**Name : Mozeb Ahmed Khan**

**Roll No: 20F-0161**

**Sec: BS(CS)-7A**

**Assignment: 01**

**Course: Applied Machine Learning**

**Question 1:**

**Part a:**

**Code:**

import numpy as np  
import pandas as pd  
  
*# Load the training data*DataX = pd.read\_csv("DataX.csv") *# Replace with the actual path to your input data file*DataY = pd.read\_csv("DataY.csv") *# Replace with the actual path to your output data file  
  
# Preprocess the data  
# Add a bias term (intercept) to the input features*DataX.insert(0, "Bias", 1)  
  
*# Convert data to NumPy arrays*X = DataX.to\_numpy()  
Y = DataY.to\_numpy()  
  
*# Define hyperparameters*learning\_rate = 0.02  
num\_iterations = 1000  
  
*# Initialize the weights with zeros*theta = np.zeros(X.shape[1])  
  
*# Gradient Descent algorithm*for i in range(num\_iterations):  
 *# Compute the predictions* predictions = X.dot(theta)  
  
 *# Compute the error (the difference between predictions and actual values)* error = predictions - Y.flatten()  
  
 *# Compute the gradient* gradient = X.T.dot(error) / len(Y)  
  
 *# Update the weights using the gradient and learning rate* theta -= learning\_rate \* gradient  
  
*# Print the learned weights (theta)*print("Learned Weights (theta):")  
print(theta)

**Output:**

**Part b:**

**Code:**

import numpy as np  
import pandas as pd  
  
*# Load the training data*DataX = pd.read\_csv("DataX.csv") *# Replace with the actual path to your input data file*DataY = pd.read\_csv("DataY.csv") *# Replace with the actual path to your output data file  
  
# Add a bias term (intercept) to the input features*DataX.insert(0, "Bias", 1)  
  
*# Convert data to NumPy arrays*X = DataX.to\_numpy()  
y = DataY.to\_numpy()  
  
*# Calculate the parameters (theta) using the closed-form solution*theta = np.linalg.inv(X.T.dot(X)).dot(X.T).dot(y)  
  
*# Print the learned parameters (theta)*print("Learned Parameters (theta):")  
print(theta)

**Output:**

**Part c:**

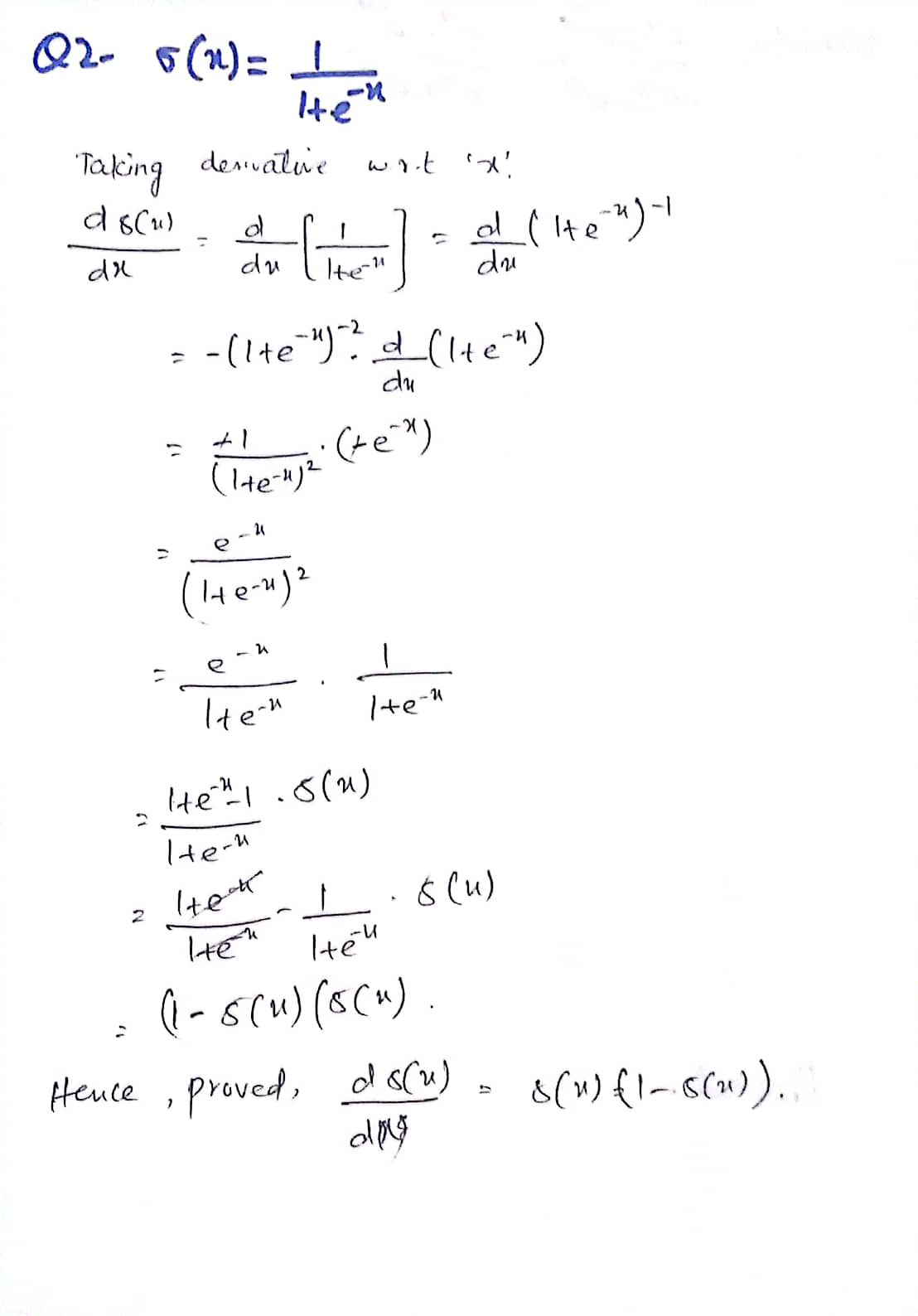
**Code:**

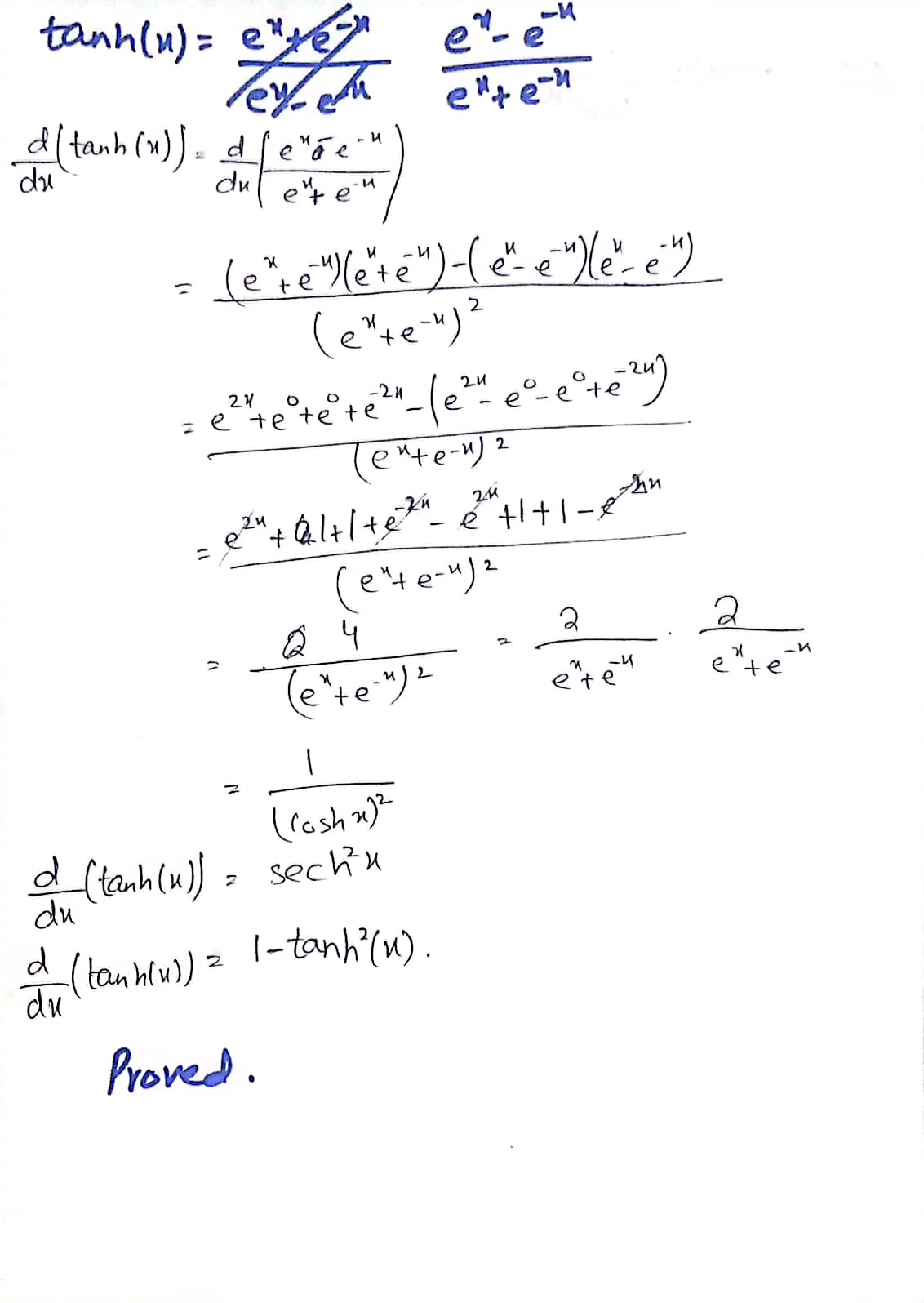
**Output:**

**Part d:**

I think logistic regression used for modelling technique to minimize the loss function and gradient descent is an optimization technique used for training puposes. Thus, both have merits and demerits and we can choose algorithm on basis of factors like data size, resources, and problem definition etc.

**Question 2:**

****

****

**Question 3:**

**Code:**

import numpy as np  
import pandas as pd  
from sklearn.model\_selection import train\_test\_split  
from sklearn.linear\_model import LogisticRegression  
from sklearn.metrics import accuracy\_score, classification\_report  
  
*# Load the input data and labels*DataX = pd.read\_csv("DataX.csv") *# Replace with the actual path to your data file*ClassY = pd.read\_csv("ClassY.csv") *# Replace with the actual path to your class labels file  
  
# Split the data into training and testing sets (80% train, 20% test)*X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(DataX, ClassY, test\_size=0.2, random\_state=42)  
  
*# Initialize and train the logistic regression model*model = LogisticRegression()  
model.fit(X\_train, Y\_train)  
  
*# Make predictions on the test set*Y\_pred = model.predict(X\_test)  
  
*# Evaluate the model*CheckAccuracy = accuracy\_score(Y\_test, Y\_pred)  
print(f"Accuracy: {CheckAccuracy \* 100:.2f}%")  
  
*# Print classification report for detailed metrics*classReport = classification\_report(Y\_test, Y\_pred)  
print("\nClassification Report:")  
print(classReport)

**Output:**

**The End.**

**Thank You.**