Abhineet Raman(2002078)

B.Sc.(H.) Computer Science 3rd Year

 Semester: 6

Machine Learning Practical File

1. Perform elementary mathematical operations in Octave/MATLAB like addition, multiplication, division and exponentiation.

Source Code: -

# Addition

a = 5

b = 10

sum = a + b

print("Sum:", sum)

# Subtraction

c = 7

d = 3

difference = c - d

print("Difference:", difference)

# Multiplication

e = 2

f = 6

product = e \* f

print("Product:", product)

# Division

g = 15

h = 3

quotient = g / h

print("Quotient:", quotient)

# Exponentiation

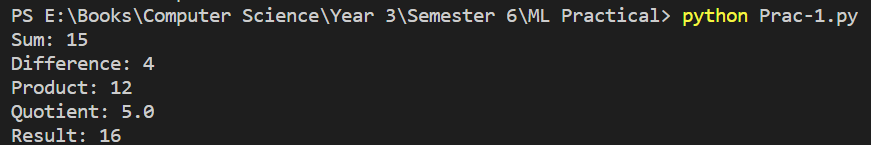
i = 2

j = 4

result = i \*\* j

print("Result:", result)

Output: -



2. Perform elementary logical operations in Octave/MATLAB (like OR, AND, Checking for Equality, NOT, XOR).

Source Code: -

# OR

a = True

b = False

result\_or = a or b

print("OR:", result\_or)

# AND

c = True

d = False

result\_and = c and d

print("AND:", result\_and)

# Checking for Equality

e = 5

f = 10

result\_equal = e == f

print("Equal:", result\_equal)

# NOT

g = True

result\_not = not g

print("NOT:", result\_not)

# XOR

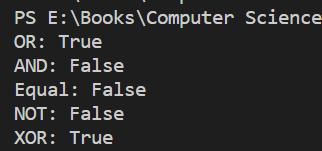
h = True

i = False

result\_xor = h != i

print("XOR:", result\_xor)

Output: -



3. Create, initialize and display simple variables and simple strings and use simple formatting for variable.

Source Code: -

# Create and initialize simple variables

x = 10

y = 3.14

z = True

# Display the variables

print("x =", x)

print("y =", y)

print("z =", z)

# Create and initialize simple strings

name = "Alice"

greeting = "Hello, " + name + "!"

# Display the strings

print(greeting)

print("The length of the name is", len(name))

# Use simple formatting for variables

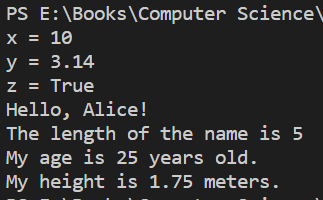
age = 25

print("My age is {} years old.".format(age))

height = 1.75

print(f"My height is {height:.2f} meters.")

Output: -



4. Create/Define single dimension / multi-dimension arrays, and arrays with specific values like array of all ones, all zeros, array with random values within a range, or a diagonal matrix.

Source Code: -

import numpy as np

# Define a single dimension array

arr1 = np.array([1, 2, 3, 4, 5])

# Display the array

print(arr1)

# Define a 2D array

arr2 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

# Display the array

print(arr2)

# Define a single dimension array of all ones

arr\_ones = np.ones(5)

# Display the array

print(arr\_ones)

# Define a 2D array of all zeros

arr\_zeros = np.zeros((3, 4))

# Display the array

print(arr\_zeros)

# Define a single dimension array with random values between 0 and 1

arr\_random = np.random.rand(5)

# Display the array

print(arr\_random)

# Define a 2D array with random values between -1 and 1

arr\_random2 = np.random.uniform(low=-1, high=1, size=(3, 4))

# Display the array

print(arr\_random2)

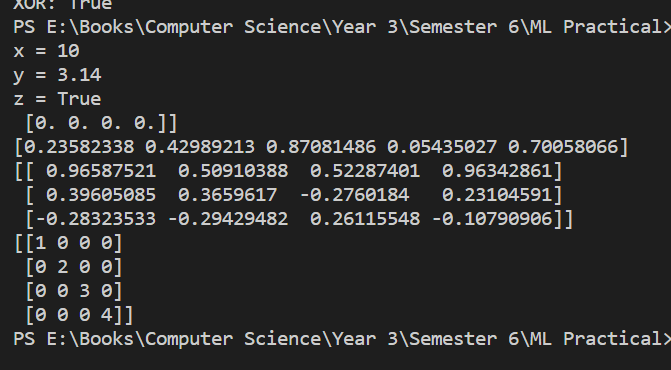
# Define a diagonal matrix

arr\_diag = np.diag([1, 2, 3, 4])

# Display the matrix

print(arr\_diag)

Output: -



5. Use command to compute the size of a matrix, size/length of a particular row/column, load data from a text file, store matrix data to a text file, finding out variables and their features in the current scope.

Source Code: -

import numpy as np

# Define a 3x4 matrix

mat = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])

# Compute the size of the matrix

mat\_size = mat.shape

# Display the size of the matrix

print(mat\_size)

# Compute the length of the first row

row\_len = len(mat[0])

# Compute the length of the second column

col\_len = len(mat[:, 1])

# Display the length of the row and column

print(row\_len, col\_len)

# Load data from a text file

data = np.loadtxt('data.txt')

# Display the loaded data

print(data)

# Save the matrix to a text file

np.savetxt('mat.txt', mat)

# Load the matrix from the text file

loaded\_mat = np.loadtxt('mat.txt')

# Display the loaded matrix

print(loaded\_mat)

def current\_scope():

    var1 = 10

    var2 = 'hello'

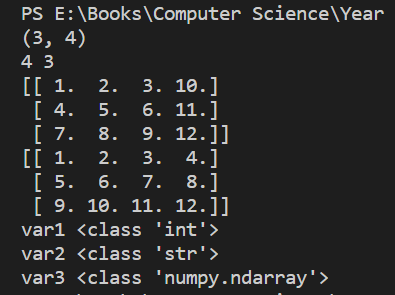
    var3 = np.array([1, 2, 3, 4])

    for name, val in locals().items():

        print(name, type(val))

current\_scope()

Output: -



6. Perform basic operations on matrices (like addition, subtraction, multiplication) and display specific rows or columns of the matrix.

Source Code: -

import numpy as np

# Define two matrices

mat1 = np.array([[1, 2], [3, 4]])

mat2 = np.array([[5, 6], [7, 8]])

# Add the matrices

mat\_sum = mat1 + mat2

# Display the result

print(mat\_sum)

# Subtract the matrices

mat\_diff = mat1 - mat2

# Display the result

print(mat\_diff)

# Multiply the matrices

mat\_prod = np.dot(mat1, mat2)

# Display the result

print(mat\_prod)

# Define a matrix

mat = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

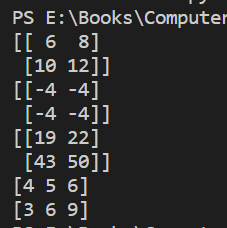
# Display the second row

print(mat[1, :])

# Display the third column

print(mat[:, 2])

Output: -



7. Perform other matrix operations like converting matrix data to absolute values, taking the negative of matrix values, adding/removing rows/columns from a matrix, finding the maximum or minimum values in a matrix or in a row/column, and finding the sum of some/all elements in a matrix.

Source Code: -

import numpy as np

# Define a matrix

mat = np.array([[-1, 2, -3], [4, -5, 6], [-7, 8, -9]])

# Convert matrix data to absolute values

mat\_abs = np.abs(mat)

# Display the result

print(mat\_abs)

# Take the negative of matrix values

mat\_neg = -mat

# Display the result

print(mat\_neg)

# Define a matrix

mat = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

# Add a row to the matrix

new\_row = np.array([10, 11, 12])

mat\_new\_row = np.vstack((mat, new\_row))

# Remove a column from the matrix

mat\_remove\_col = np.delete(mat, 1, axis=1)

# Display the results

print(mat\_new\_row)

print(mat\_remove\_col)

# Find the maximum value in the matrix

max\_val = np.max(mat)

# Find the minimum value in the matrix

min\_val = np.min(mat)

# Find the maximum value in each column

max\_col = np.max(mat, axis=0)

# Find the minimum value in each row

min\_row = np.min(mat, axis=1)

# Display the results

print(max\_val)

print(min\_val)

print(max\_col)

print(min\_row)

# Find the sum of all elements in the matrix

sum\_all = np.sum(mat)

# Find the sum of elements in each row

sum\_row = np.sum(mat, axis=1)

# Find the sum of elements in each column

sum\_col = np.sum(mat, axis=0)

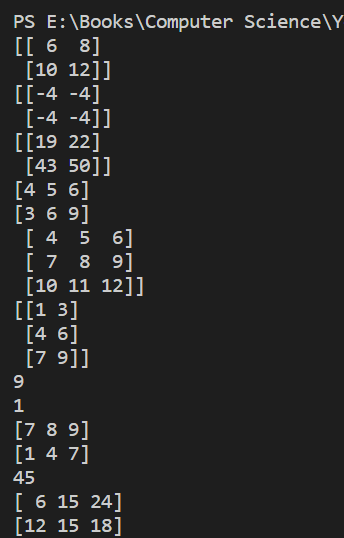
# Display the results

print(sum\_all)

print(sum\_row)

print(sum\_col)

Output: -



8. Create various type of plots/charts like histograms, plot based on sine/cosine function based on data from a matrix. Further label different axes in a plot and data in a plot.

Source Code: -

import numpy as np

import matplotlib.pyplot as plt

# Generate some random data

data = np.random.normal(size=1000)

# Create a histogram

plt.hist(data, bins=30)

# Label the axes and title the plot

plt.xlabel('Value')

plt.ylabel('Frequency')

plt.title('Histogram of Random Data')

# Display the plot

plt.show()

# Generate some data

x = np.linspace(0, 2\*np.pi, 100)

y\_sin = np.sin(x)

y\_cos = np.cos(x)

# Plot the sine and cosine functions

plt.plot(x, y\_sin, label='sin(x)')

plt.plot(x, y\_cos, label='cos(x)')

# Label the axes and title the plot

plt.xlabel('x')

plt.ylabel('y')

plt.title('Sine and Cosine Functions')

# Add a legend

plt.legend()

# Display the plot

plt.show()

# Generate some random data

x = np.random.normal(size=100)

y = np.random.normal(size=100)

# Create a scatter plot

plt.scatter(x, y)

# Label the axes and title the plot

plt.xlabel('x')

plt.ylabel('y')

plt.title('Scatter Plot of Random Data')

# Display the plot

plt.show()

# Generate some data

x = ['A', 'B', 'C', 'D', 'E']

y = [10, 24, 16, 30, 18]

# Create a bar chart

plt.bar(x, y)

# Label the axes and title the plot

plt.xlabel('Category')

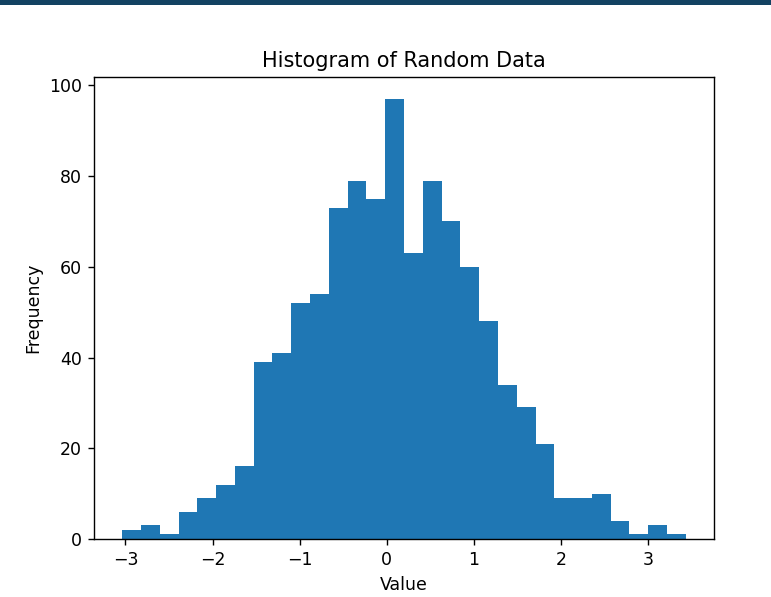
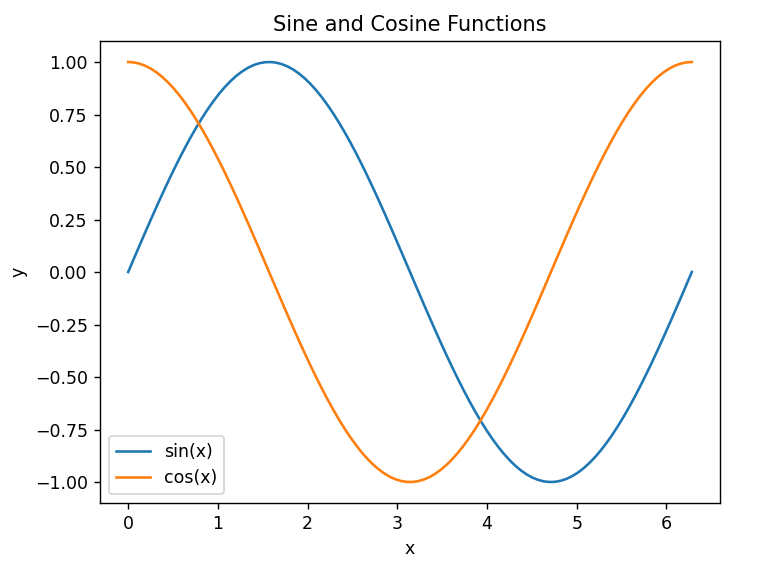
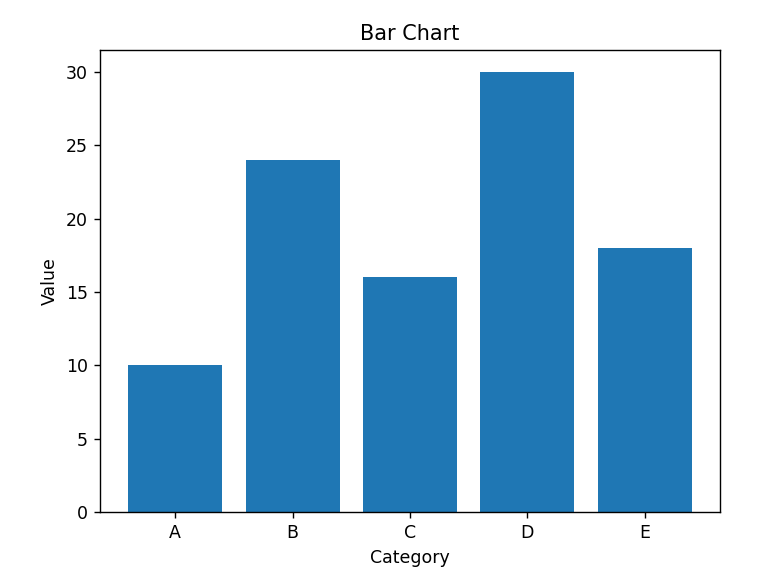
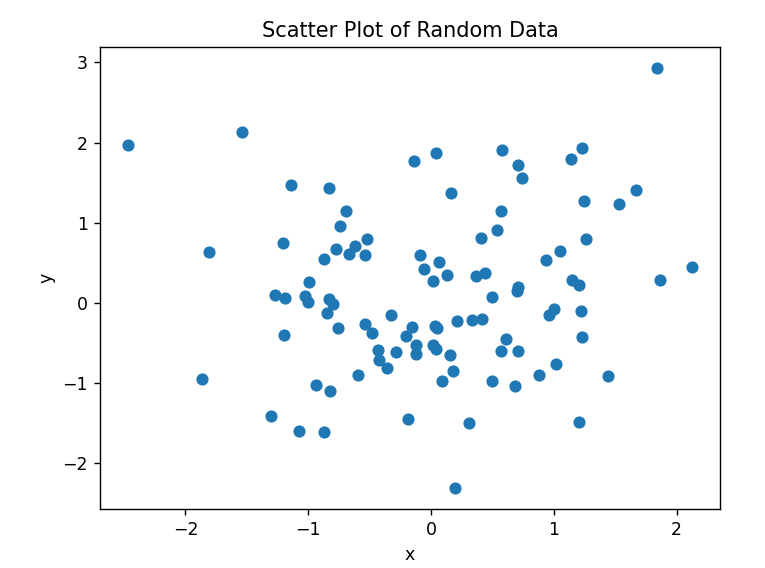
plt.ylabel('Value')

plt.title('Bar Chart')

# Display the plot

plt.show()

Output: -

9. Generate different subplots from a given plot and colour plot data.

Source Code: -

import numpy as np

import matplotlib.pyplot as plt

# Generate some random data

x = np.linspace(0, 2\*np.pi, 100)

y\_sin = np.sin(x)

y\_cos = np.cos(x)

# Create a figure with two subplots

fig, (ax1, ax2) = plt.subplots(1, 2)

# Plot the sine function on the first subplot and color it red

ax1.plot(x, y\_sin, color='red')

# Label the axes and title the first subplot

ax1.set\_xlabel('x')

ax1.set\_ylabel('y')

ax1.set\_title('Sine Function')

# Plot the cosine function on the second subplot and color it blue

ax2.plot(x, y\_cos, color='blue')

# Label the axes and title the second subplot

ax2.set\_xlabel('x')

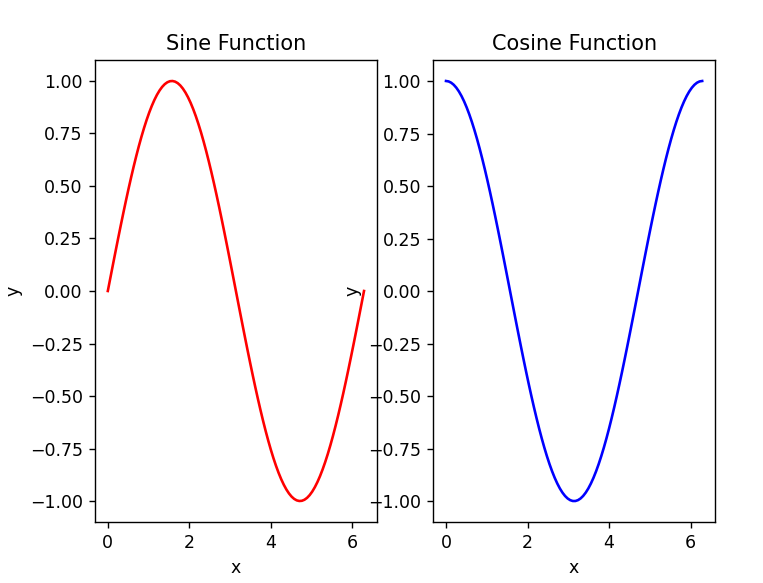
ax2.set\_ylabel('y')

ax2.set\_title('Cosine Function')

# Display the plot

plt.show()

Output: -



10. Use conditional statements and different type of loops based on simple example/s.

Source Code: -

# Example of using conditional statements and loops

# Define a list of numbers

numbers = [1, 2, 3, 4, 5]

# Define a variable to store the sum of the even numbers

sum\_even = 0

# Define a variable to store the product of the odd numbers

product\_odd = 1

# Loop through the numbers list

for num in numbers:

    # Check if the number is even

    if num % 2 == 0:

        # Add the even number to the sum

        sum\_even += num

    # Check if the number is odd

    elif num % 2 != 0:

        # Multiply the odd number to the product

        product\_odd \*= num

# Print the sum of even numbers and the product of odd numbers

print("The sum of even numbers is: ", sum\_even)

print("The product of odd numbers is: ", product\_odd)

# Example of using while loop to find the factorial of a number

# Define the number

num = 5

# Define a variable to store the factorial

factorial = 1

# Loop until the number becomes zero

while num > 0:

    # Multiply the factorial with the number

    factorial \*= num

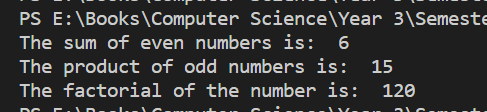
    # Decrement the number

    num -= 1

# Print the factorial

print("The factorial of the number is: ", factorial)

Output: -



11. Perform vectorized implementation of simple matrix operation like finding the transpose of a matrix, adding, subtracting or multiplying two matrices.

Source Code: -

import numpy as np

# Define two matrices

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

B = np.array([[5, 6], [7, 8]])

# Transpose of a matrix

A\_T = A.T

print("Transpose of matrix A:\n", A\_T)

# Adding two matrices

C = A + B

print("Sum of matrices A and B:\n", C)

# Subtracting two matrices

D = A - B

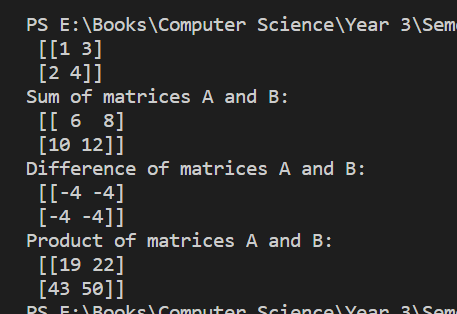
print("Difference of matrices A and B:\n", D)

# Multiplying two matrices

E = A.dot(B)

print("Product of matrices A and B:\n", E)

Output: -



12. Implement Linear Regression problem. For example, based on a dataset comprising of existing set of prices and area/size of the houses, predict the estimated price of a given house.

Source Code: -

import numpy as np

import pandas as pd

from sklearn.linear\_model import LinearRegression

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

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

# Load dataset

data = pd.read\_csv("house\_data.csv")

# Split dataset into training and testing sets

X = data.iloc[:, :-1].values

y = data.iloc[:, -1].values

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=0)

# Create linear regression model and fit it to training data

regressor = LinearRegression()

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

# Make predictions on testing data and evaluate performance

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

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

# Print performance metrics

print("Mean squared error: {:.2f}".format(mse))

print("R-squared value: {:.2f}".format(r2))

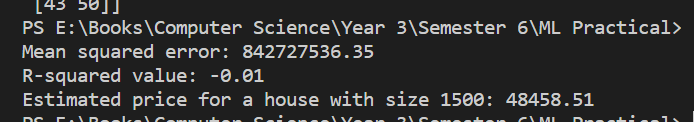
# Predict the estimated price of a given house

new\_house\_size = 1500

new\_house\_price = regressor.predict([[new\_house\_size]])

print("Estimated price for a house with size {}: {:.2f}".format(new\_house\_size, new\_house\_price[0]))

Output: -



13. Based on multiple features/variables perform Linear Regression. For example, based on a number of additional features like number of bedrooms, servant room, number of balconies, number of houses of years a house has been built – predict the price of a house.

Source Code: -

import pandas as pd

from sklearn.linear\_model import LinearRegression

# Load the dataset

data = pd.read\_csv('house\_data1.csv')

# Select the relevant features as independent variables

X = data[['bedrooms', 'servant\_room', 'balconies', 'years\_old']]

# Select the price column as the dependent variable

y = data['price']

# Fit a multiple linear regression model

model = LinearRegression()

model.fit(X, y)

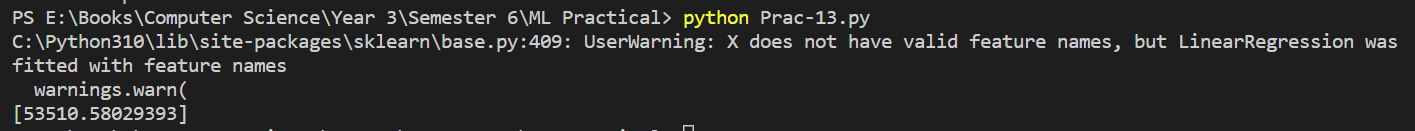
# Predict the price of a new house based on its features

new\_house = [[3, 1, 2, 5]]

predicted\_price = model.predict(new\_house)

print(predicted\_price)

Output: -



14. Implement a classification/ logistic regression problem. For example based on different features of students data, classify, whether a student is suitable for a particular activity. Based on the available dataset, a student can also implement another classification problem like checking whether an email is spam or not.

Source Code: -

import pandas as pd

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

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score

# Load the student dataset

student\_data = pd.read\_csv("student\_data.csv")

# Prepare the data

X = student\_data[['feature1', 'feature2', 'feature3', 'feature4']] # Features

y = student\_data['label'] # Target variable

# Split the 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)

# Create a logistic regression model

model = LogisticRegression()

# Train the model

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

# Make predictions on the test data

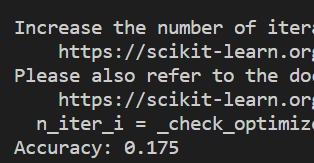
y\_pred = model.predict(X\_test)

# Calculate accuracy

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

print("Accuracy:", accuracy)

Output: -



15. Use some function for regularization of dataset based on problem 14.

Source Code: -

import pandas as pd

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

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score

# Load the student dataset

student\_data = pd.read\_csv("student\_data.csv")

# Prepare the data

X = student\_data[['feature1', 'feature2', 'feature3', 'feature4']] # Features

y = student\_data['label'] # Target variable

# Split the 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)

# Create a logistic regression model with L1 regularization

# You can also use 'l2' for L2 regularization

model = LogisticRegression(penalty='l1', solver='liblinear')

# Train the model

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

# Make predictions on the test data

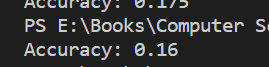
y\_pred = model.predict(X\_test)

# Calculate accuracy

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

print("Accuracy:", accuracy)

Output: -



16. Use some function for neural networks, like Stochastic Gradient Descent or backpropagation - algorithm to predict the value of a variable based on the dataset of problem 14.

Source Code: -

import numpy as np

import pandas as pd

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

from sklearn.neural\_network import MLPClassifier

from sklearn.metrics import accuracy\_score

# Load the student dataset

student\_data = pd.read\_csv("student\_data.csv")

# Prepare the data

X = student\_data[['feature1', 'feature2', 'feature3', 'feature4']] # Features

y = student\_data['label'] # Target variable

# Split the 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)

# Create a multi-layer perceptron (MLP) classifier with SGD optimizer

# You can configure the number of hidden layers and neurons per layer

# Note that MLPClassifier in scikit-learn uses 'relu' as the default activation function

model = MLPClassifier(hidden\_layer\_sizes=(100,), activation='relu', solver='sgd', learning\_rate\_init=0.1)

# Train the model

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

# Make predictions on the test data

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

# Calculate accuracy

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

print("Accuracy:", accuracy)

Output: -

