MACHINE LEARNING PRACTICAL FILE

Abhineet Raman(2002078)

B.SC.(H.) COMPUTER SCIENCE 3RD YEAR

Semester: 6

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: -

```
PS E:\Books\Computer Science\Year 3\Semester 6\ML Practical> python Prac-1.py
Sum: 15
Difference: 4
Product: 12
Quotient: 5.0
Result: 16
```

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: -

```
PS E:\Books\Computer Science
OR: True
AND: False
Equal: False
NOT: False
XOR: True
```

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: -

```
PS E:\Books\Computer Science\
x = 10
y = 3.14
z = True
Hello, Alice!
The length of the name is 5
My age is 25 years old.
My height is 1.75 meters.
```

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: -

```
XOR: True
PS E:\Books\Computer Science\Year 3\Semester 6\ML Practical>
x = 10
y = 3.14
z = True
 [0. 0. 0. 0.]]
[0.23582338 0.42989213 0.87081486 0.05435027 0.70058066]
[[ 0.96587521  0.50910388  0.52287401  0.96342861]
 [ 0.39605085  0.3659617  -0.2760184   0.23104591]
 [-0.28323533 -0.29429482  0.26115548 -0.10790906]]
[[1 0 0 0]
 [0 2 0 0]
 [0 0 3 0]
 [0 0 0 4]]
PS E:\Books\Computer Science\Year 3\Semester 6\ML Practical>
```

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: -

```
PS E:\Books\Computer Science\Year
(3, 4)
4 3
[[ 1.  2.  3. 10.]
 [ 4.  5.  6. 11.]
 [ 7.  8.  9. 12.]]
[[ 1.  2.  3.  4.]
 [ 5.  6.  7.  8.]
 [ 9. 10. 11. 12.]]
var1 <class 'int'>
var2 <class 'str'>
var3 <class 'numpy.ndarray'>
```

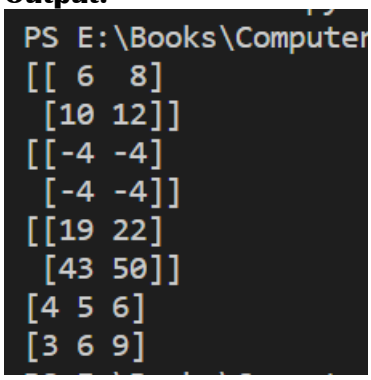
6. Perform basic operations on matrices (like addition, subtraction, multiplication) and display specific rows or columns of the matrix.

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: -

```
PS E:\Books\Computer Science\Y
[[ 6  8]
 [10 12]]
[[-4 -4]
 [-4 -4]]
[[19 22]
 [43 50]]
[4 5 6]
[3 6 9]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
[[1 3]
 [4 6]
 [7 9]]
9
1
[7 8 9]
[1 4 7]
45
[ 6 15 24]
[12 15 18]
```

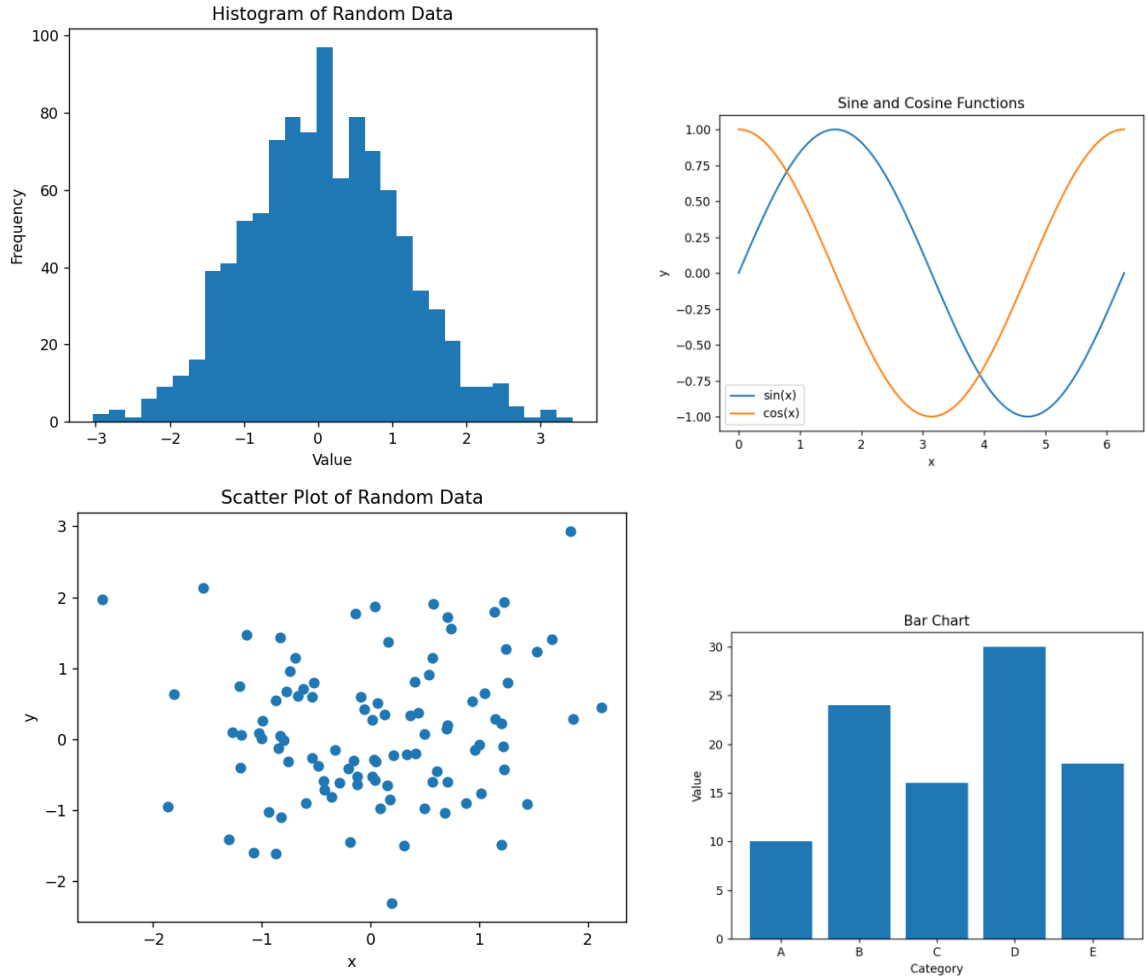
8. Create various type of plots/charts like histograms, plot based on sine/cosine function based on data from a matrix. Further label different axes in a plot and data in a plot.

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: -



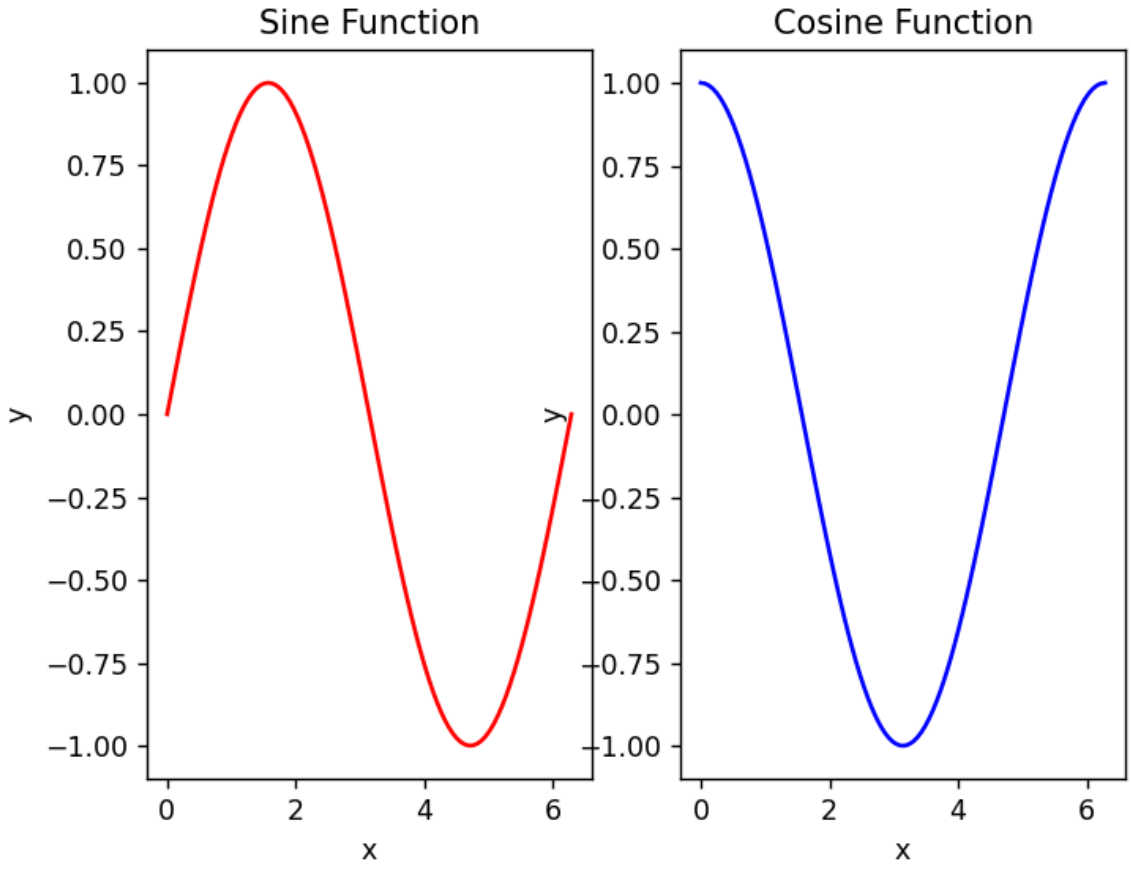
9. Generate different subplots from a given plot and colour plot data.

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: -



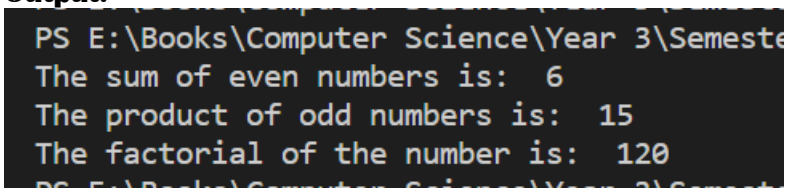
10. Use conditional statements and different type of loops based on simple example/s.

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: -

```
PS E:\Books\Computer Science\Year 3\Sem
 [[1 3]
 [2 4]]
Sum of matrices A and B:
 [[ 6  8]
 [10 12]]
Difference of matrices A and B:
 [[-4 -4]
 [-4 -4]]
Product of matrices A and B:
 [[19 22]
 [43 50]]
PS E:\Books\Computer Science\Year 3\Sem
```

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: -

```
  [45 50]]
PS E:\Books\Computer Science\Year 3\Semester 6\ML Practical>
Mean squared error: 842727536.35
R-squared value: -0.01
Estimated price for a house with size 1500: 48458.51
```

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: -

```
PS E:\Books\Computer Science\Year 3\Semester 6\ML Practical> python Prac-13.py
C:\Python310\lib\site-packages\sklearn\base.py:409: UserWarning: X does not have valid feature names, but LinearRegression was
fitted with feature names
  warnings.warn(
[53510.58029393]
```

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: -

```
Increase the number of iter
    https://scikit-learn.or
Please also refer to the do
    https://scikit-learn.or
  n_iter_i = _check_optimiz
Accuracy: 0.175
```

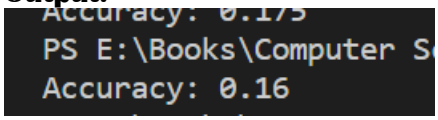
15. Use some function for regularization of dataset based on problem 14.

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: -