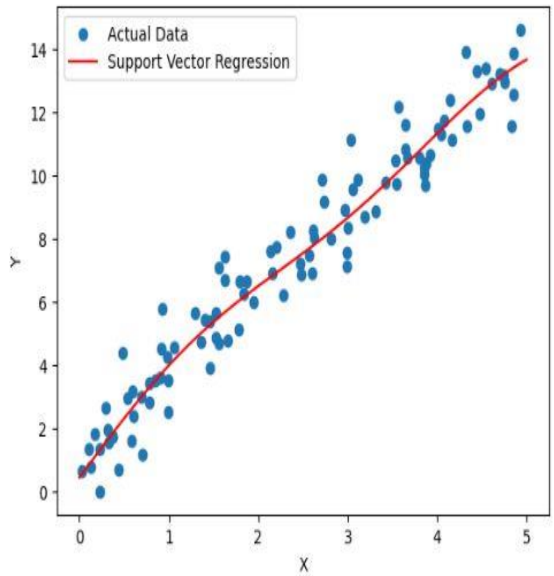
Write a Python Program to implement Support Vector Regression on a dataset of your own choice

SOURCE CODE

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, r2_score

# Generate a hypothetical dataset
np.random.seed(42)
X = 5 * np.random.rand(100, 1)
y = 3 * X + np.sin(X) + np.random.randn(100, 1)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Initialize the Support Vector Regression model
svr_model = SVR(kernel='rbf', C=100, gamma=0.1, epsilon=0.1)

# Train the model
svr_model.fit(X_train, y_train.ravel())

# Make predictions on the test set
y_pred = svr_model.predict(X_test)

# Calculate mean squared error and R^2 score
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)
print("R^2 Score:", r2)

# Plot the results
plt.scatter(X, y, label='Actual Data')
x_plot = np.linspace(0, 5, 100).reshape(-1, 1)
y_plot = svr_model.predict(x_plot)
plt.plot(x_plot, y_plot, color='red', label='Support Vector
Regression')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.show()
```

OUTPUT

```
Mean Squared Error: 0.7057530819594434
R^2 Score: 0.9557452312697399
```



QUESTION 19

Write a Python Program to implement Artificial Neural Network for data classification, choose dataset of your own choice

SOURCE CODE

```
import numpy as np
import tensorflow as tf
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

# Load the breast cancer dataset
cancer = load_breast_cancer()
X = cancer.data
y = cancer.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Standardize the feature values
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Build the ANN model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(32, activation='relu',
input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(16, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=50, batch_size=32,
verbose=1)

# Evaluate the model on the test set
y_pred_prob = model.predict(X_test)
y_pred = np.round(y_pred_prob).flatten()
```

```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Example: Make a prediction for a new data point
new_data_point = np.array(X_test[0]).reshape(1, -1)
prediction_prob = model.predict(new_data_point)
prediction = np.round(prediction_prob).flatten()[0]

print("Prediction for the new data point:", prediction)
```

OUTPUT

```
Epoch 1/50
15/15 [==============================] - 1s 7ms/step - loss:
0.7394 - accuracy: 0.4374
Epoch 2/50
15/15 [==============================] - 0s 2ms/step - loss:
0.4940 - accuracy: 0.7604
Epoch 3/50
15/15 [==============================] - 0s 3ms/step - loss:
0.3444 - accuracy: 0.9275
Epoch 4/50
15/15 [==============================] - 0s 2ms/step - loss:
0.2470 - accuracy: 0.9516
Epoch 5/50
15/15 [==============================] - 0s 3ms/step - loss:
0.1878 - accuracy: 0.9582
Epoch 6/50
15/15 [==============================] - 0s 2ms/step - loss:
0.1517 - accuracy: 0.9648
Epoch 7/50
15/15 [==============================] - 0s 4ms/step - loss:
0.1302 - accuracy: 0.9670
Epoch 8/50
15/15 [==============================] - 0s 4ms/step - loss:
0.1144 - accuracy: 0.9714
Epoch 9/50
15/15 [==============================] - 0s 3ms/step - loss:
0.1033 - accuracy: 0.9714
Epoch 10/50
15/15 [==============================] - 0s 3ms/step - loss:
0.0938 - accuracy: 0.9736
Epoch 11/50
15/15 [==============================] - 0s 1ms/step - loss:
0.0857 - accuracy: 0.9736
Epoch 12/50
```

```
15/15 [==============================] - 0s 2ms/step - loss: 0.0794 - accuracy: 0.9780
Epoch 13/50
15/15 [==============================] - 0s 2ms/step - loss: 0.0744 - accuracy: 0.9802
Epoch 14/50

15/15 [==============================] - 0s 2ms/step - loss: 0.0701 - accuracy: 0.9890
Epoch 15/50
15/15 [==============================] - 0s 4ms/step - loss: 0.0660 - accuracy: 0.9868
Epoch 16/50
15/15 [==============================] - 0s 3ms/step - loss: 0.0625 - accuracy: 0.9868
Epoch 17/50
15/15 [==============================] - 0s 2ms/step - loss: 0.0596 - accuracy: 0.9868
Epoch 18/50
15/15 [==============================] - 0s 2ms/step - loss: 0.0571 - accuracy: 0.9868
Epoch 19/50
15/15 [==============================] - 0s 2ms/step - loss: 0.0549 - accuracy: 0.9890
Epoch 20/50
15/15 [==============================] - 0s 2ms/step - loss: 0.0525 - accuracy: 0.9890
Epoch 21/50
15/15 [==============================] - 0s 2ms/step - loss: 0.0507 - accuracy: 0.9890
Epoch 22/50
15/15 [==============================] - 0s 2ms/step - loss: 0.0487 - accuracy: 0.9890
Epoch 23/50
15/15 [==============================] - 0s 2ms/step - loss: 0.0473 - accuracy: 0.9890
Epoch 24/50
15/15 [==============================] - 0s 3ms/step - loss: 0.0453 - accuracy: 0.9890
Epoch 25/50
15/15 [==============================] - 0s 3ms/step - loss: 0.0440 - accuracy: 0.9890
Epoch 26/50
15/15 [==============================] - 0s 3ms/step - loss: 0.0424 - accuracy: 0.9912
Epoch 27/50
15/15 [==============================] - 0s 3ms/step - loss: 0.0411 - accuracy: 0.9912
```

```
Epoch 28/50
15/15 [==============================] - 0s 2ms/step - loss:
0.0394 - accuracy: 0.9912
Epoch 29/50
15/15 [==============================] - 0s 2ms/step - loss:
0.0388 - accuracy: 0.9912
Epoch 30/50
15/15 [==============================] - 0s 3ms/step - loss:
0.0369 - accuracy: 0.9934
Epoch 31/50
15/15 [==============================] - 0s 2ms/step - loss:
0.0355 - accuracy: 0.9934
Epoch 32/50
15/15 [==============================] - 0s 2ms/step - loss:
0.0347 - accuracy: 0.9934
Epoch 33/50
15/15 [==============================] - 0s 4ms/step - loss:
0.0330 - accuracy: 0.9934
Epoch 34/50
15/15 [==============================] - 0s 3ms/step - loss:
0.0321 - accuracy: 0.9934
Epoch 35/50
15/15 [==============================] - 0s 3ms/step - loss:
0.0306 - accuracy: 0.9934
Epoch 36/50
15/15 [==============================] - 0s 2ms/step - loss:
0.0296 - accuracy: 0.9934
Epoch 37/50
15/15 [==============================] - 0s 2ms/step - loss:
0.0286 - accuracy: 0.9934
Epoch 38/50
15/15 [==============================] - 0s 2ms/step - loss:
0.0275 - accuracy: 0.9934
Epoch 39/50
15/15 [==============================] - 0s 3ms/step - loss:
0.0267 - accuracy: 0.9934
Epoch 40/50
15/15 [==============================] - 0s 2ms/step - loss:
0.0256 - accuracy: 0.9934
Epoch 41/50
15/15 [==============================] - 0s 2ms/step - loss:
0.0248 - accuracy: 0.9934
Epoch 42/50
15/15 [==============================] - 0s 2ms/step - loss:
0.0243 - accuracy: 0.9934
Epoch 43/50
15/15 [==============================] - 0s 1ms/step - loss:
0.0234 - accuracy: 0.9934
```

```
Epoch 44/50
15/15 [==============================] - 0s 3ms/step - loss:
0.0224 - accuracy: 0.9934
Epoch 45/50
15/15 [==============================] - 0s 3ms/step - loss:
0.0215 - accuracy: 0.9934
Epoch 46/50
15/15 [==============================] - 0s 3ms/step - loss:
0.0209 - accuracy: 0.9934
Epoch 47/50
15/15 [==============================] - 0s 3ms/step - loss:
0.0202 - accuracy: 0.9934
Epoch 48/50
15/15 [==============================] - 0s 1ms/step - loss:
0.0197 - accuracy: 0.9934
Epoch 49/50
15/15 [==============================] - 0s 2ms/step - loss:
0.0188 - accuracy: 0.9956
Epoch 50/50
15/15 [==============================] - 0s 1ms/step - loss:
0.0181 - accuracy: 0.9956
4/4 [==============================] - 0s 1ms/step
Accuracy: 0.9824561403508771
1/1 [==============================] - 0s 12ms/step
Prediction for the new data point: 1.0
```

QUESTION 20

Write a Python Program to implement Feed Forward Neural Network on a given dataset for data classification, choose dataset of your own choice

SOURCE CODE

```
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_classification
from sklearn.metrics import accuracy_score

# Generate a hypothetical dataset for binary classification
X, y = make_classification(n_samples=1000, n_features=10,
n_informative=8, n_redundant=2, random_state=42)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Standardize the feature values
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Build the FNN model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(32, activation='relu',
input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(16, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=50, batch_size=32,
verbose=1)

# Evaluate the model on the test set
y_pred_prob = model.predict(X_test)
y_pred = np.round(y_pred_prob).flatten()
```

```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Example: Make a prediction for a new data point
new_data_point = np.random.rand(1, X_train.shape[1])  #
Replace this with your new data point
prediction_prob = model.predict(new_data_point)
prediction = np.round(prediction_prob).flatten()[0]

print("Prediction for the new data point:", prediction)
```

OUTPUT

```
Epoch 1/50
25/25 [==============================] - 0s 951us/step -
loss: 0.6983 - accuracy: 0.5800
Epoch 2/50
25/25 [==============================] - 0s 2ms/step - loss:
0.6119 - accuracy: 0.6737
Epoch 3/50
25/25 [==============================] - 0s 1ms/step - loss:
0.5614 - accuracy: 0.7475
Epoch 4/50
25/25 [==============================] - 0s 1ms/step - loss:
0.5192 - accuracy: 0.7862
Epoch 5/50
25/25 [==============================] - 0s 3ms/step - loss:
0.4826 - accuracy: 0.8025
Epoch 6/50
25/25 [==============================] - 0s 2ms/step - loss:
0.4488 - accuracy: 0.8250
Epoch 7/50
25/25 [==============================] - 0s 1ms/step - loss:
0.4164 - accuracy: 0.8438
Epoch 8/50
25/25 [==============================] - 0s 2ms/step - loss:
0.3886 - accuracy: 0.8550
Epoch 9/50
25/25 [==============================] - 0s 2ms/step - loss:
0.3603 - accuracy: 0.8637
Epoch 10/50
25/25 [==============================] - 0s 1ms/step - loss:
0.3360 - accuracy: 0.8800
Epoch 11/50
25/25 [==============================] - 0s 2ms/step - loss:
0.3121 - accuracy: 0.8825
```

```
Epoch 12/50
25/25 [==============================] - 0s 1ms/step - loss:
0.2889 - accuracy: 0.8913
Epoch 13/50
25/25 [==============================] - 0s 2ms/step - loss:
0.2698 - accuracy: 0.9013
Epoch 14/50
25/25 [==============================] - 0s 2ms/step - loss:
0.2531 - accuracy: 0.9075
Epoch 15/50
25/25 [==============================] - 0s 997us/step -
loss: 0.2397 - accuracy: 0.9175
Epoch 16/50
25/25 [==============================] - 0s 2ms/step - loss:
0.2260 - accuracy: 0.9187
Epoch 17/50
25/25 [==============================] - 0s 1ms/step - loss:
0.2146 - accuracy: 0.9212
Epoch 18/50
25/25 [==============================] - 0s 2ms/step - loss:
0.2052 - accuracy: 0.9262
Epoch 19/50
25/25 [==============================] - 0s 1ms/step - loss:
0.1995 - accuracy: 0.9275
Epoch 20/50
25/25 [==============================] - 0s 2ms/step - loss:
0.1909 - accuracy: 0.9300
Epoch 21/50
25/25 [==============================] - 0s 962us/step -
loss: 0.1859 - accuracy: 0.9362
Epoch 22/50
25/25 [==============================] - 0s 2ms/step - loss:
0.1803 - accuracy: 0.9375
Epoch 23/50
25/25 [==============================] - 0s 1ms/step - loss:
0.1749 - accuracy: 0.9425
Epoch 24/50
25/25 [==============================] - 0s 2ms/step - loss:
0.1691 - accuracy: 0.9450
Epoch 25/50
25/25 [==============================] - 0s 2ms/step - loss:
0.1656 - accuracy: 0.9450
Epoch 26/50
25/25 [==============================] - 0s 1ms/step - loss:
0.1619 - accuracy: 0.9463
Epoch 27/50
25/25 [==============================] - 0s 2ms/step - loss:
0.1581 - accuracy: 0.9425
```

```
Epoch 28/50
25/25 [==============================] - 0s 3ms/step - loss:
0.1546 - accuracy: 0.9475
Epoch 29/50
25/25 [==============================] - 0s 3ms/step - loss:
0.1523 - accuracy: 0.9450
Epoch 30/50
25/25 [==============================] - 0s 3ms/step - loss:
0.1485 - accuracy: 0.9500
Epoch 31/50
25/25 [==============================] - 0s 2ms/step - loss:
0.1447 - accuracy: 0.9488
Epoch 32/50
25/25 [==============================] - 0s 1ms/step - loss:
0.1420 - accuracy: 0.9488
Epoch 33/50
25/25 [==============================] - 0s 1ms/step - loss:
0.1400 - accuracy: 0.9488
Epoch 34/50
25/25 [==============================] - 0s 2ms/step - loss:
0.1369 - accuracy: 0.9488
Epoch 35/50
25/25 [==============================] - 0s 2ms/step - loss:
0.1350 - accuracy: 0.9488
Epoch 36/50
25/25 [==============================] - 0s 1000us/step -
loss: 0.1324 - accuracy: 0.9525
Epoch 37/50
25/25 [==============================] - 0s 2ms/step - loss:
0.1296 - accuracy: 0.9513
Epoch 38/50
25/25 [==============================] - 0s 2ms/step - loss:
0.1284 - accuracy: 0.9538
Epoch 39/50
25/25 [==============================] - 0s 1ms/step - loss:
0.1246 - accuracy: 0.9563
Epoch 40/50
25/25 [==============================] - 0s 2ms/step - loss:
0.1229 - accuracy: 0.9563
Epoch 41/50
25/25 [==============================] - 0s 2ms/step - loss:
0.1198 - accuracy: 0.9575
Epoch 42/50
25/25 [==============================] - 0s 949us/step -
loss: 0.1192 - accuracy: 0.9588
Epoch 43/50
25/25 [==============================] - 0s 2ms/step - loss:
0.1168 - accuracy: 0.9575
```

```
Epoch 44/50
25/25 [==============================] - 0s 2ms/step - loss:
0.1151 - accuracy: 0.9600
Epoch 45/50
25/25 [==============================] - 0s 1ms/step - loss:
0.1121 - accuracy: 0.9625
Epoch 46/50
25/25 [==============================] - 0s 3ms/step - loss:
0.1091 - accuracy: 0.9650
Epoch 47/50
25/25 [==============================] - 0s 2ms/step - loss:
0.1081 - accuracy: 0.9650
Epoch 48/50
25/25 [==============================] - 0s 2ms/step - loss:
0.1065 - accuracy: 0.9638
Epoch 49/50
25/25 [==============================] - 0s 3ms/step - loss:
0.1044 - accuracy: 0.9663
Epoch 50/50
25/25 [==============================] - 0s 3ms/step - loss:
0.1028 - accuracy: 0.9688
7/7 [==============================] - 0s 916us/step
Accuracy: 0.905
1/1 [==============================] - 0s 20ms/step
Prediction for the new data point: 0.0
```

QUESTION 21

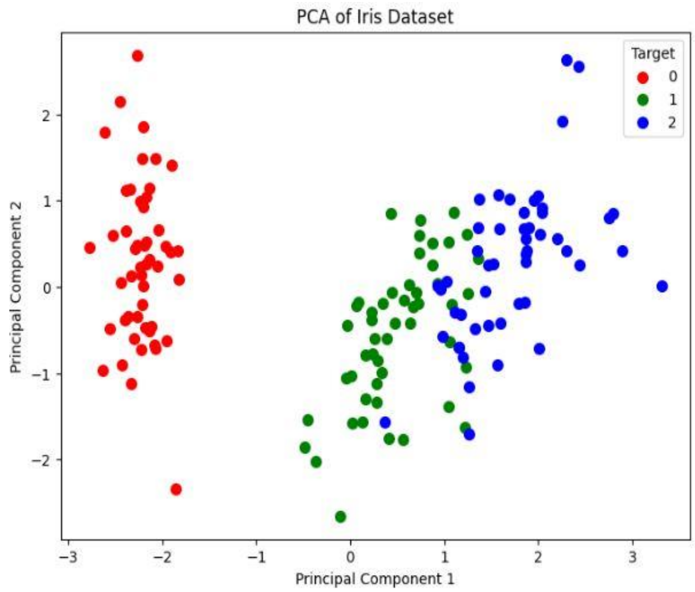
Write a Python Program to implement Principal Component Analysis on a dataset of your own choice

SOURCE CODE

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Load the Iris dataset
iris = datasets.load_iris()
X = iris.data
y = iris.target

# Standardize the feature values
X_standardized = StandardScaler().fit_transform(X)

# Apply PCA
pca = PCA(n_components=2)
principal_components = pca.fit_transform(X_standardized)

# Create a DataFrame for visualization
df_pca = pd.DataFrame(data=principal_components,
columns=['Principal Component 1', 'Principal Component 2'])
df_pca['Target'] = y

# Visualize the results
plt.figure(figsize=(8, 6))
targets = [0, 1, 2]
colors = ['r', 'g', 'b']

for target, color in zip(targets, colors):
    indices_to_keep = df_pca['Target'] == target
    plt.scatter(df_pca.loc[indices_to_keep, 'Principal
Component 1'],
                df_pca.loc[indices_to_keep, 'Principal
Component 2'],
                c=color, s=50)

plt.title('PCA of Iris Dataset')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
```

```
plt.legend(targets, title='Target')
plt.show()
```

OUTPUT



QUESTION 22

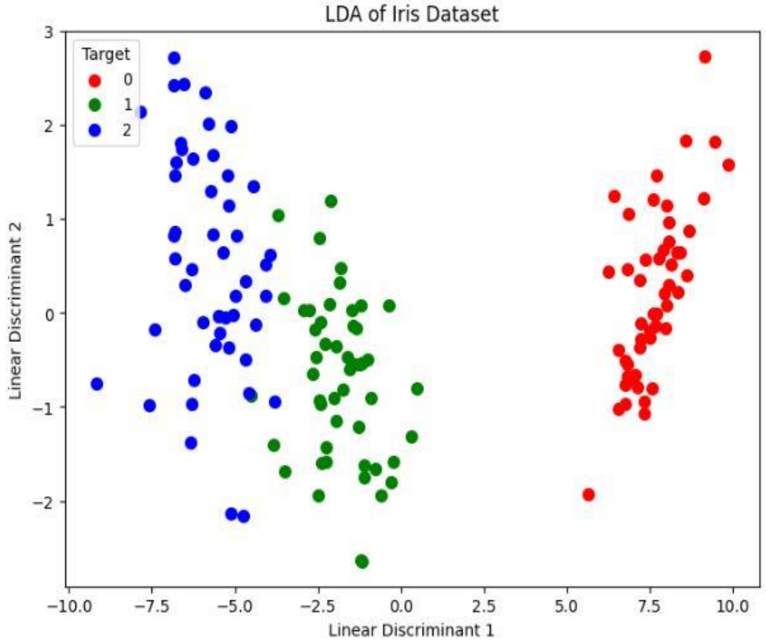
Write a Python Program to implement Linear Discriminant Analysis on a dataset of your own choice

SOURCE CODE

```
class Node:
    def __init__(self, state, cost, heuristic, parent=None):
        self.state = state
        self.cost = cost
        self.heuristic = heuristic
        self.parent = parent

def ida_star_search(initial_state, heuristic, goal_state):
    bound = heuristic(initial_state, goal_state)
    path = [Node(initial_state, 0, bound)]

    while True:
        result, new_bound = depth_limited_search(path,
goal_state, heuristic, bound)
        if result == "found":
            return path
        if result == "not_found":
            return None
        bound = new_bound

def depth_limited_search(path, goal_state, heuristic,
bound):
    node = path[-1]
    cost = node.cost + node.heuristic
    if cost > bound:
        return "not_found", cost
    if node.state == goal_state:
        return "found", bound

    min_bound = float('inf')

    successors = get_successors(node, goal_state, heuristic)
    for successor in successors:
        if successor.cost + successor.heuristic > bound:
            min_bound = min(min_bound, successor.cost +
successor.heuristic)
        else:
            path.append(successor)
            result, new_bound = depth_limited_search(path,
goal_state, heuristic, bound)
```

```
            if result == "found":
                return "found", bound
            min_bound = min(min_bound, new_bound)
            path.pop()

    return "not_found", min_bound

def get_successors(node, goal_state, heuristic):
    # Example: Generating successors for a simple 8-puzzle
problem
    successors = []
    state = node.state
    # Code for generating successors based on the specific
problem
    # ...

    return successors

# Example usage
def eight_puzzle_heuristic(state, goal_state):
    # Example: Manhattan distance heuristic for the 8-puzzle
problem
    # ...

# Specify your initial state, goal state, and heuristic
function
initial_state = ...
goal_state = ...
heuristic_function = eight_puzzle_heuristic

result = ida_star_search(initial_state, heuristic_function,
goal_state)

if result:
    print("IDA* Path:")
    for node in result:
        print("State:", node.state, "Cost:", node.cost,
"Heuristic:", node.heuristic)
else:
    print("No solution found.")
```

OUTPUT



LDA of Iris Dataset
Target
0
1
2
Linear Discriminant 2
Linear Discriminant 1
3
2
1
0
−1
−2
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

QUESTION 23

Write a Python Program to implement Apriori Algorithm on a dataset of your own choice

SOURCE CODE

```
# Make sure to install mlxtend library
# You can install it using: pip install mlxtend

from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori,
association_rules
import pandas as pd

# Sample retail dataset
dataset = [['Milk', 'Bread', 'Eggs'],
           ['Milk', 'Cheese'],
           ['Cheese', 'Bread', 'Butter'],
           ['Milk', 'Bread', 'Butter'],
           ['Eggs', 'Bread']]

# Convert the dataset into a transaction format
te = TransactionEncoder()
te_ary = te.fit(dataset).transform(dataset)
df = pd.DataFrame(te_ary, columns=te.columns_)

# Apply Apriori algorithm
frequent_itemsets = apriori(df, min_support=0.4,
use_colnames=True)

# Generate association rules
rules = association_rules(frequent_itemsets,
metric="confidence", min_threshold=0.7)

# Display frequent itemsets
print("Frequent Itemsets:")
print(frequent_itemsets)

# Display association rules
print("\nAssociation Rules:")
print(rules)
```

OUTPUT

```
requent Itemsets:
   support        itemsets
0      0.6         (Bread)
1      0.4        (Cheese)
2      0.4          (Eggs)
3      0.8          (Milk)
4      0.4   (Eggs, Bread)
5      0.4   (Bread, Milk)
6      0.4  (Cheese, Milk)

Association Rules:
  antecedents consequents  antecedent support  consequent
support  support  \
0      (Eggs)     (Bread)                 0.4
0.6      0.4
1    (Cheese)      (Milk)                 0.4
0.8      0.4

   confidence      lift  leverage  conviction  zhangs_metric
0         1.0  1.666667      0.16         inf       0.666667
1         1.0  1.250000      0.08         inf       0.333333
```

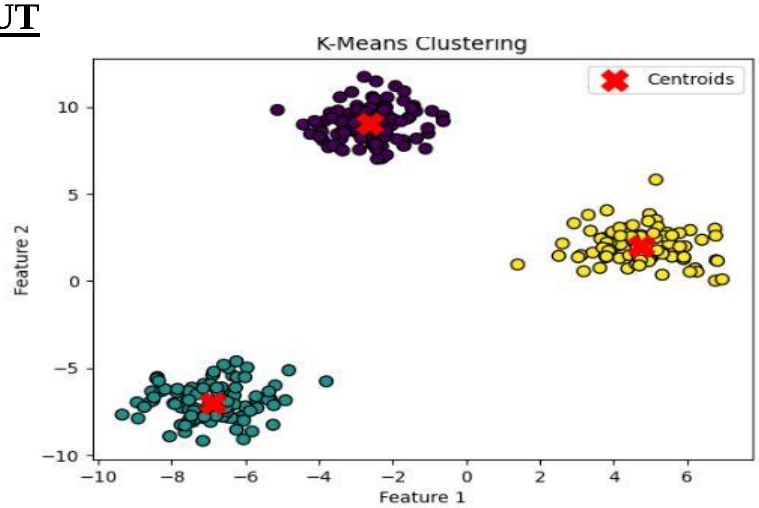
QUESTION 24

Write a Python Program to implement K-Means Algorithm on a dataset of your own choice

SOURCE CODE

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs

# Generate a hypothetical dataset with three clusters
X, y = make_blobs(n_samples=300, centers=3, random_state=42)

# Apply K-Means algorithm
kmeans = KMeans(n_clusters=3, random_state=42)
y_kmeans = kmeans.fit_predict(X)

# Visualize the results
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, cmap='viridis',
edgecolor='k', s=50)
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='red',
marker='X', s=200, label='Centroids')
plt.title('K-Means Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show()
```

OUTPUT



QUESTION 25

Write a Python Program to implement DBSCAN Algorithm on a dataset of your own choice

SOURCE CODE

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN
from sklearn.datasets import make_blobs
from sklearn.preprocessing import StandardScaler

# Generate a hypothetical dataset with varying densities
X, y = make_blobs(n_samples=300, centers=3, cluster_std=1.0,
random_state=42)

# Standardize the feature values
X = StandardScaler().fit_transform(X)

# Apply DBSCAN algorithm
dbscan = DBSCAN(eps=0.5, min_samples=5)
labels = dbscan.fit_predict(X)

# Visualize the results
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis',
edgecolor='k', s=50)
plt.title('DBSCAN Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```

OUTPUT