import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
from scipy import signal  
from sklearn.preprocessing import MinMaxScaler  
  
cwd = "/Users/zhoumanqing/Documents/pycharm/811/EMG\_Data\_Assignment1.xlsx"  
dataFrame = pd.read\_excel(cwd, delimiter=',',)  
  
  
class SignalProcess(object):  
  
 def \_\_init\_\_(self, dataFrame):  
 dataFrame.drop(dataFrame.columns[[8, 9, 10, 11, 12]], axis=1, inplace=True)  
 self.dataFrame = dataFrame  
  
 def get\_metrics(self):  
 return self.dataFrame.describe()  
  
 def scaled\_vals(self):  
 scaler = MinMaxScaler(feature\_range=(0, 1))  
 rescaled\_values = scaler.fit\_transform(self.dataFrame)  
 x = rescaled\_values.reshape(8, 2048)  
 return x  
  
 # def freq\_dom(self):  
 # fourier\_list = [  
 # for i in self.matched\_linear\_filter():  
 # fourier\_list.append(np.fft.fft(i))  
 # fourier\_list = np.asarray(fourier\_list)  
 # return fourier\_list  
  
 def rms\_val(self):  
 filtered\_vals = []  
 for i in self.differential\_emg():  
 rms = np.sqrt(np.mean(i)\*\*2)  
 filtered\_vals.append(rms)  
 return filtered\_vals  
  
 def white\_noise(self):  
 # mu\_list = []  
 # for i in self.scaled\_vals():  
 # mu, sigma = np.mean(i), np.std(i)  
 # mu\_list.append(np.random.normal(mu, sigma, 20))  
 # # mu, sigma = np.mean(self.scaled\_vals()), np.std(self.scaled\_vals())  
 # return mu\_list  
 return np.random.normal(2\*self.scaled\_vals() + 2, 15)  
 white\_noise\_list = []  
 for i in self.scaled\_vals():  
 mu = np.mean(i)  
 sigma = np.std(i)  
 print(mu, sigma)  
 white\_noise\_list.append(np.random.normal(mu, sigma, 2048))  
 return white\_noise\_list  
  
 def filter\_signal(self):  
 # set order and threshold of butterworth signal  
 b, a = signal.butter(4, 0.2, 'low')  
  
 # create the filtered signal  
 filtered\_signal = []  
 for i in self.white\_noise():  
 output\_signal = signal.filtfilt(b, a, i)  
 filtered\_signal.append(output\_signal)  
  
 filtered\_signal = np.asarray(filtered\_signal)  
 # print(filtered\_signal.shape) # (8, 2048)  
 return filtered\_signal  
  
 def matched\_linear\_filter(self):  
 linear\_filter\_vals = []  
 for i in self.filter\_signal():  
 corr = signal.correlate(i, np.ones(128), mode='same') / 128  
 linear\_filter\_vals.append(corr)  
 return linear\_filter\_vals  
  
  
 def signal\_plotter(self):  
 x\_axis = []  
 x\_vals = 0.029  
 for n in range(0, 2048):  
 x\_axis.append(x\_vals)  
 x\_vals = x\_vals + 0.029  
  
 for i, k in enumerate(self.matched\_linear\_filter()):  
 plt.subplot(8, 1, i+1)  
 plt.plot(x\_axis, k, label=i)  
 plt.legend()  
 plt.xlabel('time (seconds)')  
 # plt.xlabel('frequency (hertz)')  
 plt.ylabel('voltage (millivolts)')  
 # plt.tight\_layout()  
 plt.show()  
  
  
 def differential\_emg(self):  
 return -np.diff(self.matched\_linear\_filter(), axis=0)  
  
  
 def sqrt\_vals(self):  
 rms\_list = []  
 for i in self.rms\_val():  
 rms\_list.append(np.sqrt(i))  
 return rms\_list  
  
 def signal\_plotter(self):  
 x\_axis = []  
 x\_vals = 0.029  
 for n in range(0, 2048):  
 x\_axis.append(x\_vals)  
 x\_vals = x\_vals + 0.029  
  
 for i, k in enumerate(self.matched\_linear\_filter()):  
 plt.subplot(8, 1, i + 1)  
 plt.plot(x\_axis, k, label=i)  
 plt.legend()  
 plt.xlabel('time (milliseconds)')  
 # plt.xlabel('frequency (hertz)')  
 plt.ylabel('voltage (millivolts)')  
 # plt.tight\_layout()  
 plt.show()  
  
  
  
x = SignalProcess(dataFrame)  
x.signal\_plotter()