import torch

import torch.nn as nn

import torch.optim as optim

import pandas as pd

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

from sklearn.preprocessing import StandardScaler

# Load your dataset

df = pd.read\_csv('path/to/your/file.csv')

# Assuming the last column is the target variable

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

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

# Split the data into train and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Standardize the data

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

# Convert data to PyTorch tensors

X\_train = torch.tensor(X\_train, dtype=torch.float32)

X\_test = torch.tensor(X\_test, dtype=torch.float32)

y\_train = torch.tensor(y\_train, dtype=torch.float32)

y\_test = torch.tensor(y\_test, dtype=torch.float32)

# Define a simple neural network

class SimpleNN(nn.Module):

def \_\_init\_\_(self, input\_size):

super(SimpleNN, self).\_\_init\_\_()

self.fc1 = nn.Linear(input\_size, 64)

self.fc2 = nn.Linear(64, 32)

self.fc3 = nn.Linear(32, 1)

def forward(self, x):

x = torch.relu(self.fc1(x))

x = torch.relu(self.fc2(x))

x = self.fc3(x)

return x

# Instantiate the model, define the loss function and the optimizer

input\_size = X\_train.shape[1]

model = SimpleNN(input\_size)

criterion = nn.MSELoss() # Assuming a regression problem

optimizer = optim.Adam(model.parameters(), lr=0.001)

# Training the model

epochs = 100

for epoch in range(epochs):

model.train()

optimizer.zero\_grad()

outputs = model(X\_train)

loss = criterion(outputs.squeeze(), y\_train)

loss.backward()

optimizer.step()

if (epoch+1) % 10 == 0:

print(f'Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.4f}')

# Evaluate the model

model.eval()

with torch.no\_grad():

predictions = model(X\_test).squeeze()

test\_loss = criterion(predictions, y\_test)

print(f'Test Loss: {test\_loss.item():.4f}')