```python
# Homework 1 | Problem 1 | Question 1 | Part C | Conor X Devlin
"""
Let d = 10 and N = 100, and σ = 0.01. Write a Python program to solve a linear regression
problem using SGD with T = 2000 iterations and a constant learning rate. You may generate
the data such that each entries of xi and w□ are drawn from standard normal distribution.
Plot the linear regression loss function L(wt) versus t = 1, . . . , T and report the learning rate
you used. Submit your code as a PDF file.

d = 10 | N = 100 | σ = 0.01 | T = 2000 | lr=?

"""
import numpy as np
import matplotlib.pyplot as plt

def loss_function(N, X, w, y):
    """ From Notes/Part A/B--Using Emp. Squared Loss L(w) = (1/(2N)) ||Xw - y||^2 """
    r = X @ w - y
    w_L = ((1 / (2*N)) * (r @ r))
    return w_L

def run_sgd(d=10, N=100, sigma=0.01, T=2000, lr=0.01):
    rng = np.random.default_rng(63)

    # Generating Data
    X = rng.standard_normal((N,d))          # Data Matrix
    #print(f"Data Matrix {X}\n")
    w_true = rng.standard_normal(d)         # Truth Weights
    print(f"True Weights {w_true}\n")
    y = X @ w_true + sigma * rng.standard_normal(N) # Noisy Targets

    # Initialize Weights
    w = np.zeros(d)
    final_loss = 0.0
    # Graph for all the data points
    loss_graph = np.empty(T)

    for t in range(T):
        i = rng.integers(0, N)          # Stochastically picking a sample
        x_i = X[i]
        y_i = y[i]
        grad = (x_i @ w - y_i) * x_i       # Stochastic Grad. min L(w) = (xi^T * w - yi) * xi [Part B]
        w -= lr * grad                  # eta * grad
        w_final = loss_function(N, X, w, y)
        loss_graph[t] = w_final

    print(f"Learning Rate:  {lr}")
    print(f"Final SGD Loss: {w_final}")
    print(f"True Weights    {w_true}\n")

    # Plotting the linear regression loss function L(w_t) versus t = 1, . . . , T
    plt.figure()
    plt.plot(np.arange(1, T + 1), loss_graph)
    plt.xlabel(f"Iteration t [0..2000] && d = {d}")
    plt.ylabel("Loss Function: L(w_t) = (1/(2N)) ||Xw - y||^2")
    plt.title(f"Linear Regression via SGD (Learning Rate = {lr})")
    plt.grid(True)
    plt.tight_layout()
    plt.savefig("loss_curve.png", dpi=150)
    plt.show()

if __name__ == "__main__":
    #d = 10           # dimensions Either 10 or 200
    d = 200
    N = 100          # dimensions
    sigma = 0.01     # σ
    T = 2000         # Iterations
    lr = 0.01        # Learning Rate (Constant)
    run_sgd(d, N, sigma, T, lr)
```