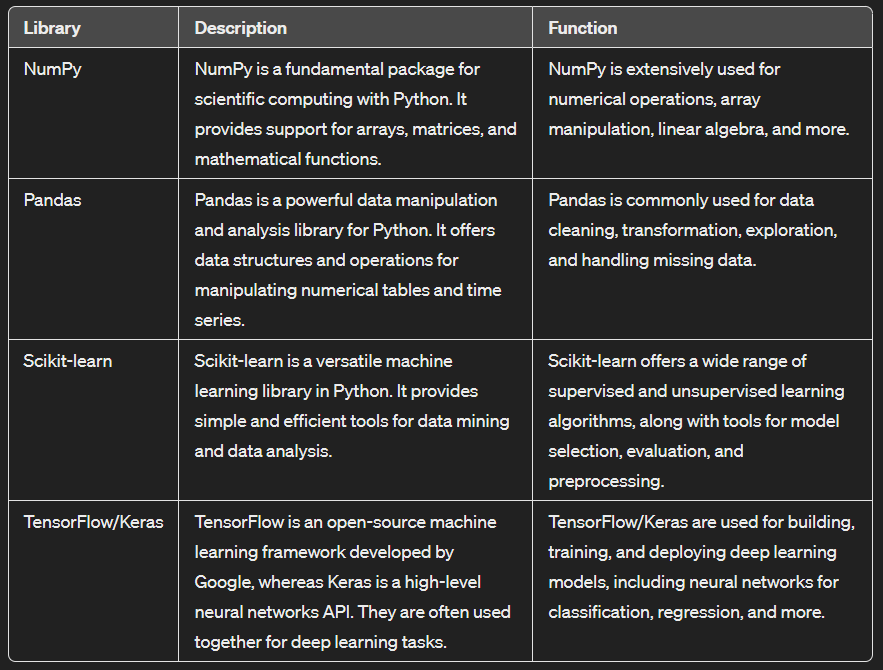
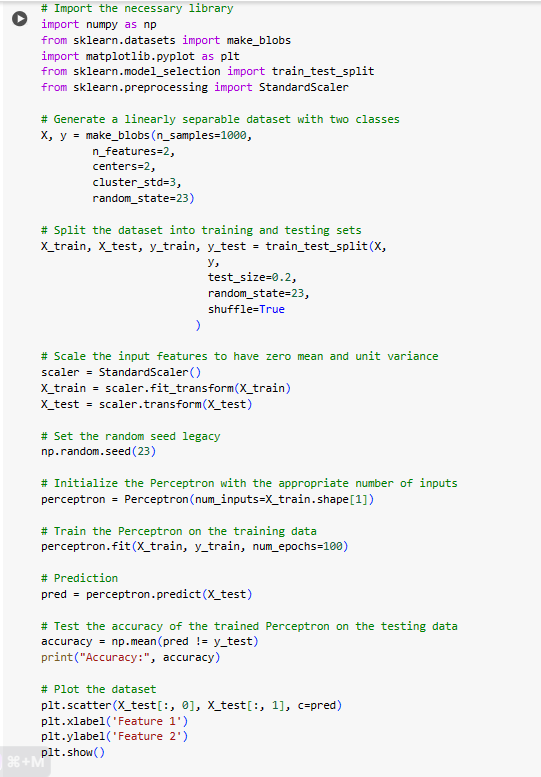
1. Study/explore the library of python (3-4 with description)

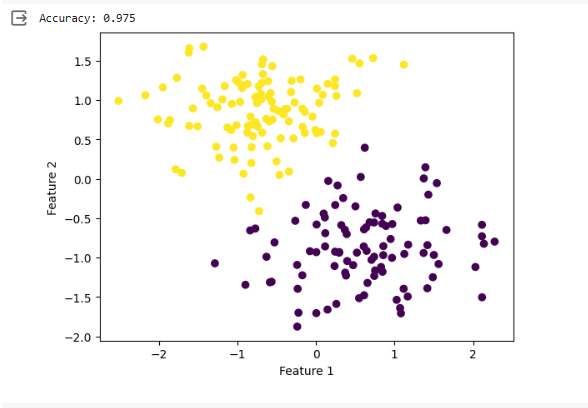


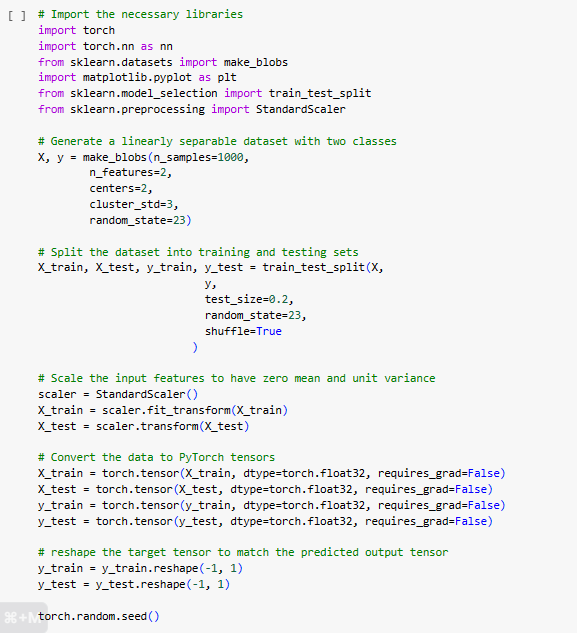
1. Perceptron implementation.

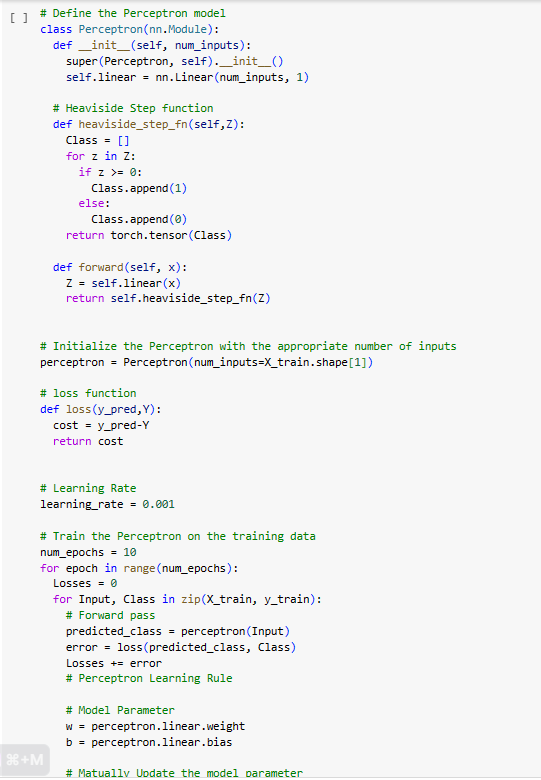
Code:

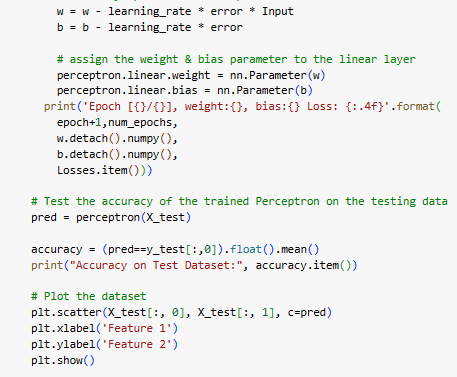


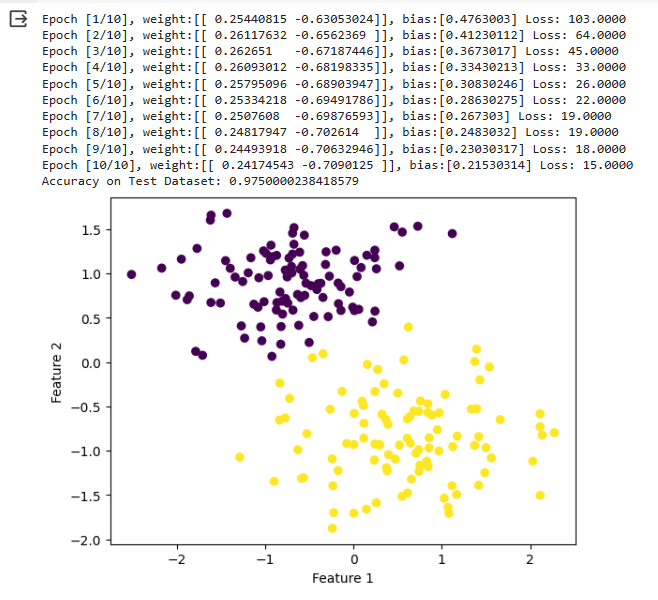












1. Implementation of id3.

Code:

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

import math

import copy

dataset = pd.read\_csv('tennis.csv')

X = dataset.iloc[:, 1:].values

# print(X)

attribute = ['outlook', 'temp', 'humidity', 'wind']

class Node(object):

def \_\_init\_\_(self):

self.value = None

self.decision = None

self.childs = None

def findEntropy(data, rows):

yes = 0

no = 0

ans = -1

idx = len(data[0]) - 1

entropy = 0

for i in rows:

if data[i][idx] == 'Yes':

yes = yes + 1

else:

no = no + 1

x = yes/(yes+no)

y = no/(yes+no)

if x != 0 and y != 0:

entropy = -1 \* (x\*math.log2(x) + y\*math.log2(y))

if x == 1:

ans = 1

if y == 1:

ans = 0

return entropy, ans

def findMaxGain(data, rows, columns):

maxGain = 0

retidx = -1

entropy, ans = findEntropy(data, rows)

if entropy == 0:

"""if ans == 1:

print("Yes")

else:

print("No")"""

return maxGain, retidx, ans

for j in columns:

mydict = {}

idx = j

for i in rows:

key = data[i][idx]

if key not in mydict:

mydict[key] = 1

else:

mydict[key] = mydict[key] + 1

gain = entropy

# print(mydict)

for key in mydict:

yes = 0

no = 0

for k in rows:

if data[k][j] == key:

if data[k][-1] == 'Yes':

yes = yes + 1

else:

no = no + 1

# print(yes, no)

x = yes/(yes+no)

y = no/(yes+no)

# print(x, y)

if x != 0 and y != 0:

gain += (mydict[key] \* (x\*math.log2(x) + y\*math.log2(y)))/14

# print(gain)

if gain > maxGain:

# print("hello")

maxGain = gain

retidx = j

return maxGain, retidx, ans

def buildTree(data, rows, columns):

maxGain, idx, ans = findMaxGain(X, rows, columns)

root = Node()

root.childs = []

# print(maxGain

#

# )

if maxGain == 0:

if ans == 1:

root.value = 'Yes'

else:

root.value = 'No'

return root

root.value = attribute[idx]

mydict = {}

for i in rows:

key = data[i][idx]

if key not in mydict:

mydict[key] = 1

else:

mydict[key] += 1

newcolumns = copy.deepcopy(columns)

newcolumns.remove(idx)

for key in mydict:

newrows = []

for i in rows:

if data[i][idx] == key:

newrows.append(i)

# print(newrows)

temp = buildTree(data, newrows, newcolumns)

temp.decision = key

root.childs.append(temp)

return root

def traverse(root):

print(root.decision)

print(root.value)

n = len(root.childs)

if n > 0:

for i in range(0, n):

traverse(root.childs[i])

def calculate():

rows = [i for i in range(0, 14)]

columns = [i for i in range(0, 4)]

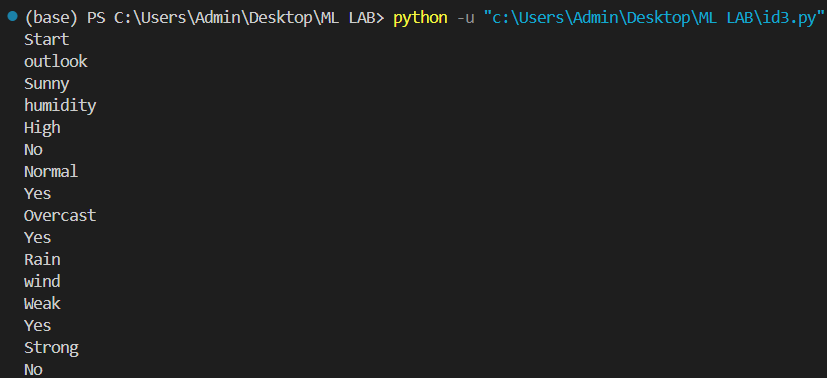
root = buildTree(X, rows, columns)

root.decision = 'Start'

traverse(root)

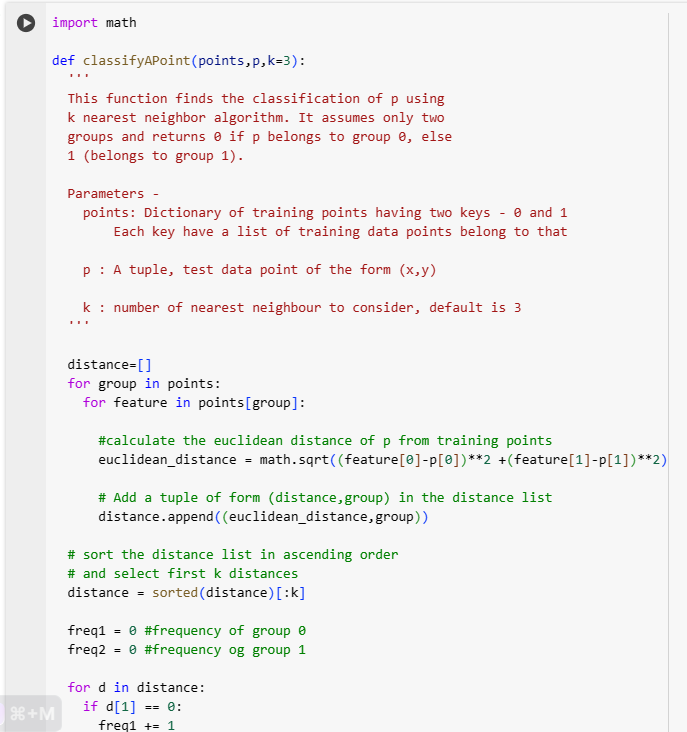
calculate()

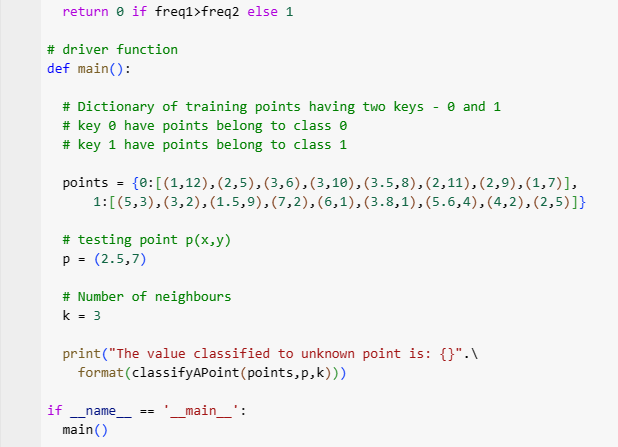
Output:



1. Implementing K- nearest neighbour.

Code:

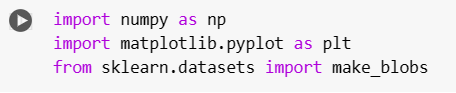




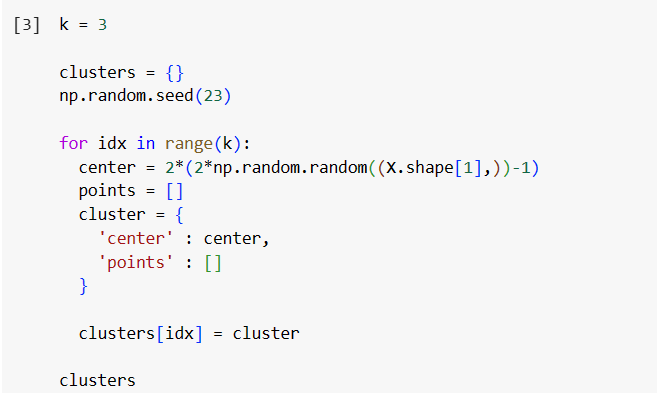
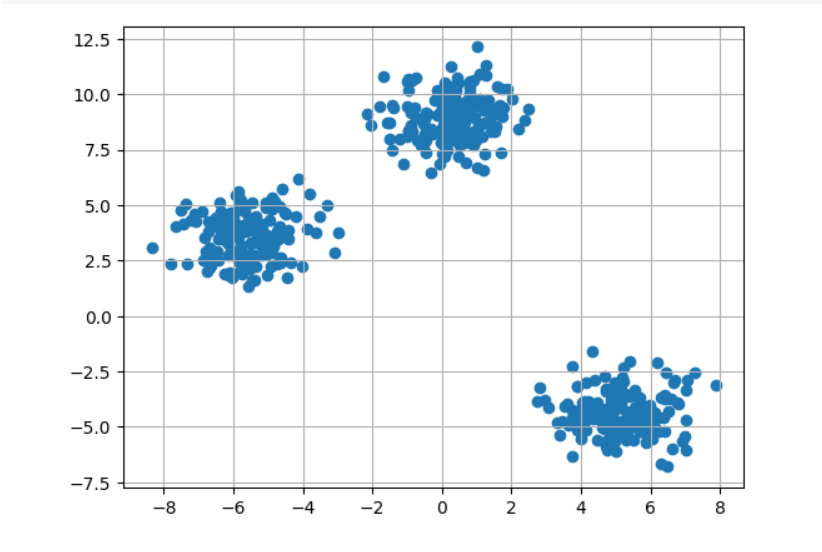


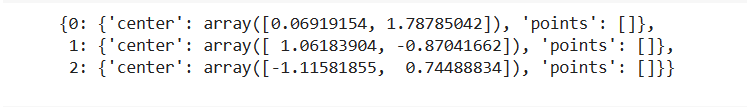
1. Implementing K- means clustering.

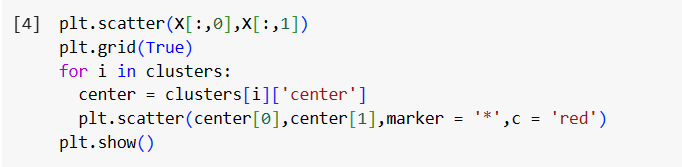
Code:

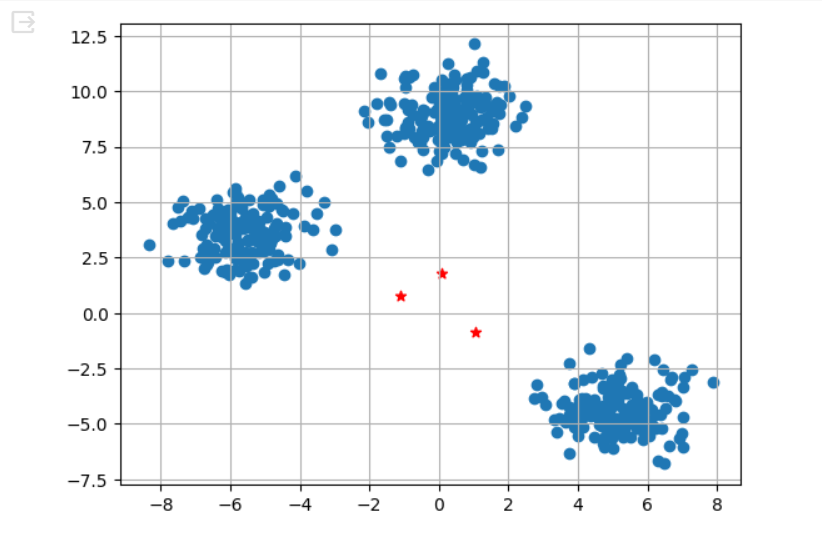


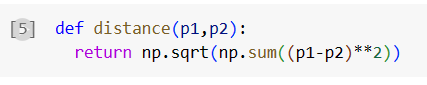




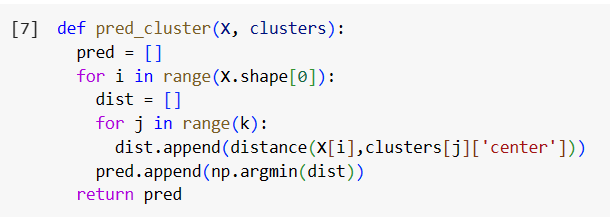


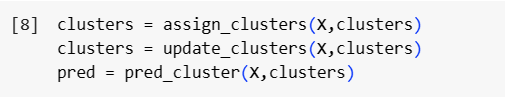


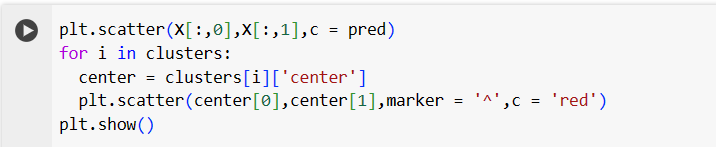


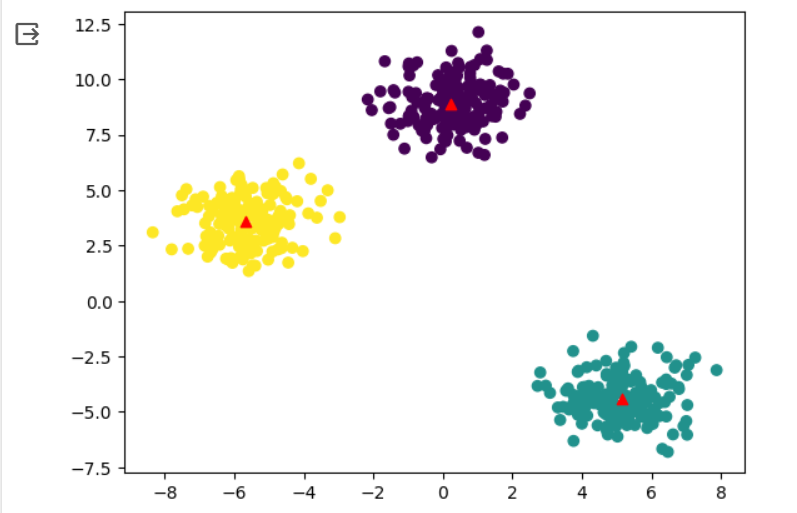






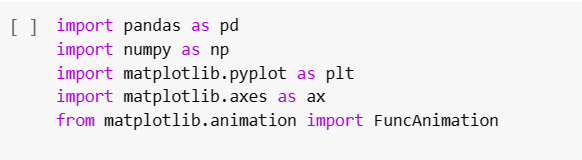


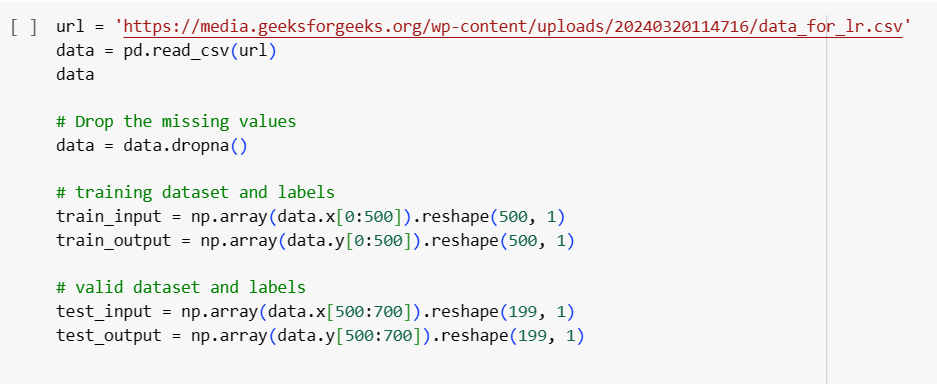


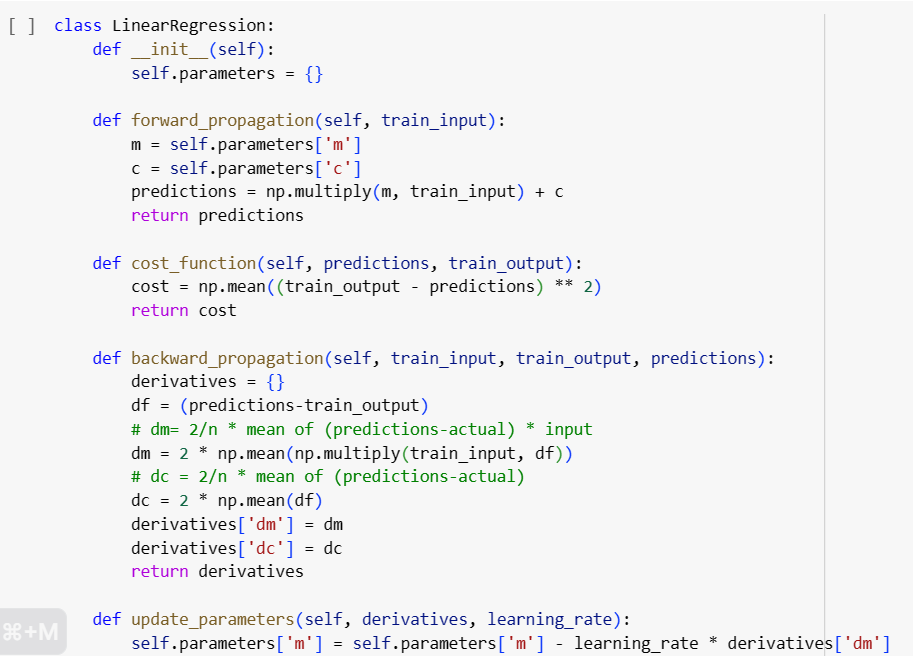


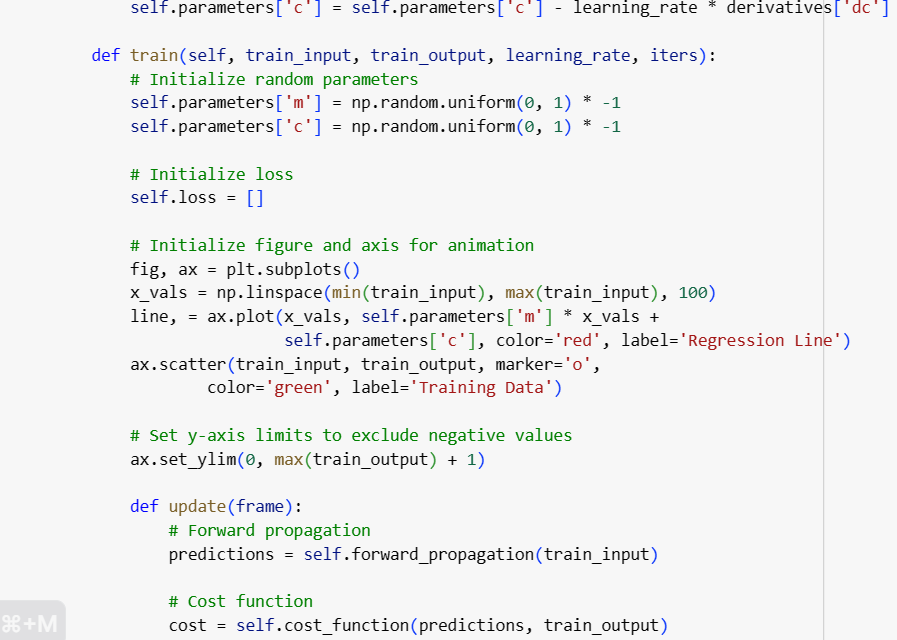
1. Implementing regression
2. Linear

Code:

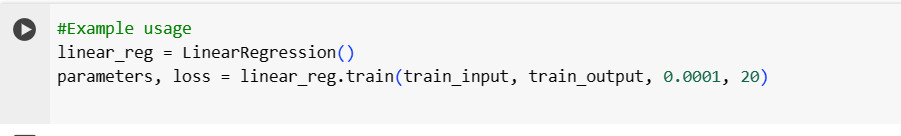


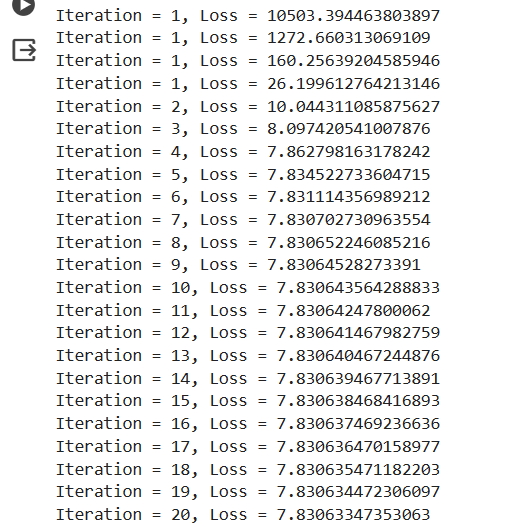


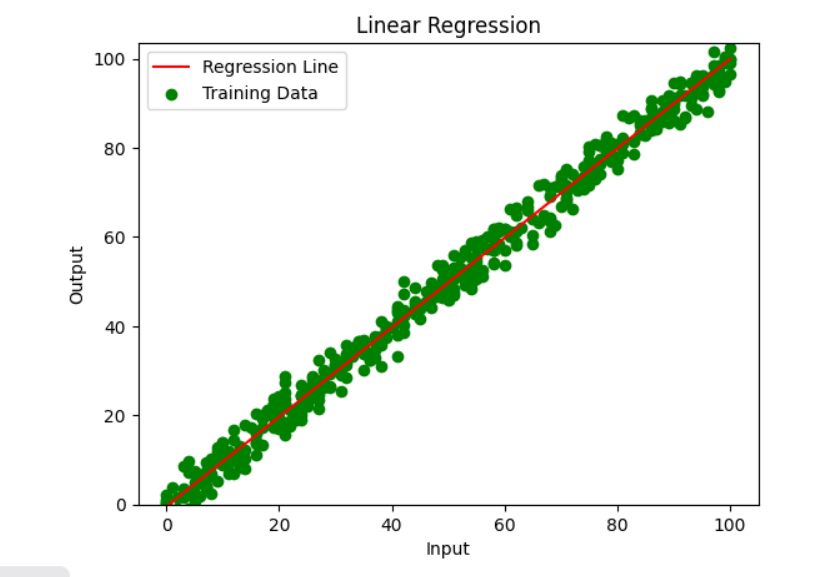












1. Logistic

Code:

# Importing required libraries

import numpy as np

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

from sklearn.linear\_model import LogisticRegression

from sklearn.datasets import make\_classification

from sklearn.metrics import accuracy\_score

# Generating some example data

X, y = make\_classification(n\_samples=1000, n\_features=10, n\_classes=2, random\_state=42)

# Splitting data 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)

# Creating and training the logistic regression model

logreg = LogisticRegression()

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

# Making predictions on the testing set

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

# Calculating the accuracy of the model

accuracy = accuracy\_score(y\_test, y\_pred)

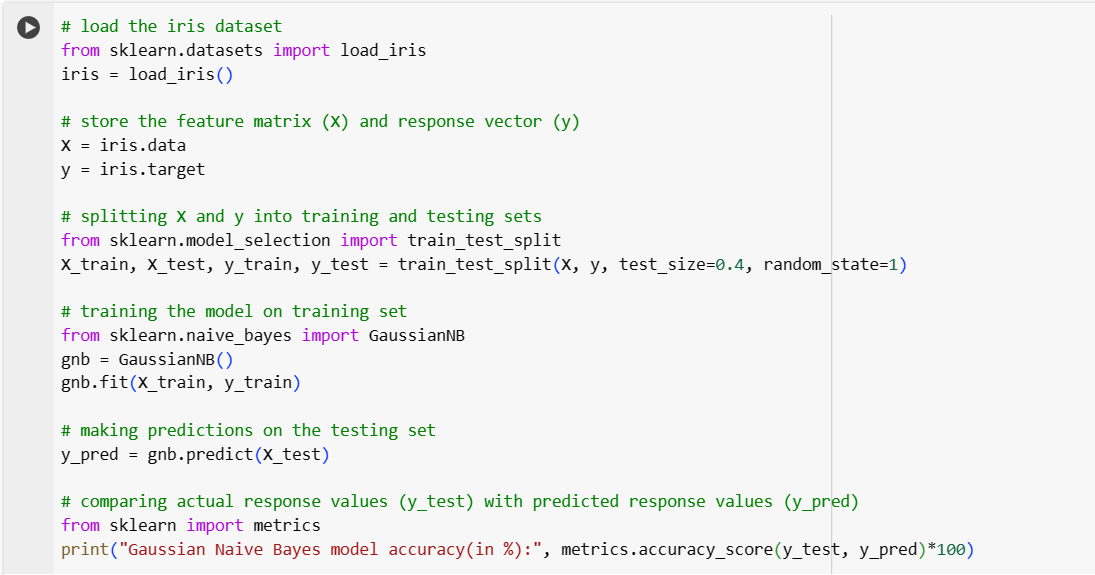
print("Accuracy:", accuracy)

Output:



1. Implementing Naive Bayes.

Code:

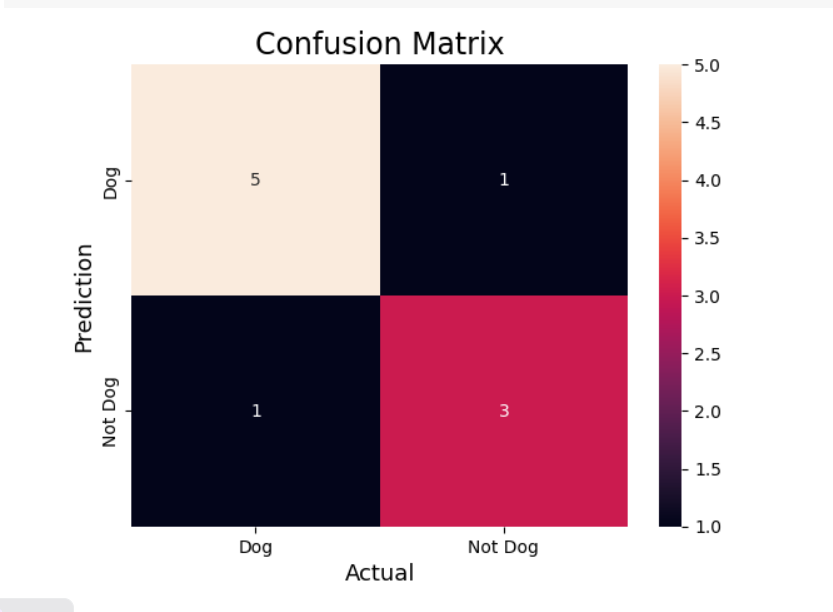


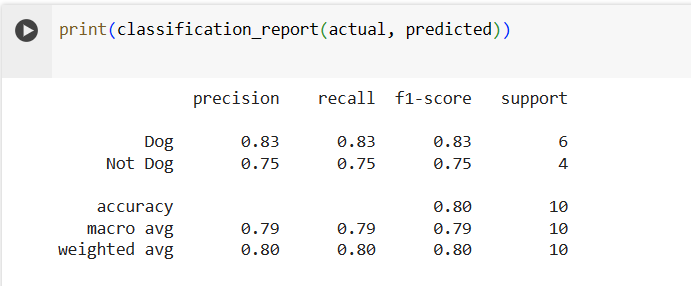


1. Implementing Confusion matrix.

Code:

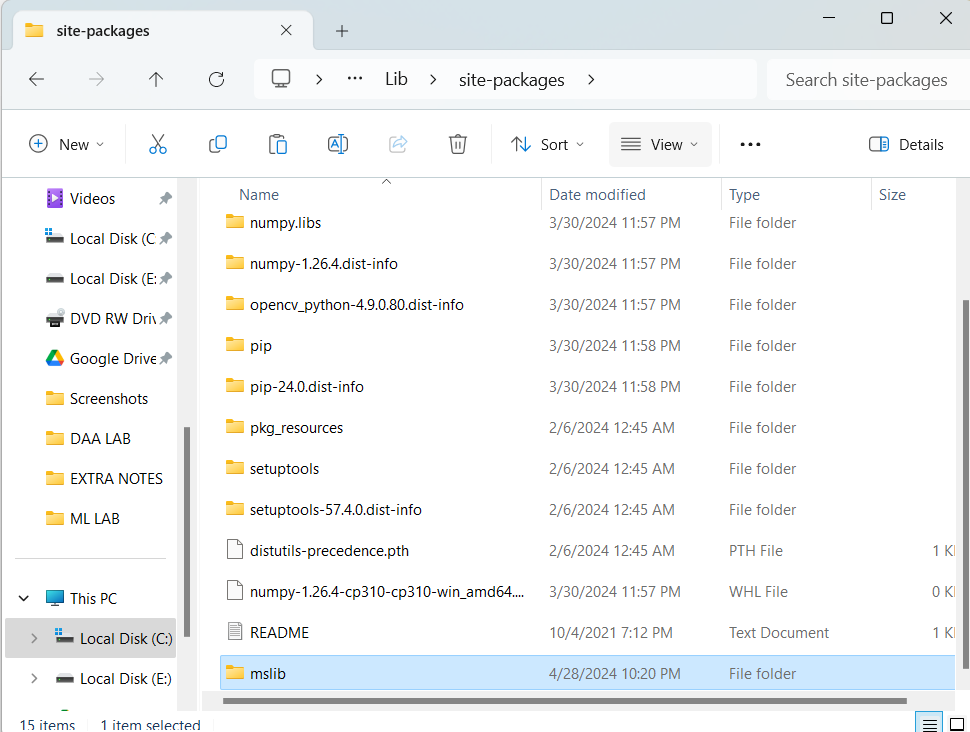


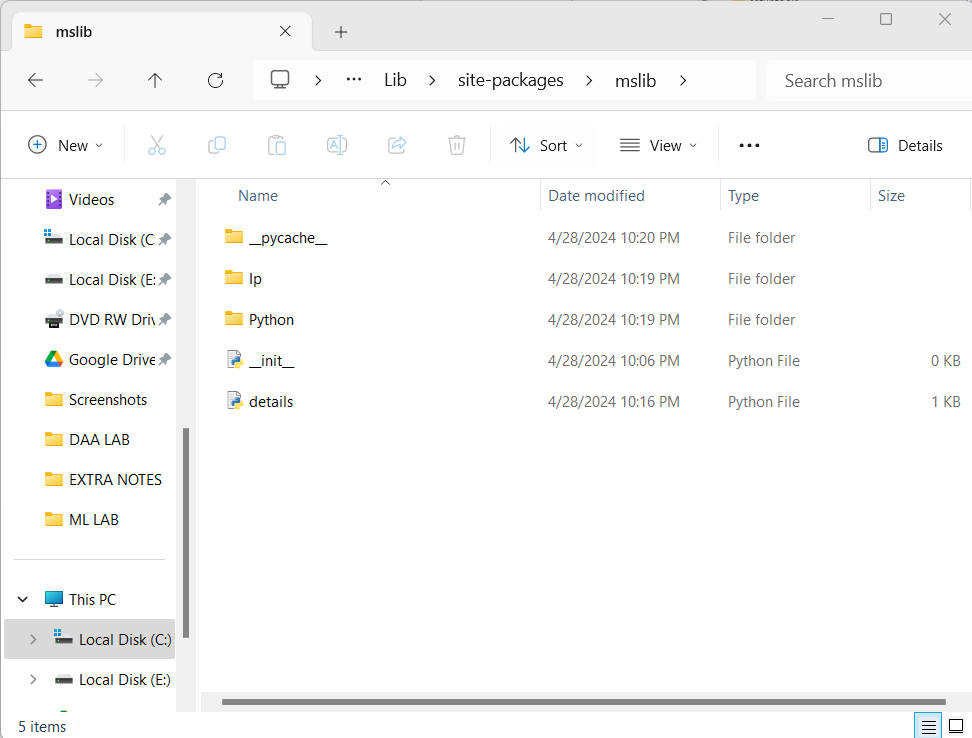


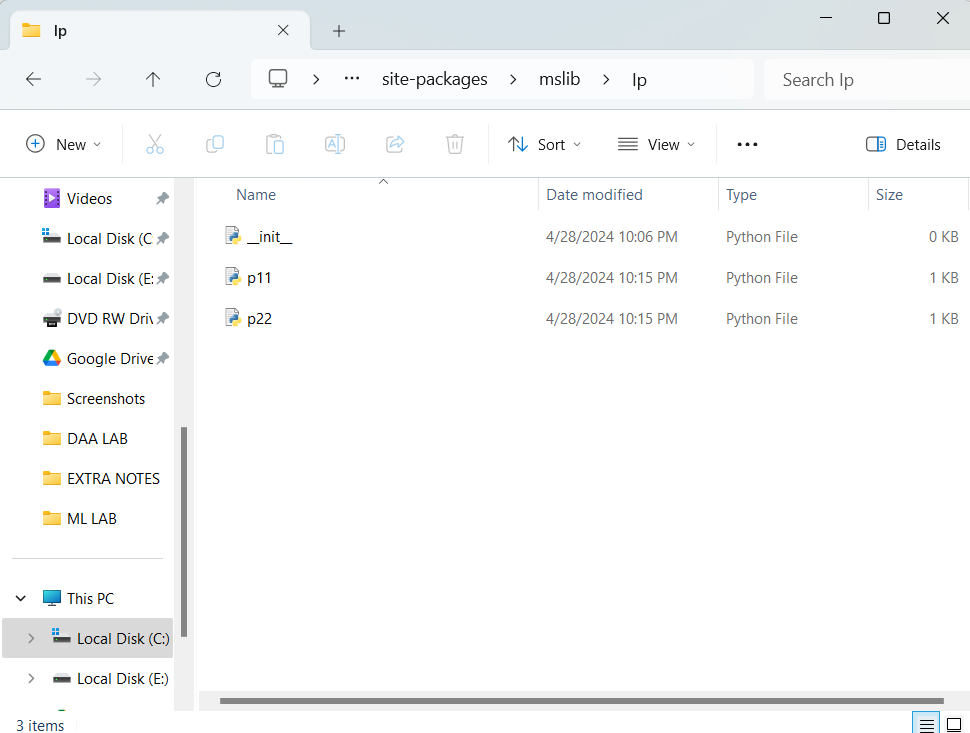


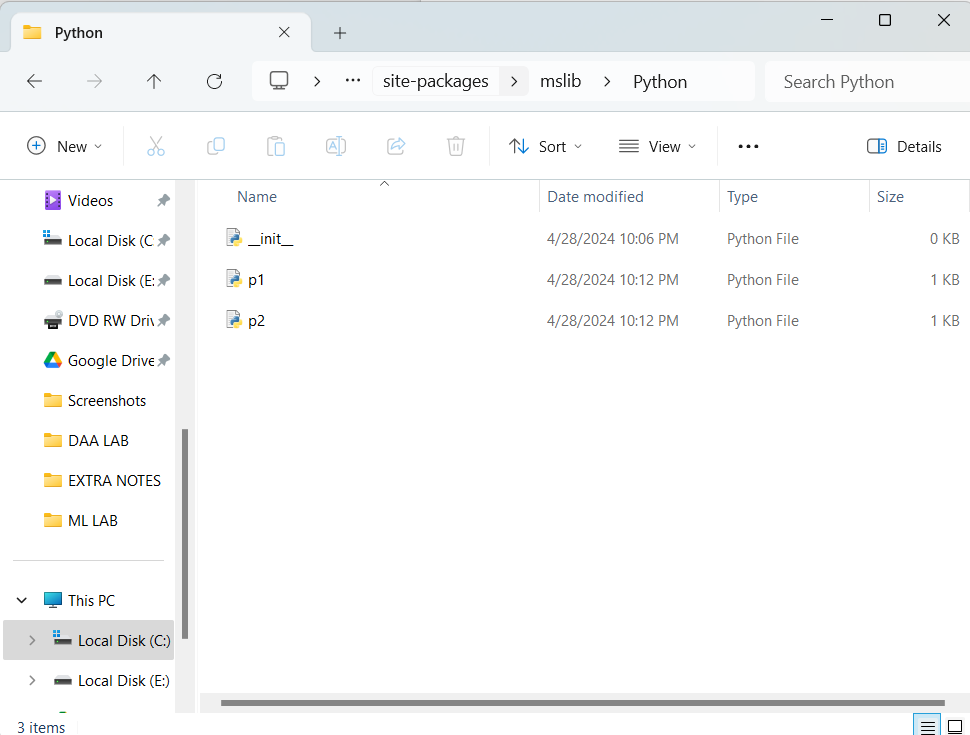
1. Implementing custom library

Code:









1. Implementing ML PROJECT

PS: Credit Card Fraud Detection Using ML

* Credit card fraud is a significant concern for both credit card companies and cardholders. Fraudulent transactions can lead to financial losses for cardholders and card issuers, as well as damage to the reputation of the financial institution. Machine learning techniques offer a promising approach to detect fraudulent transactions automatically, helping to mitigate these risks.
* Logistic regression is a powerful and interpretable algorithm that can be effectively used for credit card fraud detection by leveraging historical transaction data to predict the likelihood of fraudulent activity. Its simplicity, efficiency, and probabilistic nature make it a valuable tool in the arsenal of fraud detection algorithms used by financial institutions and credit card companies.
* Logistic regression is a statistical method used for binary classification tasks, where the target variable has only two possible outcomes. Despite its name, logistic regression is a classification algorithm rather than a regression algorithm. It's widely used in various fields, including finance, healthcare, marketing, and more.

ML model link:

[mlproject.ipynb - Colab (google.com)](https://colab.research.google.com/drive/17fcJv5g-aWaEnulMnFurC-y4jJi0C6VE?authuser=3)

INDEX

|  |  |  |  |
| --- | --- | --- | --- |
| S.NO | EXPERIMENT | SUBMISSION DATE | SIGN |
| 1. | Study/explore the library of python (3-4 with description) |  |  |
| 2. | Implementing perceptron |  |  |
| 3. | Implementing ID3 |  |  |
| 4. | Implementing KNN |  |  |
| 5. | Implementing K-MEANS |  |  |
| 6. | Implementing REGRESSION(LINEAR & LOGISTIC) |  |  |
| 7. | Implementing NAIVE BAYES |  |  |
| 8. | Implementing CONFUSION MATRIX |  |  |
| 9. | Implementing CUSTOM LIBRARY |  |  |
| 10. | Implementing ML PROJECT |  |  |