**Assignment 1 (A)**

**Installation and working of Python**

**1. Installing Python**

To install Python, follow these steps:

**On Windows:**

1. Visit the official website: <https://www.python.org/downloads/>.
2. Download the latest version of Python.
3. Run the installer.
   * Check the box "Add Python to PATH".
   * Click "Install Now".
4. Once installed, open the command prompt and type python --version to verify the installation.

**2. Running Python**

Once Python is installed, you can run Python code in two main ways:

1. **Using the Python Interpreter (Interactive Mode):**
   * Open a terminal or command prompt.
   * Type python or python3 (depending on your system).
   * This will open the Python shell, where you can type and execute Python commands interactively.
2. **Running a Python Script:**
   * Create a Python file with a .py extension, e.g., script.py.
   * Write your Python code in this file.
   * Run the script from the terminal:

**python script.py**

**Assignment 1 (B)**

**Program to print numbers that are not multiple of file**

**# Define a list of numbers**

A = [10, 23, 45, 67, 55, 78, 100, 36]

**# Iterate through the list and print numbers that are not multiples of 5**

print("Numbers that are not multiples of 5")

for number in A:

    if number % 5 != 0:

        print(number)

**OUTPUT:**

Numbers that are not multiples of 5

23

67

78

36

**Assignment 2 (A)**

**Program to find given integer can expressed as sum of two semi primes or not**

def is\_prime(n):

    if n <= 1:

        return False

    if n <= 3:

        return True

    if n % 2 == 0 or n % 3 == 0:

        return False

    for i in range(5, int(n\*\*0.5) + 1, 6):

        if n % i == 0 or n % (i + 2) == 0:

            return False

    return True

def is\_semi\_prime(n):

    for i in range(2, int(n\*\*0.5) + 1):

        if n % i == 0 and is\_prime(i) and is\_prime(n // i):

            return True

    return False

def can\_be\_expressed\_as\_sum\_of\_two\_semi\_primes(N):

    for i in range(2, N):

        if is\_semi\_prime(i) and is\_semi\_prime(N - i):

            return True

    return False

**# Example usage**

N = 30

if can\_be\_expressed\_as\_sum\_of\_two\_semi\_primes(N):

    print(f"{N} can be expressed as a sum of two semi-primes.")

else:

    print(f"{N} cannot be expressed as a sum of two semi-primes.")

**OUTPUT:**

30 can be expressed as a sum of two semi-primes.

**Assignment 2 (B)**

**Program to print square matrix in a counter-clockwise spiral form**

def counter\_clockwise\_spiral(matrix):

    if not matrix:

        return []

    result = []

    while matrix:

        # Add the first column

        for row in matrix:

            result.append(row.pop(0))

        # Remove empty rows

        matrix = [row for row in matrix if row]

        # Rotate the matrix counter-clockwise

        matrix = [list(row) for row in zip(\*matrix)][::-1]

    return result

**# Example usage:**

matrix = [

    [1, 2, 3, 4],

    [5, 6, 7, 8],

    [9, 10, 11, 12],

    [13, 14, 15, 16]

]

print(counter\_clockwise\_spiral(matrix))

**OUTPUT:**

[1, 5, 9, 13, 4, 3, 2, 16, 12, 8, 14, 15, 6, 10, 7, 11]

**Assignment 3 (A)**

**Illustrate lists and tuple functions**

**# List functions**

my\_list = [1, 2, 3, 4, 5]

my\_list.append(6)  # Adds an element to the end of the list

print("List after append:", my\_list)

my\_list.remove(3)  # Removes the first occurrence of the element

print("List after remove:", my\_list)

my\_list.reverse()  # Reverses the list

print("List after reverse:", my\_list)

**# Tuple functions**

my\_tuple = (1, 2, 3, 4, 5)

print("Tuple:", my\_tuple)

print("Count of 3 in tuple:", my\_tuple.count(3))  # Counts the number of occurrences of an element

print("Index of 4 in tuple:", my\_tuple.index(4))  # Finds the index of the first occurrence of an element

**OUTPUT:**

List after append: [1, 2, 3, 4, 5, 6]

List after remove: [1, 2, 4, 5, 6]

List after reverse: [6, 5, 4, 2, 1]

Tuple: (1, 2, 3, 4, 5)

Count of 3 in tuple: 1

Index of 4 in tuple: 3

**Assignment 3 (B)**

**Illustrate sets and dictionary functions**

**# Set functions**

my\_set = {1, 2, 3, 4, 5}

my\_set.add(6)  # Adds an element to the set

print("Set after add:", my\_set)

my\_set.remove(3)  # Removes an element from the set

print("Set after remove:", my\_set)

my\_set.update([7, 8, 9])  # Adds multiple elements to the set

print("Set after update:", my\_set)

**# Dictionary functions**

my\_dict = {'a': 1, 'b': 2, 'c': 3}

print("Dictionary:", my\_dict)

print("Keys in dictionary:", my\_dict.keys())  # Returns a view object of the dictionary's keys

print("Values in dictionary:", my\_dict.values())  # Returns a view object of the dictionary's values

print("Items in dictionary:", my\_dict.items())  # Returns a view object of the dictionary's key-value pairs

my\_dict.update({'d': 4})  # Updates the dictionary with the given key-value pairs

print("Dictionary after update:", my\_dict)

**OUTPUT:**

Set after add: {1, 2, 3, 4, 5, 6}

Set after remove: {1, 2, 4, 5, 6}

Set after update: {1, 2, 4, 5, 6, 7, 8, 9}

Dictionary: {'a': 1, 'b': 2, 'c': 3}

Keys in dictionary: dict\_keys(['a', 'b', 'c'])

Values in dictionary: dict\_values([1, 2, 3])

Items in dictionary: dict\_items([('a', 1), ('b', 2), ('c', 3)])

Dictionary after update: {'a': 1, 'b': 2, 'c': 3, 'd': 4}

**Assignment 4 (A)**

**Solve Two/three jug problem**

from collections import deque

def water\_jug\_bfs(jug1\_capacity, jug2\_capacity, target):

visited = set()

queue = deque([(0, 0)]) # (jug1, jug2)

visited.add((0, 0))

while queue:

jug1, jug2 = queue.popleft()

# If we reach the target amount in any of the jugs

if jug1 == target or jug2 == target:

return True

# All possible states from current state

possible\_states = [

(jug1\_capacity, jug2), # Fill jug1

(jug1, jug2\_capacity), # Fill jug2

(0, jug2), # Empty jug1

(jug1, 0), # Empty jug2

(min(jug1\_capacity, jug1 + jug2), max(0, jug2 - (jug1\_capacity - jug1))), # Pour jug2 -> jug1

(max(0, jug1 - (jug2\_capacity - jug2)), min(jug2\_capacity, jug1 + jug2)) # Pour jug1 -> jug2

]

# Enqueue the new states if not visited

for state in possible\_states:

if state not in visited:

visited.add(state)

queue.append(state)

return False

jug1\_capacity = 4

jug2\_capacity = 3

target = 2

print(water\_jug\_bfs(jug1\_capacity, jug2\_capacity, target))

**Output:** True

**Assignment 4 (B)**

**Implement Tic-tac-Toe**

def print\_board(board):

    for row in board:

        print(" | ".join(row))

        print("-" \* 5)

def check\_winner(board, player):

    win\_cond = [player] \* 3

    return any(row == win\_cond for row in board) or any(col == win\_cond for col in zip(\*board)) or \

           [board[i][i] for i in range(3)] == win\_cond or [board[i][2-i] for i in range(3)] == win\_cond

def play\_tic\_tac\_toe():

    board = [[' '] \* 3 for \_ in range(3)]

    players = ['X', 'O']

    for turn in range(9):

        print\_board(board)

        player = players[turn % 2]

        print(f"Player {player}'s turn")

        row, col = map(int, input("Enter row and col (0, 1, 2): ").split())

        if board[row][col] == ' ':

            board[row][col] = player

            if check\_winner(board, player):

                print\_board(board)

                print(f"Player {player} wins!")

                return

        else:

            print("Invalid move. Try again.")

            continue

    print("It's a draw!")

play\_tic\_tac\_toe()

**OUTPUT:**

| |

-----

| |

-----

| |

-----

Player X's turn

Enter row and col (0, 1, 2): 0 1

| X |

-----

| |

-----

| |

-----

Player O's turn

Enter row and col (0, 1, 2): 0 0

O | X |

-----

| |

-----

| |

-----

Player X's turn

Enter row and col (0, 1, 2): 1 1

O | X |

-----

| X |

-----

| |

-----

Player O's turn

Enter row and col (0, 1, 2): 2 1

O | X |

-----

| X |

-----

| O |

**Assignment 5**

**Illustrate NumPy and Pandas library**

import numpy as np

import pandas as pd

# NumPy Example

array = np.array([[1, 2, 3], [4, 5, 6]])

print("NumPy Array:")

print(array)

# Pandas Example

data = {'Name': ['Alice', 'Bob', 'Charlie'], 'Age': [25, 30, 35]}

df = pd.DataFrame(data)

print("\nPandas DataFrame:")

print(df)

**OUTPUT:**

NumPy Array:

[[1 2 3]

[4 5 6]]

Pandas DataFrame:

Name Age

0 Alice 25

1 Bob 30

2 Charlie 35

**Assignment 6(A)**

**Implement Depth First Search**

def dfs(graph, start, visited=None):

    if visited is None:

        visited = set()

    visited.add(start)

    print(start, end=' ')

    for neighbor in graph[start]:

        if neighbor not in visited:

            dfs(graph, neighbor, visited)

# Example usage:

graph = {

    'A': ['B', 'C'],

    'B': ['D', 'E'],

    'C': ['F'],

    'D': [],

    'E': ['F'],

    'F': []

}

print("DFS Traversal:")

dfs(graph, 'A')

**OUTPUT:**

DFS Traversal:

A B D E F C

**Assignment 6(B)**

**Implement Breadth First Search**

from collections import deque

def bfs(graph, start):

    visited = set()

    queue = deque([start])

    while queue:

        vertex = queue.popleft()

        if vertex not in visited:

            print(vertex, end=' ')

            visited.add(vertex)

            queue.extend(graph[vertex])

# Example usage:

graph = {

    'A': ['B', 'C'],

    'B': ['D', 'E'],

    'C': ['F'],

    'D': [],

    'E': ['F'],

    'F': []

}

print("\nBFS Traversal:")

bfs(graph, 'A')

**OUTPUT:**

BFS Traversal:

A B C D E F

**Assignment 6(C)**

**Implement A\* informed search**

from queue import PriorityQueue

def a\_star(graph, start, goal, heuristic):

pq = PriorityQueue()

pq.put((0, start))

came\_from = {start: None}

cost\_so\_far = {start: 0}

while not pq.empty():

current = pq.get()[1]

if current == goal:

break

for neighbor, cost in graph[current].items():

new\_cost = cost\_so\_far[current] + cost

if neighbor not in cost\_so\_far or new\_cost < cost\_so\_far[neighbor]:

cost\_so\_far[neighbor] = new\_cost

priority = new\_cost + heuristic[neighbor]

pq.put((priority, neighbor))

came\_from[neighbor] = current

# Reconstruct path

path = []

node = goal

while node:

path.append(node)

node = came\_from[node]

return path[::-1]

# Example usage:

graph = {

'A': {'B': 1, 'C': 3},

'B': {'D': 2, 'E': 4},

'C': {'F': 6},

'D': {},

'E': {'F': 1},

'F': {}

}

heuristic = {'A': 7, 'B': 6, 'C': 5, 'D': 3, 'E': 1, 'F': 0}

print("\nA\* Path:", a\_star(graph, 'A', 'F', heuristic))

**OUTPUT:**

A\* Path: ['A', 'B', 'E', 'F']

**Assignment 7(A)**

**Implementation of various knowledge representation techniques**

**#Propositional Logic**

class PropositionalLogic:

    def \_\_init\_\_(self):

        self.knowledge\_base = {}

    def add\_fact(self, statement, truth\_value):

        self.knowledge\_base[statement] = truth\_value

    def query(self, statement):

        return self.knowledge\_base.get(statement, None)

**# Example**

logic = PropositionalLogic()

logic.add\_fact("It is raining", True)

logic.add\_fact("I need an umbrella", True)

print(logic.query("It is raining"))

**#First order Predicate Logic**

class PredicateLogic:

    def \_\_init\_\_(self):

        self.facts = []

    def add\_fact(self, predicate, arguments):

        self.facts.append((predicate, arguments))

    def query(self, predicate, arguments):

        return any(

            p == predicate and args == arguments

            for p, args in self.facts

        )

**# Example**

pred\_logic = PredicateLogic()

pred\_logic.add\_fact("is\_mammal", ["dog"])

pred\_logic.add\_fact("is\_mammal", ["cat"])

print(pred\_logic.query("is\_mammal", ["dog"]))

**OUTPUT:**

True

True

**Assignment 7(B)**

**Working of Bayesian network**

import numpy as np

class BayesianNetwork:

    def \_\_init\_\_(self):

        # Network structure

        self.nodes = {}

        self.edges = {}

    def add\_node(self, name, prob\_table=None):

        """

        Add a node to the Bayesian network

        :param name: Node name

        :param prob\_table: Conditional probability table

        """

        self.nodes[name] = prob\_table if prob\_table is not None else {}

    def add\_edge(self, parent, child):

        """

        Add a directed edge between nodes

        :param parent: Parent node name

        :param child: Child node name

        """

        if child not in self.edges:

            self.edges[child] = []

        self.edges[child].append(parent)

    def set\_conditional\_prob(self, node, conditions, probability):

        """

        Set conditional probability for a node

        :param node: Node name

        :param conditions: Parent node conditions

        :param probability: Probability value

        """

self.nodes[node][conditions] = probability

    def inference(self, query\_node, evidence=None):

        """

        Perform probabilistic inference

        :param query\_node: Node to query

        :param evidence: Known evidence

        :return: Probability of query node

        """

        if evidence is None:

            evidence = {}

        # If no parents, return unconditional probability

        if query\_node not in self.edges or not self.edges[query\_node]:

            return self.nodes[query\_node].get(True, 0.5)

        # Calculate conditional probability based on parent states

        parents = self.edges[query\_node]

        relevant\_conditions = tuple(evidence.get(p, False) for p in parents)

        return self.nodes[query\_node].get(relevant\_conditions, 0.5)

# Example: Medical Diagnosis Network

def medical\_diagnosis\_example():

    bn = BayesianNetwork()

    # Add nodes

    bn.add\_node('Smoking', {True: 0.3, False: 0.7})

    bn.add\_node('Lung Cancer')

    bn.add\_node('X-Ray')

    # Add edges

    bn.add\_edge('Smoking', 'Lung Cancer')

    bn.add\_edge('Lung Cancer', 'X-Ray')

    # Set conditional probabilities

    bn.set\_conditional\_prob('Lung Cancer', (True,), 0.1)  # Smoking -> Lung Cancer

    bn.set\_conditional\_prob('Lung Cancer', (False,), 0.01)

    bn.set\_conditional\_prob('X-Ray', (True,), 0.8)  # Lung Cancer -> X-Ray

    bn.set\_conditional\_prob('X-Ray', (False,), 0.1)

    # Inference

    print("Probability of Lung Cancer given Smoking:",

          bn.inference('Lung Cancer', {'Smoking': True}))

    print("Probability of X-Ray given Smoking:",

          bn.inference('X-Ray', {'Smoking': True, 'Lung Cancer': True}))

medical\_diagnosis\_example()

**OUTPUT:**

Probability of Lung Cancer given Smoking: 0.1

Probability of X-Ray given Smoking: 0.8

**Assignment 8(A)**

**Implement Classification Techniques**

import pandas as pd

import sklearn

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

from sklearn.linear\_model import LogisticRegression

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

import seaborn as sns

import matplotlib.pyplot as plt

# Load dataset

data = pd.read\_csv('asthma.csv')  # Replace with your dataset

# Check the columns of the dataset

print("Columns in dataset:", data.columns)

# Set the target column to one of the severity columns, e.g., 'Severity\_Moderate'

target\_column = 'Severity\_Moderate'

if target\_column not in data.columns:

    print(f"Error: '{target\_column}' column not found in dataset")

else:

    # Preprocessing

    data = data.dropna()  # Drop missing values

    X = data.drop(target\_column, axis=1)  # Features

    y = data[target\_column]               # Target variable

    # Split data

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

    model = LogisticRegression(solver='liblinear')

    model.fit(X\_train, y\_train)

    # Predictions

    y\_pred = model.predict(X\_test)

    # Evaluate

    print("Accuracy:", accuracy\_score(y\_test, y\_pred))

    print("Classification Report:\n", classification\_report(y\_test, y\_pred))

    sns.heatmap(confusion\_matrix(y\_test, y\_pred), annot=True, fmt="d")

    plt.show()

**OUTPUT :**

Columns in dataset: Index(['Tiredness', 'Dry-Cough', 'Difficulty-in-Breathing', 'Sore-Throat',

'None\_Sympton', 'Pains', 'Nasal-Congestion', 'Runny-Nose',

'None\_Experiencing', 'Age\_0-9', 'Age\_10-19', 'Age\_20-24', 'Age\_25-59',

'Age\_60+', 'Gender\_Female', 'Gender\_Male', 'Severity\_Mild',

'Severity\_Moderate', 'Severity\_None'],

dtype='object')

Accuracy: 0.7467171717171717

Classification Report:

precision recall f1-score support

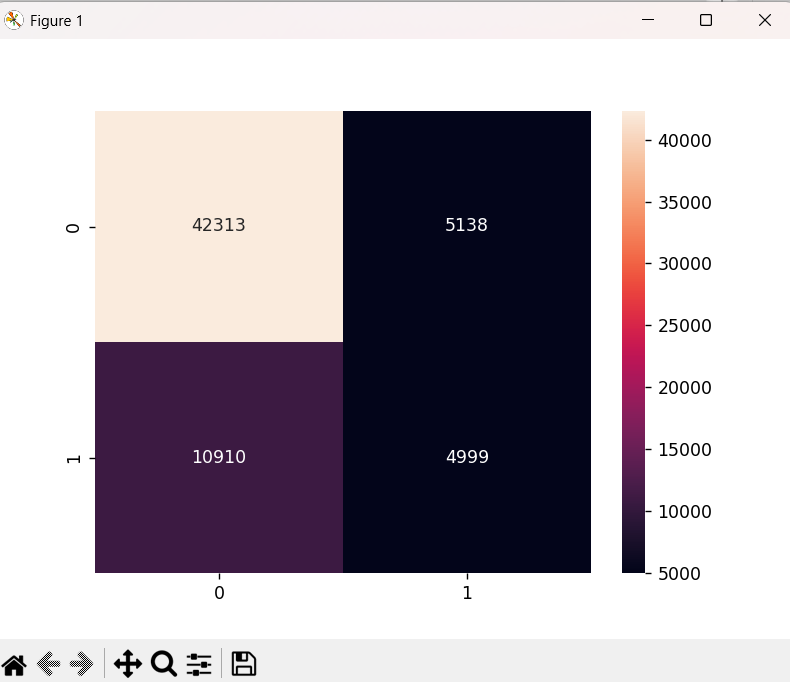
0 0.80 0.89 0.84 47451

1 0.49 0.31 0.38 15909

accuracy 0.75 63360

macro avg 0.64 0.60 0.61 63360

weighted avg 0.72 0.75 0.73 63360



**Assignment 8(B)**

**Implement Clustering Techniques**

from sklearn.cluster import KMeans

from sklearn.datasets import make\_blobs

import matplotlib.pyplot as plt

# Generate synthetic data

X, \_ = make\_blobs(n\_samples=300, centers=4, cluster\_std=0.60, random\_state=0)

# Model

kmeans = KMeans(n\_clusters=4, random\_state=42)

kmeans.fit(X)

# Predictions

clusters = kmeans.predict(X)

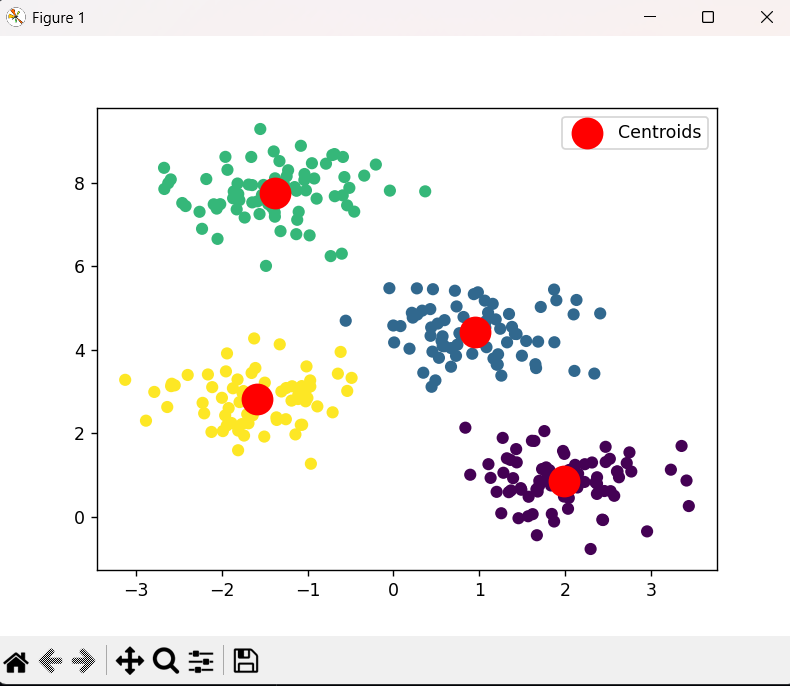
# Visualization

plt.scatter(X[:, 0], X[:, 1], c=clusters, cmap='viridis')

plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], s=300, c='red', label='Centroids')

plt.legend()

plt.show()



**Assignment 8(C)**

**Implement Regression Techniques**

import pandas as pd

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

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

import matplotlib.pyplot as plt

# Load dataset

data = pd.read\_csv('diabetes.csv')

# Preprocessing

X = data[['Glucose', 'BloodPressure']]  # Example feature columns

y = data['Outcome']                     # Target variable

# Split data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Model

model = LinearRegression()

model.fit(X\_train, y\_train)

# Predictions

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

# Evaluate

print("Mean Squared Error:", mean\_squared\_error(y\_test, y\_pred))

print("R-squared Score:", r2\_score(y\_test, y\_pred))

# Visualization

plt.scatter(y\_test, y\_pred)

plt.xlabel('Actual')

plt.ylabel('Predicted')

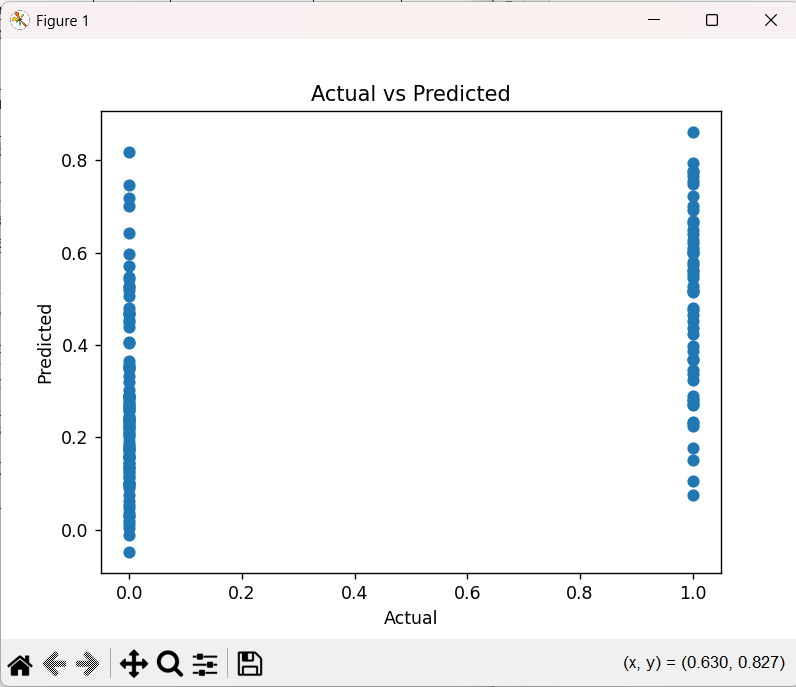
plt.title('Actual vs Predicted')

plt.show()

**OUTPUT:**

Mean Squared Error: 0.1714899803247617

R-squared Score: 0.253065863474371



**Assignment 9**

**Natural language processing tool development**

import nltk

import pandas as pd

from nltk.sentiment.vader import SentimentIntensityAnalyzer

nltk.downloader.download("vader\_lexicon")

sid=SentimentIntensityAnalyzer()

L=["Hello my name is Satyam Kulshreshtha", "I am a MCA 2nd Year Student", "i am happy", "My college name is Ajay Kumar Garg Engineering College "]

i=1

for data in L:

    print(i)

    ss = sid.polarity\_scores(data)

    print(data)

    i = i + 1

    for k in ss:

        print(k, ss[k])

**OUTPUT:**

[nltk\_data] Downloading package vader\_lexicon to

[nltk\_data] C:\Users\satya\AppData\Roaming\nltk\_data...

1

Hello my name is Satyam Kulshreshtha

neg 0.0

neu 1.0

pos 0.0

compound 0.0

2

I am a MCA 2nd Year Student

neg 0.0

neu 1.0

pos 0.0

compound 0.0

3

i am happy

neg 0.0

neu 0.213

pos 0.787

compound 0.5719

4

My college name is Ajay Kumar Garg Engineering College

neg 0.0

neu 1.0

pos 0.0

compound 0.0

**Assignment 10**

**Implementation of pattern recognition problems**

from sklearn import tree

from sklearn.datasets import load\_iris

from sklearn.metrics import accuracy\_score

import numpy

iris = load\_iris()

x\_coordinate = iris.data

y\_coordinate = iris.target

plant\_names = iris.target\_names

array\_ids= numpy.random.permutation(len(x\_coordinate))

x\_coordinate\_train = x\_coordinate [array\_ids[:-15]]

x\_coordinate\_real = x\_coordinate[array\_ids[-15:]]

y\_coordinate\_train = y\_coordinate[array\_ids[:-15]]

y\_coordinate\_real = y\_coordinate[array\_ids[-15:]]

data\_classification = tree. DecisionTreeClassifier()

data\_classification.fit(x\_coordinate\_train,y\_coordinate\_train)

prediction = data\_classification.predict(x\_coordinate\_real)

print(prediction)

print(y\_coordinate\_real)

print("Accuracy in percent: %.2f" %((accuracy\_score (prediction,y\_coordinate\_real)) \* 100))

**OUTPUT:**

[0 0 0 1 0 2 2 2 2 0 2 2 1 1 2]

[0 0 0 1 0 2 2 2 2 0 2 2 1 1 2]

Accuracy in percent: 100.00