Practical __

Writeup:

Practical __

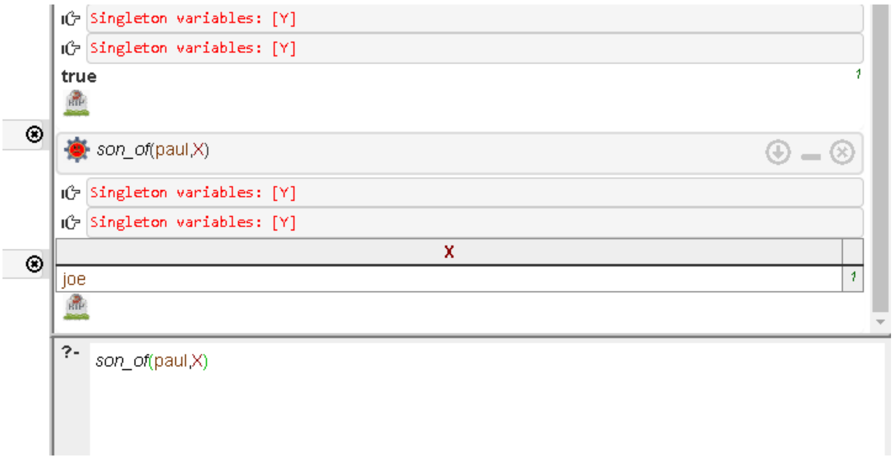
Aim:

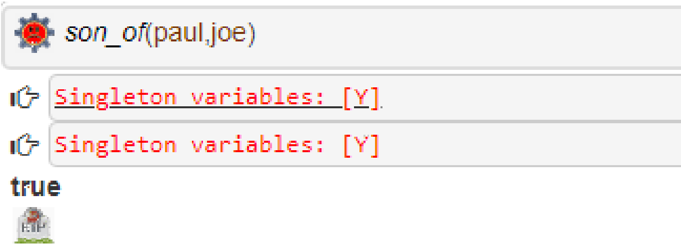
Description:

Code:

```
father(joe,paul)
father(joe,mary)
father(joe,hope)
mother(jane,paul)
mother(jane,mary)
mother(jane,hope)
male(paul)
male(joe)
male(raphl)
male(X):-father(X,Y)
female(mary)
female(jane)
female(hope)
female(X):-mother(X,Y)
son_of(X,Y):- father(Y,X),male(X)
son_of(X,Y):- mother(Y,X),male(X)
daughter_of(X,Y):- father(Y,X),female(X)
daughter_of(X,Y):- mother(Y,X),female(X)
sibling_of(X,Y):- father(Z,X),father(Z,Y),X\=Y
sibling_of(X,Y):- mother(Z,X),mother(Z,Y),X\=Y
```

Output:


Singleton variables: [Y]
Singleton variables: [Y]
true
son_of(paul,X)
Singleton variables: [Y]
Singleton variables: [Y]
X
joe
?-
son_of(paul,X)


son_of(paul,joe)
Singleton variables: [Y]
Singleton variables: [Y]
true

Practical __

Writeup:

Practical __

Aim:

Description:

Code:

```
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
quality = ctrl.Antecedent(np.arange(0, 11, 1), 'quality')
service = ctrl.Antecedent(np.arange(0, 11, 1), 'service')
tip = ctrl.Consequent(np.arange(0, 26, 1), 'tip')
quality.automf(3)
service.automf(3)
tip['low'] = fuzz.trimf(tip.universe, [0, 0, 13])
tip['medium'] = fuzz.trimf(tip.universe, [0, 13, 25])
tip['high'] = fuzz.trimf(tip.universe, [13, 25, 25])
quality['average'].view()
service.view()
tip.view()
rule1 = ctrl.Rule(quality['poor'] | service['poor'], tip['low'])
rule2 = ctrl.Rule(service['average'], tip['medium'])
rule3 = ctrl.Rule(service['good'] | quality['good'], tip['high'])
rule1.view()
tipping_ctrl = ctrl.ControlSystem([rule1, rule2, rule3])
tipping = ctrl.ControlSystemSimulation(tipping_ctrl)
tipping.input['quality'] = 6.5
tipping.input['service'] = 9.8
tipping.compute()
print(tipping.output['tip'])
tip.view(sim=tipping)
```

Output:

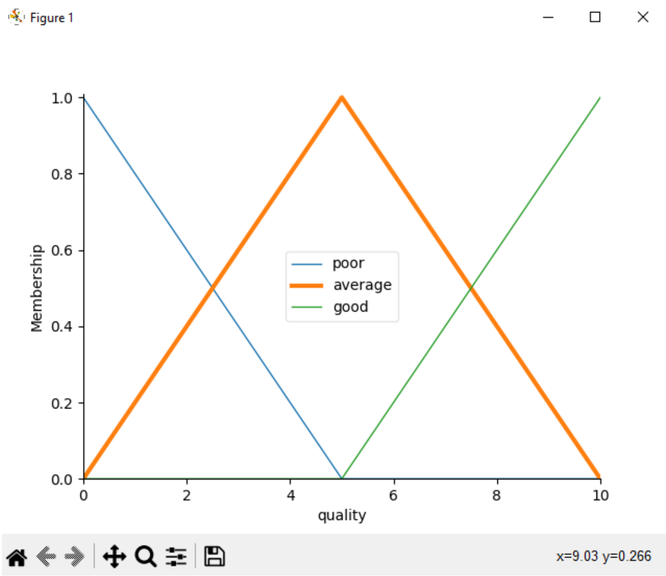


Figure 1
Membership
poor
average
good
quality
x=9.03 y=0.266

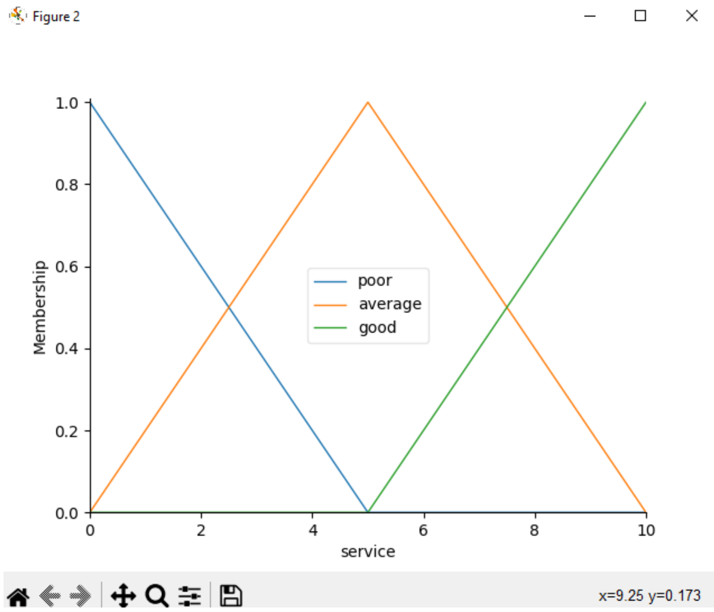


Figure 2
Membership
poor
average
good
service
x=9.25 y=0.173

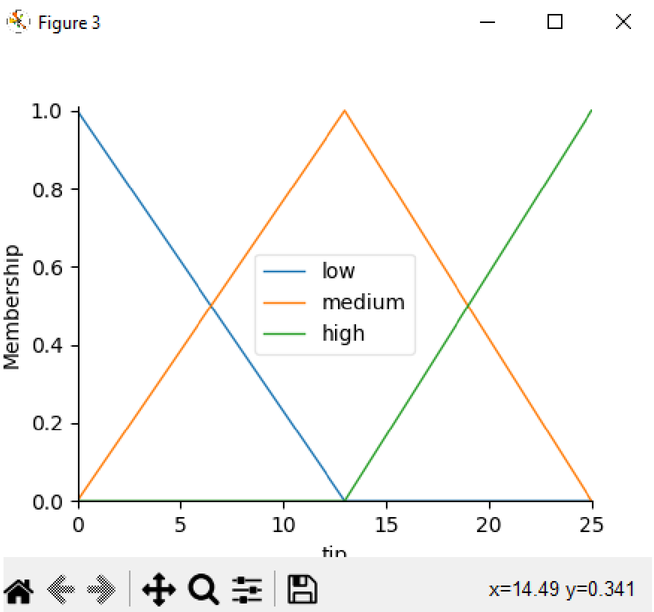


Figure 3
Membership
low
medium
high
x=14.49 y=0.341

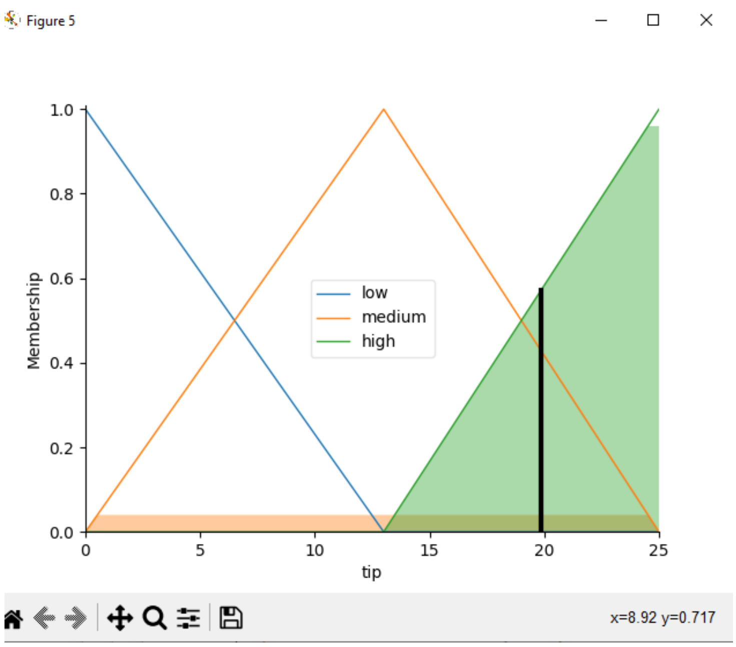


Figure 5
Membership
low
medium
high
tip
x=8.92 y=0.717

Practical __

Writeup:

Practical __

Aim:

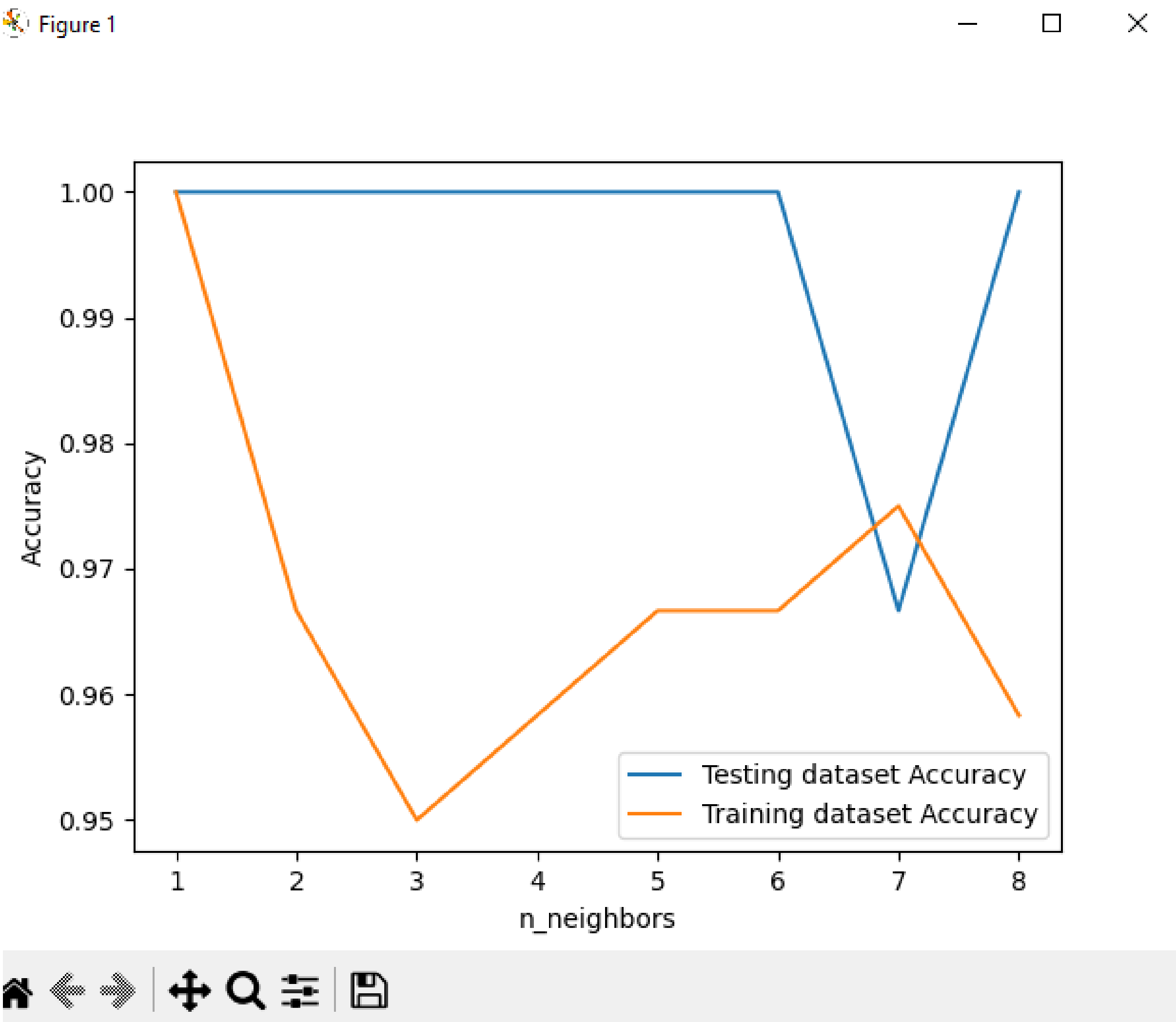
Description:

Code:

```
# Import necessary modules
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
import numpy as np
import matplotlib.pyplot as plt
irisData = load_iris()
# Create feature and target arrays
X = irisData.data
y = irisData.target
# Split into training and test set
X_train, X_test, y_train, y_test = train_test_split(
 X, y, test_size = 0.2, random_state=42)
neighbors = np.arange(1, 9)
train_accuracy = np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))
# Loop over K values
for i, k in enumerate(neighbors):
 knn = KNeighborsClassifier(n_neighbors=k)
 knn.fit(X_train, y_train)
# Compute training and test data accuracy
 train_accuracy[i] = knn.score(X_train, y_train)
 test_accuracy[i] = knn.score(X_test, y_test)
# Generate plot
plt.plot(neighbors, test_accuracy, label = 'Testing dataset Accuracy')
plt.plot(neighbors, train_accuracy, label = 'Training dataset Accuracy')
plt.legend()
plt.xlabel('n_neighbors')
```

```
plt.ylabel('Accuracy')
plt.show()
```

Output:



Practical __

Aim:

Description:

Code:

```
import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

customer_data = pd.read_csv("C:\Users\Sakib\Downloads\Mall_Customers.csv")

customer_data.shape

customer_data.head()

data = customer_data.iloc[:, 3:5].values

import scipy.cluster.hierarchy as shc

plt.figure(figsize=(10, 7))

plt.title("Customer Dendograms")

dend = shc.dendrogram(shc.linkage(data, method='ward'))

from sklearn.cluster import AgglomerativeClustering

cluster = AgglomerativeClustering()

cluster.fit_predict(data)

plt.figure(figsize=(10, 7))

plt.scatter(data[:,0], data[:,1], c=np.arctan2(data[:,0], data[:,1]), cmap='rainbow', s=50,
alpha=0.8)

plt.show()
```

Output:

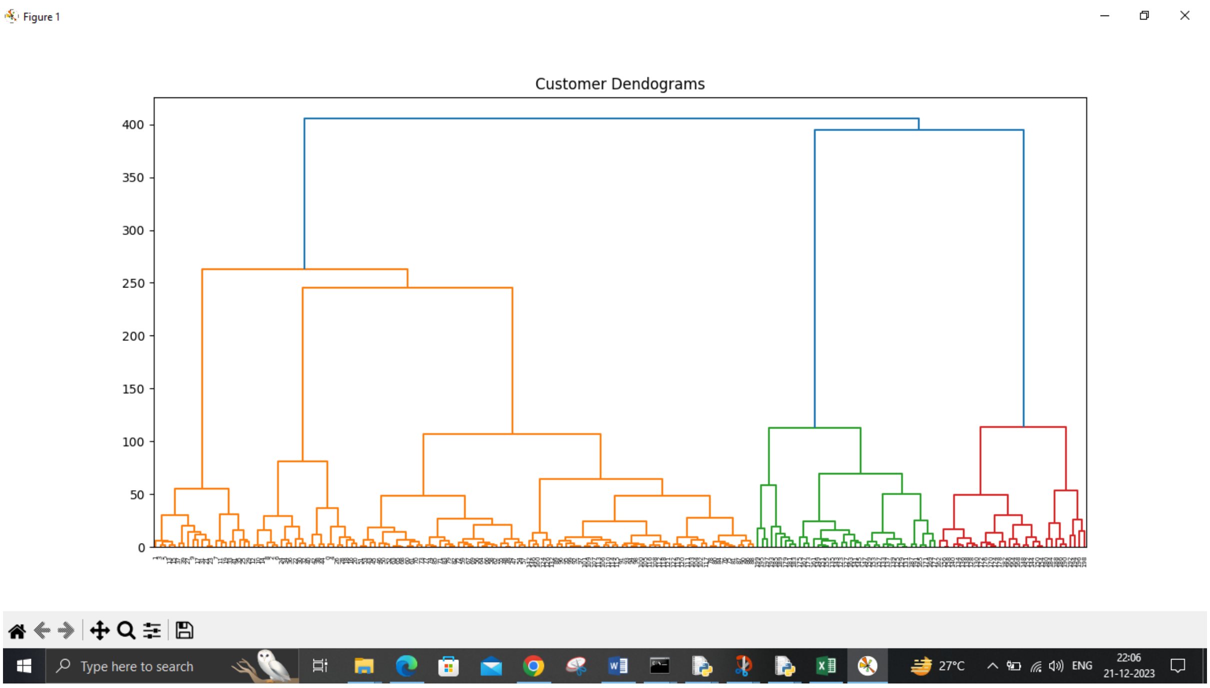


Figure 1
Customer Dendograms
400
350
300
250
200
150
100
50
0
Type here to search
27°C
ENG
22:06
21-12-2023

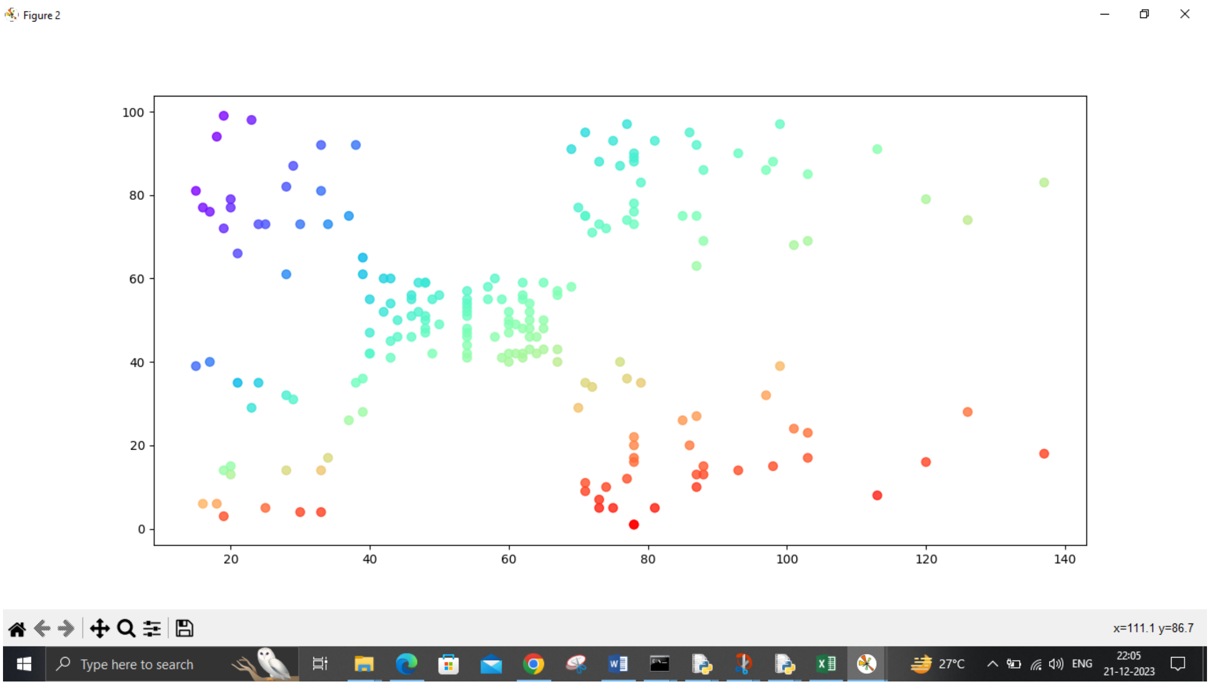


Figure 2
100
80
60
40
20
0
20
40
60
80
100
120
140
x=111.1 y=86.7
Type here to search
27°C
ENG
22:05
21-12-2023

Practical __

Writeup:

Practical __

Aim:

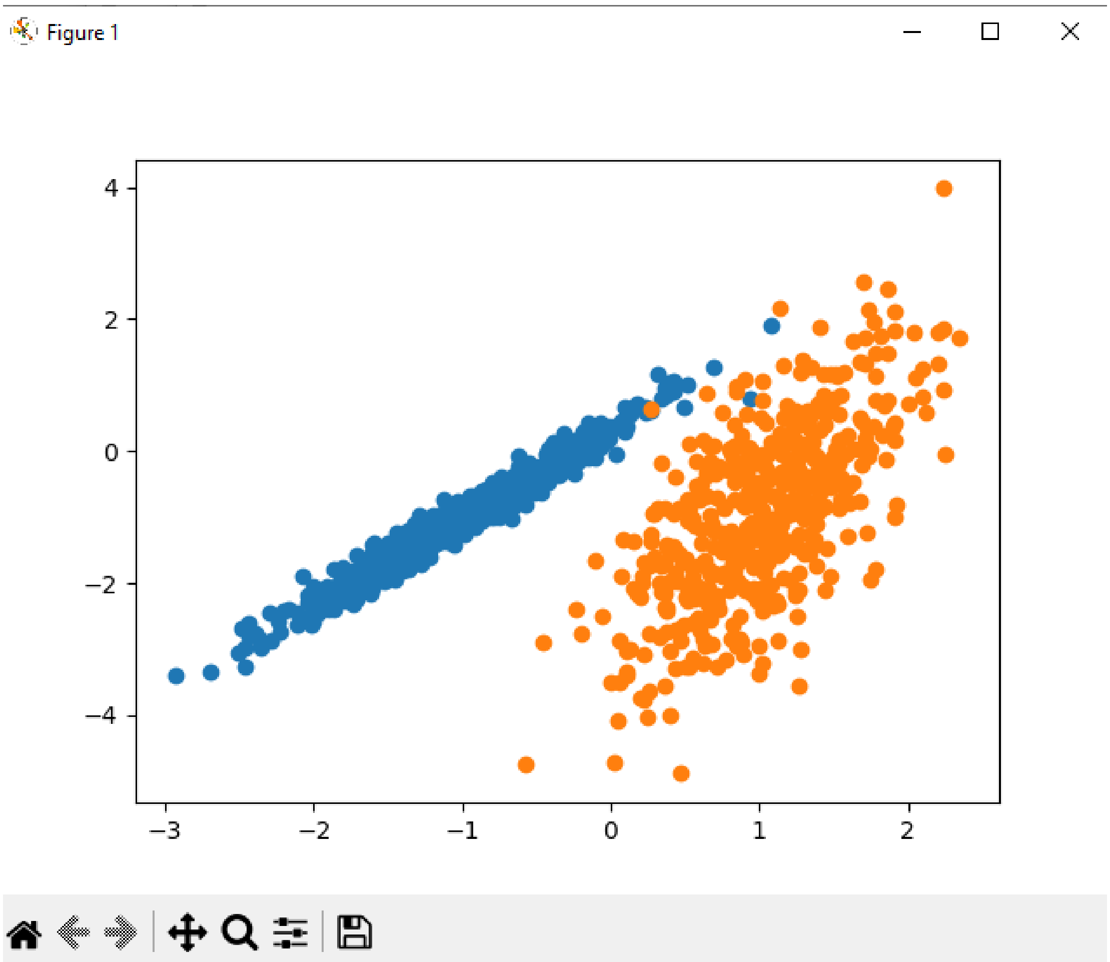
Description:

Code:

main.py

```
from numpy import where
from sklearn.datasets import make_classification
from matplotlib import pyplot
x,y = make_classification(n_samples=1000, n_features=2, n_informative=2, n_redundant=0,
 n_clusters_per_class=1, random_state=4)
for class_value in range(2):
 row_ix = where(y == class_value)
 pyplot.scatter(x[row_ix, 0], x[row_ix, 1])
pyplot.show()
```

Output:



Practical __

Writeup:

Practical __

Aim:

Description:

Code:

```
#Import scikit-learn dataset library
from sklearn import datasets
#Import svm model
from sklearn import svm
# Import train_test_split function
from sklearn.model_selection import train_test_split
#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
#Load dataset
cancer = datasets.load_breast_cancer()
# print the names of the 13 features
print("Features: ", cancer.feature_names)
# print the label type of cancer('malignant' 'benign')
print("Labels: ", cancer.target_names)
# print data(feature)shape
cancer.data.shape
# print the cancer data features (top 5 records)
print(cancer.data[0:5])
# print the cancer labels (0:malignant, 1:benign)
print(cancer.target)
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target,
test_size=0.3,random_state=109) # 70% training and 30% test
#Create a svm Classifier
clf = svm.SVC(kernel='linear') # Linear Kernel
#Train the model using the training sets
clf.fit(X_train, y_train)
#Predict the response for test dataset
```

y_pred = clf.predict(X_test)

Model Accuracy: how often is the classifier correct?

print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

Model Precision: what percentage of positive tuples are labeled as such?

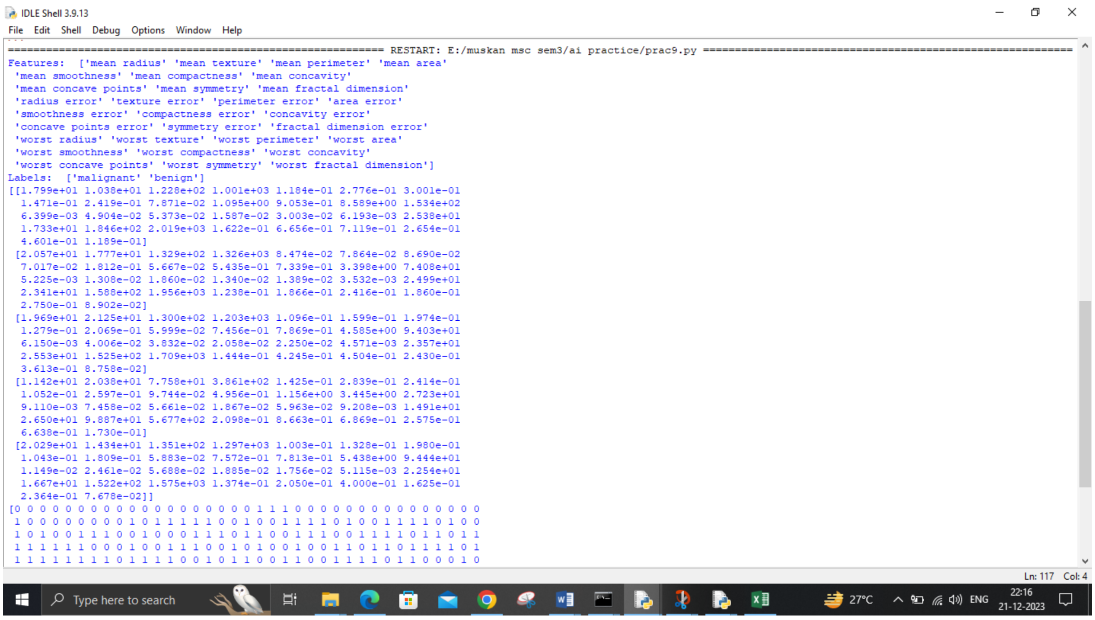
print("Precision:",metrics.precision_score(y_test, y_pred))

Model Recall: what percentage of positive tuples are labelled as such?

print("Recall:",metrics.recall_score(y_test, y_pred))

print("___________done by Muskan Momin_")

Output:



Practical __

Writeup:

Practical __

Aim:

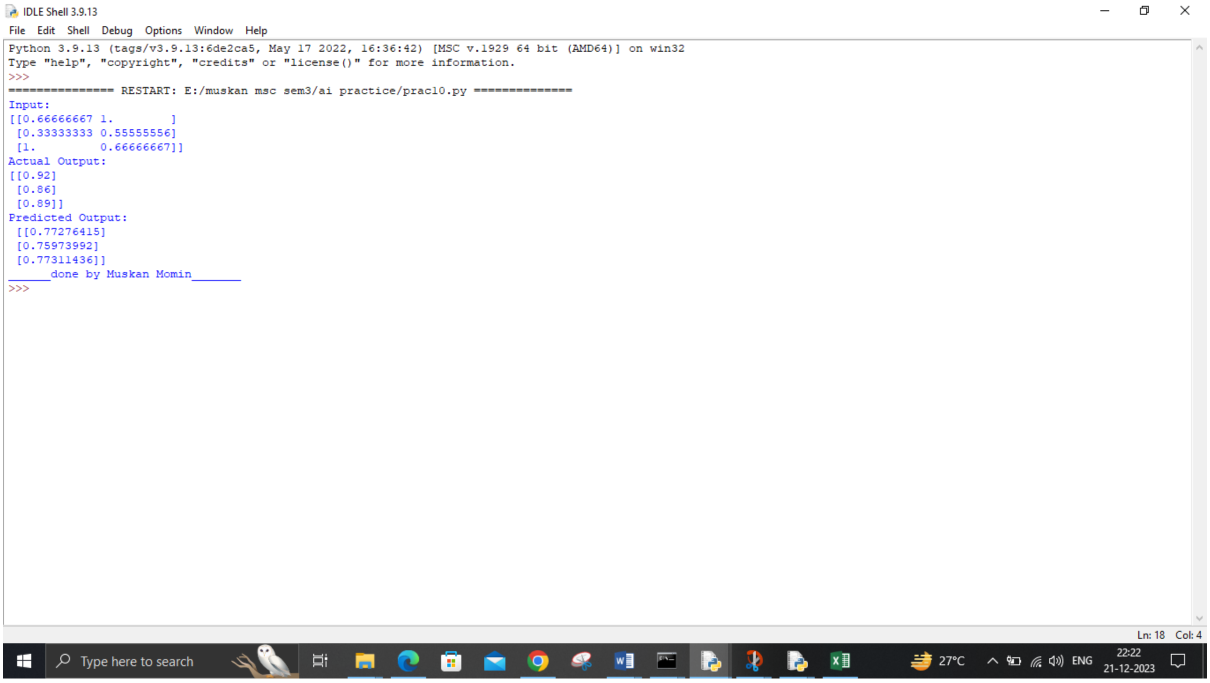
Description:

Code:

```
import numpy as np

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)  # two inputs [sleep,study]

y = np.array(([92], [86], [89]), dtype=float)  # one output [Expected % in Exams]

X = X / np.amax(X, axis=0)  # maximum of X array longitudinally

y = y / 100

# Sigmoid Function

def sigmoid(x):

   return 1 / (1 + np.exp(-x))

# Derivative of Sigmoid Function

def derivatives_sigmoid(x):

   return x * (1 - x)

# Variable initialization

epoch = 5000  # Setting training iterations

lr = 0.1  # Setting learning rate

inputlayer_neurons = 2  # number of features in data set

hiddenlayer_neurons = 3  # number of hidden layers neurons

output_neurons = 1  # number of neurons at output layer

# weight and bias initialization

wh = np.random.uniform(size=(inputlayer_neurons, hiddenlayer_neurons))  # weight of the link from input node to hidden node

bh = np.random.uniform(size=(1, hiddenlayer_neurons))  # bias of the link from input node to hidden node

wout = np.random.uniform(size=(hiddenlayer_neurons, output_neurons))  # weight of the link from hidden node to output node

bout = np.random.uniform(size=(1, output_neurons))  # bias of the link from hidden node to output node

# draws a random range of numbers uniformly of dim x*y

for i in range(epoch):

   # Forward Propogation
```

```
    hinp1 = np.dot(X, wh)
    hinp = hinp1 + bh
    hlayer_act = sigmoid(hinp)
    outinp1 = np.dot(hlayer_act, wout)
    outinp = outinp1 + bout
    output = sigmoid(outinp)
    # Backpropagation
    EO = y - output
    outgrad = derivatives_sigmoid(output)
    d_output = EO * outgrad
    EH = d_output.dot(wout.T)
    # how much hidden layer weights contributed to error
    hiddengrad = derivatives_sigmoid(hlayer_act)
    d_hiddenlayer = EH * hiddengrad
# dotproduct of nextlayererror and currentlayerop
wout += hlayer_act.T.dot(d_output) * lr
wh += X.T.dot(d_hiddenlayer) * lr
print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n", output)
print("______done by Muskan Momin_______")
```

Output:


IDLE Shell 3.9.13
File Edit Shell Debug Options Window Help
Python 3.9.13 (tags/v3.9.13:6de2ca5, May 17 2022, 16:36:42) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=============== RESTART: E:/muskan msc sem3/ai practice/prac10.py ==============
Input:
[[0.66666667 1.   ]
 [0.33333333 0.55555556]
 [1.   0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.77276415]
 [0.75973992]
 [0.77311436]]
______done by Muskan Momin_______
>>>
Ln: 18 Col: 4
Type here to search
27°C
ENG
22:22
21-12-2023

Practical __

Writeup:

Practical __

Aim:

Description:

Code:

```
import random
# Number of individuals in each generation
POPULATION_SIZE = 100
# Valid genes
GENES = '''abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOP
QRSTUVWXYZ 1234567890, .-;:_!"#%&/()=?@${[]}'''
# Target string to be generated
TARGET = "Muskan Momin"
class Individual(object):
	'''
	Class representing individual in population
	'''
	def __init__(self, chromosome):
		self.chromosome = chromosome
		self.fitness = self.cal_fitness()
	@classmethod
	def mutated_genes(self):
		'''
		create random genes for mutation
		'''
		global GENES
		gene = random.choice(GENES)
		return gene
	@classmethod
	def create_gnome(self):
		'''
		create chromosome or string of genes
		'''
```

```
        global TARGET
        gnome_len = len(TARGET)
        return [self.mutated_genes() for _ in range(gnome_len)]
    def mate(self, par2):
        '''
        Perform mating and produce new offspring
        '''
        # chromosome for offspring
        child_chromosome = []
        for gp1, gp2 in zip(self.chromosome, par2.chromosome):
            # random probability
            prob = random.random()
            # if prob is less than 0.45, insert gene
            # from parent 1
            if prob < 0.45:
                child_chromosome.append(gp1)
            # if prob is between 0.45 and 0.90, insert
            # gene from parent 2
            elif prob < 0.90:
                child_chromosome.append(gp2)
            # otherwise insert random gene(mutate),
            # for maintaining diversity
            else:
                child_chromosome.append(self.mutated_genes())
        # create new Individual(offspring) using
        # generated chromosome for offspring
        return Individual(child_chromosome)
    def cal_fitness(self):
        '''
```

```
        Calculate fittness score, it is the number of
        characters in string which differ from target
        string.
        '''
        global TARGET
        fitness = 0
        for gs, gt in zip(self.chromosome, TARGET):
            if gs != gt: fitness+= 1
        return fitness
# Driver code
def main():
    global POPULATION_SIZE
    #current generation
    generation = 1
    found = False
    population = []
    # create initial population
    for _ in range(POPULATION_SIZE):
                gnome = Individual.create_gnome()
                population.append(Individual(gnome))
    while not found:
        # sort the population in increasing order of fitness score
        population = sorted(population, key = lambda x:x.fitness)
        # if the individual having lowest fitness score ie.
        # 0 then we know that we have reached to the target
        # and break the loop
        if population[0].fitness <= 0:
            found = True
            break
```

```
            # Otherwise generate new offsprings for new generation
            new_generation = []
            # Perform Elitism, that mean 10% of fittest population
            # goes to the next generation
            s = int((10*POPULATION_SIZE)/100)
            new_generation.extend(population[:s])
            # From 50% of fittest population, Individuals
            # will mate to produce offspring
            s = int((90*POPULATION_SIZE)/100)
            for _ in range(s):
                parent1 = random.choice(population[:50])
                parent2 = random.choice(population[:50])
                child = parent1.mate(parent2)
                new_generation.append(child)
            population = new_generation
            print("Generation: {}\tString: {}\tFitness: {}".\
                format(generation,
                "".join(population[0].chromosome),
                population[0].fitness))
            generation += 1
        print("Generation: {}\tString: {}\tFitness: {}".\
            format(generation,
            "".join(population[0].chromosome),
            population[0].fitness))
if __name__ == '__main__':
    main()
```

Output:


IDLE Shell 3.9.13
File Edit Shell Debug Options Window Help
Python 3.9.13 (tags/v3.9.13:6de2ca5, May 17 2022, 16:36:42) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=============== RESTART: E:\muskan msc sem3\ai practice\prac11.py ==============
Generation: 1 String: Qf_ZBR 8/sQJ Fitness: 11
Generation: 2 String: on ka$xuF fk Fitness: 10
Generation: 3 String: on ka$xuF fk Fitness: 10
Generation: 4 String: o5Zkau uF Gg Fitness: 9
Generation: 5 String: ;rsfa] Bo i# Fitness: 7
Generation: 6 String: ;rskan BjZi# Fitness: 6
Generation: 7 String: ;rskan BjZi# Fitness: 6
Generation: 8 String: ;rskan BjZi# Fitness: 6
Generation: 9 String: ;rskan BjZi# Fitness: 6
Generation: 10 String: ERskan BoZiW Fitness: 5
Generation: 11 String: ERskan BoZiW Fitness: 5
Generation: 12 String: Vu_kan Bomi[Fitness: 4
Generation: 13 String: Vu_kan Bomi[Fitness: 4
Generation: 14 String: Vu_kan Bomi[Fitness: 4
Generation: 15 String: Vu_kan Bomi[Fitness: 4
Generation: 16 String: Vuskan ;omi[Fitness: 3
Generation: 17 String: Vuskan ;omi[Fitness: 3
Generation: 18 String: ouskan Momiu Fitness: 2
Generation: 19 String: ouskan Momiu Fitness: 2
Generation: 20 String: ouskan Momiu Fitness: 2
Generation: 21 String: ouskan Momiu Fitness: 2
Generation: 22 String: ouskan Momiu Fitness: 2
Generation: 23 String: ouskan Momiu Fitness: 2
Generation: 24 String: ouskan Momiu Fitness: 2
Generation: 25 String: ouskan Momiu Fitness: 2
Generation: 26 String: ouskan Momiu Fitness: 2
Generation: 27 String: ouskan Momiu Fitness: 2
Generation: 28 String: Muskan Momif Fitness: 1
Generation: 29 String: Muskan Momif Fitness: 1
Generation: 30 String: Muskan Momif Fitness: 1
Generation: 31 String: Muskan Momif Fitness: 1
Generation: 32 String: Muskan Momif Fitness: 1
Generation: 33 String: Muskan Momif Fitness: 1
Generation: 34 String: Muskan Momin Fitness: 0
>>>
Ln: 21 Col: 46
Type here to search
27°C
ENG
22:27
21-12-2023

Practical __

Writeup:

Practical __

Aim:

Description:

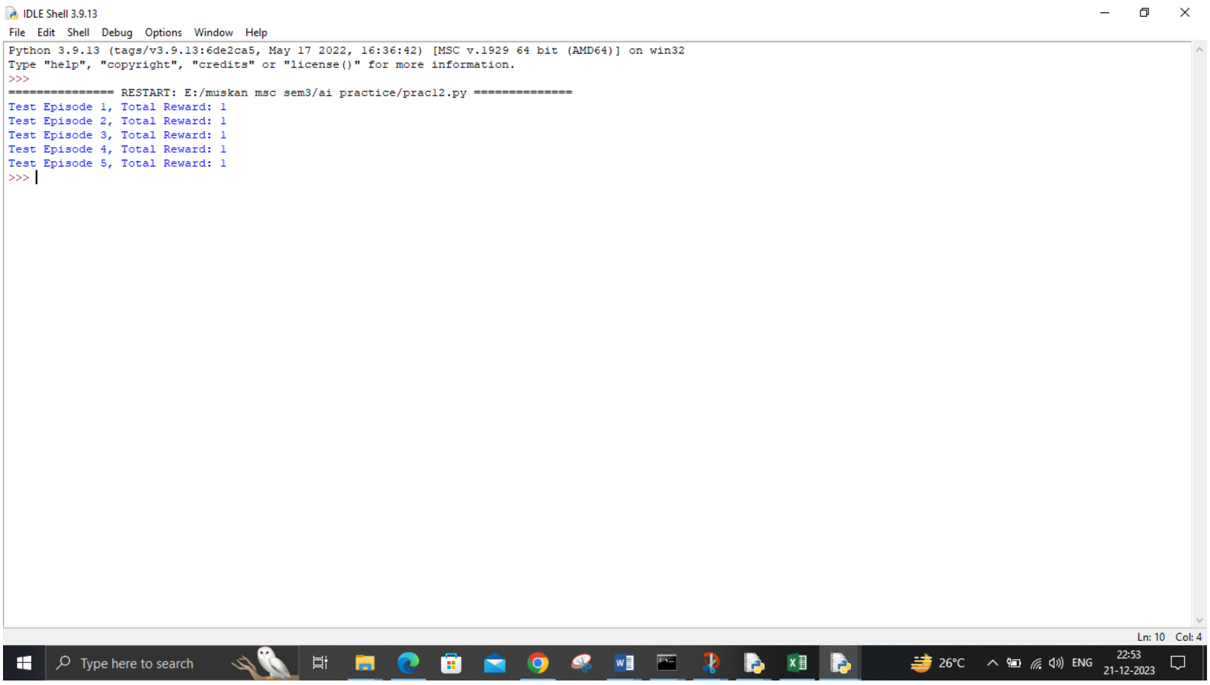
Code:

```
import numpy as np

class QLearningAgent:

    def __init__(self, num_states, num_actions, learning_rate=0.1, discount_factor=0.9,
exploration_rate=0.1):

        # Initialize Q-table with zeros

        self.q_table = np.zeros((num_states, num_actions))

        self.learning_rate = learning_rate

        self.discount_factor = discount_factor

        self.exploration_rate = exploration_rate

    def select_action(self, state):

        # Choose the action with the highest Q-value, with exploration

        return np.argmax(self.q_table[state, :]) if np.random.rand() > self.exploration_rate else
np.random.choice(len(self.q_table[state, :]))

    def update_q_table(self, state, action, reward, next_state):

        # Update Q-value using the Q-learning update rule

        best_next_action = np.argmax(self.q_table[next_state, :])

        self.q_table[state, action] += self.learning_rate * (reward + self.discount_factor *
self.q_table[next_state, best_next_action] - self.q_table[state, action])

class ParkingEnvironment:

    def __init__(self):

        # Initialize parking environment with 3 states and 2 actions

        self.num_states = 3

        self.num_actions = 2

        self.goal_state = 2

        self.agent_state = 0

        self.done = False

    def reset(self):

        # Reset the environment for a new episode

        self.agent_state = 0
```

```
        self.done = False
    def step(self, action):
        if self.done:
            return self.agent_state, 0, self.done
        # Update agent's position based on the action
        if action == 0:  # Move left
            self.agent_state = max(0, self.agent_state - 1)
        else:  # Move right
            self.agent_state = min(self.num_states - 1, self.agent_state + 1)
        # Provide reward based on the agent reaching the goal state
        reward = 1 if self.agent_state == self.goal_state else 0
        self.done = (self.agent_state == self.goal_state)
        return self.agent_state, reward, self.done
# Main loop
num_states = 3
num_actions = 2
num_episodes = 100
# Create Q-learning agent and parking environment
agent = QLearningAgent(num_states, num_actions)
environment = ParkingEnvironment()
# Training loop
for episode in range(num_episodes):
    environment.reset()
    # Episode loop
    while not environment.done:
        state = environment.agent_state
        action = agent.select_action(state)
        next_state, reward, done = environment.step(action)
        agent.update_q_table(state, action, reward, next_state)
```

```
# Test the trained agent
test_episodes = 5
for episode in range(test_episodes):
    environment.reset()
    # Episode loop for testing
    while not environment.done:
        state = environment.agent_state
        action = agent.select_action(state)
        next_state, reward, done = environment.step(action)
    print(f"Test Episode {episode + 1}, Total Reward: {reward}")
```

Output:



Practical __

Writeup:

Practical __

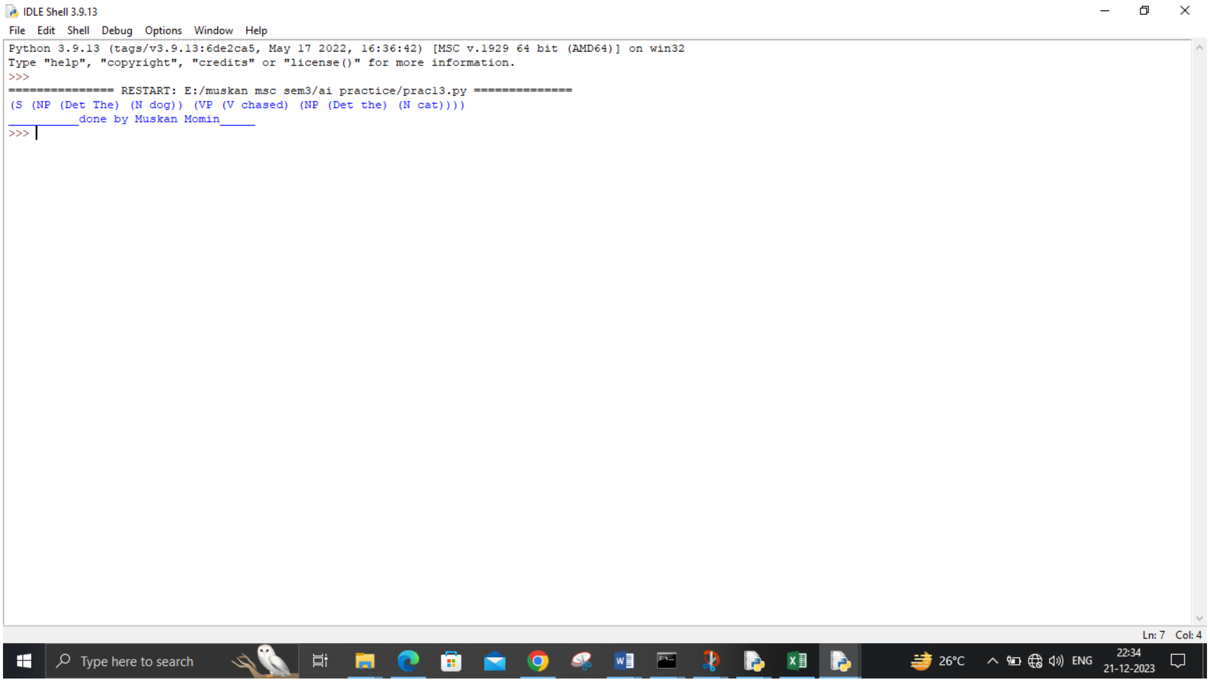
Aim:

Description:

Code:

```
# Importing the required libraries
import nltk
from nltk import CFG
# Defining the grammar rules
grammar = CFG.fromstring("""
    S -> NP VP
    NP -> Det N | Det N PP
    VP -> V NP | V NP PP
    PP -> P NP
    Det -> 'The' | 'a' | 'the'
    N -> 'dog' | 'cat' | 'house' | 'car'
    V -> 'chased' | 'ate' | 'drove'
    P -> 'in' | 'on' | 'at'
""")
# Creating the parser
parser = nltk.ChartParser(grammar)
# Parsing a sentence
sentence = "The dog chased the cat"
for tree in parser.parse(sentence.split()):print(tree)
print("__________done by Muskan Momin_____")
```

Output:


IDLE Shell 3.9.13
File Edit Shell Debug Options Window Help
Python 3.9.13 (tags/v3.9.13:6de2ca5, May 17 2022, 16:36:42) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=============== RESTART: E:/muskan msc sem3/ai practice/prac13.py ==============
(S (NP (Det The) (N dog)) (VP (V chased) (NP (Det the) (N cat))))
__________done by Muskan Momin_____
>>>
Ln: 7 Col: 4
Type here to search
26°C
ENG
22:34
21-12-2023