**M Junaid Farooq   
20F-0427  
BCS-8B**

**-------------------------------------------------------------------------------------------------------------**

**Code of making CSV’s**

import os  
import wfdb  
import csv  
  
  
tracker = 0  
# Open the RECORDS.txt file and read addresses  
with open('RECORDS.txt', 'r') as f:  
 addresses = [line.strip() for line in f]  
  
 file\_index = 1  
# Loop through each address line  
for address in addresses:  
 # Extract the first part of the address (assuming it's the base name)  
 base\_name = address.split()[0]  
  
 print("NOW FILE IS : ", base\_name)  
 # Initialize file index  
  
 # Loop to generate file names with incrementing indices  
 while True:  
  
 folder\_path = base\_name  
 file\_count = len([name for name in os.listdir(folder\_path) if os.path.isfile(os.path.join(folder\_path, name))])  
  
 file\_count=file\_count-1  
 file\_count=file\_count/2  
 print(file\_count, "--", tracker)  
  
 if tracker == file\_count:  
 tracker = 0  
 break  
  
 # Format the file name with padded zeros  
 file\_name = os.path.join(base\_name, f"JS{file\_index:05d}")  
  
 #print("hi 01")  
 # Check if file exists before processing  
 if os.path.exists(file\_name + ".hea"):  
 try:  
  
 #print("hi 02")  
 # Read the WFDB file using `rdsamp` function  
 signals, metadata = wfdb.rdsamp(file\_name)  
  
 # Access metadata information  
 fs = metadata['fs']  
 sig\_len = metadata['sig\_len']  
 n\_sig = metadata['n\_sig']  
 sig\_names = metadata['sig\_name']  
 units = metadata['units']  
 comments = metadata['comments']  
  
 # Open a new CSV file for writing  
  
 csv\_file\_name = os.path.join("CSV", f"JS{file\_index:05d}.csv")  
 with open(csv\_file\_name, 'w', newline='') as csvfile:  
 writer = csv.writer(csvfile)  
  
 # Write the header row to the CSV file  
 header = ['time'] + sig\_names  
 writer.writerow(header)  
  
 # Write the data to the CSV file  
 for i in range(sig\_len):  
 time\_stamp = i / fs # Calculate the time stamp  
 row = [time\_stamp] + [signals[i][j] for j in range(n\_sig)]  
 writer.writerow(row)  
  
 print(f"File {file\_name} processed successfully.")  
 tracker = tracker + 1  
  
 except Exception as e:  
 print(f"Error processing file {file\_name}: {str(e)}")  
 os.system('PAUSE')  
 tracker = tracker + 1  
  
 # Increment the file index  
 file\_index += 1  
  
 else:  
 # Handle the case where the file doesn't exist (optional)  
 print(f"File not found: {file\_name}")  
 file\_index += 1

**Code of reducing noise**

import os  
import numpy as np  
import math  
import sys  
import wfdb  
import csv  
  
def NLM\_1dDarbon(signal,Nvar,P,PatchHW):  
 if isinstance(P,int): # scalar has been entered; expand into patch sample index vector  
 P = P-1 #Python start index from 0  
 Pvec = np.array(range(-P,P+1))  
 else:  
 Pvec = P # use the vector that has been input  
 signal = np.array(signal)  
 #debug = [];  
 N = len(signal)  
  
 denoisedSig = np.empty(len(signal)) #NaN \* ones(size(signal));  
 denoisedSig[:] = np.nan  
 # to simpify, don't bother denoising edges  
 iStart = PatchHW+1  
 iEnd = N - PatchHW  
 denoisedSig[iStart: iEnd] = 0  
  
 #debug.iStart = iStart;  
 #debug.iEnd = iEnd;  
  
 # initialize weight normalization  
 Z = np.zeros(len(signal))  
 cnt = np.zeros(len(signal))  
  
 # convert lambda value to 'h', denominator, as in original Buades papers  
 Npatch = 2 \* PatchHW + 1  
 h = 2 \* Npatch \* Nvar\*\*2  
  
 for idx in Pvec: # loop over all possible differences: s - t  
 # do summation over p - Eq.3 in Darbon  
 k = np.array(range(N))  
 kplus = k + idx  
 igood = np.where((kplus >=0) & (kplus < N)) # ignore OOB data; we could also handle it  
 SSD = np.zeros(len(k))  
 SSD[igood] = (signal[k[igood]] - signal[kplus[igood]])\*\*2  
 Sdx = np.cumsum(SSD)  
  
 for ii in range(iStart,iEnd): # loop over all points 's'  
 distance = Sdx[ii + PatchHW] - Sdx[ii - PatchHW-1] #Eq 4;this is in place of point - by - point MSE  
 # but note the - 1; we want to icnlude the point ii - iPatchHW  
  
 w = math.exp(-distance/h) # Eq 2 in Darbon  
 t = ii + idx # in the papers, this is not made explicit  
  
 if t>0 and t<N:  
 denoisedSig[ii] = denoisedSig[ii] + w \* signal[t]  
 Z[ii] = Z[ii] + w  
 cnt[ii] = cnt[ii] + 1  
 print('ii',ii)  
 print('t',t)  
 print('w',w)  
 print('denoisedSig[ii]', denoisedSig[ii])  
 print('Z[ii]',Z[ii])  
 # loop over shifts  
  
 # now apply normalization  
 denoisedSig = denoisedSig/(Z + sys.float\_info.epsilon)  
 denoisedSig[0: PatchHW+1] =signal[0: PatchHW+1]  
 denoisedSig[ - PatchHW: ] =signal[- PatchHW: ]  
 #debug.Z = Z;  
  
def process\_file(mat\_file\_path, hea\_file\_path, result\_folder, file\_name):  
 # Read the WFDB file using rdsamp function  
 signals, metadata = wfdb.rdsamp(hea\_file\_path)  
  
 # Access metadata information  
 fs = metadata['fs']  
 sig\_len = metadata['sig\_len']  
 n\_sig = metadata['n\_sig']  
 sig\_names = metadata['sig\_name']  
 units = metadata['units']  
 comments = metadata['comments']  
  
 # Apply denoising to each signal  
 denoised\_signals = []  
 for signal\_idx in range(n\_sig):  
 signal = signals[:, signal\_idx]  
  
 # Apply NLM denoising  
 Nvar = np.std(signal)  
 P = 3 # Example value for the patch size  
 PatchHW = 1 # Example value for half patch size  
 denoised\_signal = NLM\_1dDarbon(signal, Nvar, P, PatchHW)  
  
 denoised\_signals.append(denoised\_signal)  
  
 # Prepare CSV file name  
 csv\_file\_name = os.path.join(result\_folder, f"{file\_name}\_denoised.csv")  
  
 # Write denoised signals to CSV file  
 with open(csv\_file\_name, 'w', newline='') as csvfile:  
 writer = csv.writer(csvfile)  
  
 # Write the header row to the CSV file  
 header = ['time'] + sig\_names  
 writer.writerow(header)  
  
 # Write the denoised signals to the CSV file  
 for i in range(sig\_len):  
 time\_stamp = i / fs # Calculate the time stamp  
 row = [time\_stamp] + [denoised\_signals[j][i] for j in range(n\_sig)]  
 writer.writerow(row)  
  
 print(f"File {file\_name} processed successfully.")  
  
  
# Main script remains mostly the same  
def process\_directory(directory, base\_dir, result\_base\_dir):  
 result\_dir = os.path.join(result\_base\_dir, directory)  
 os.makedirs(result\_dir, exist\_ok=True)  
  
 # Check if the directory contains a RECORDS file  
 records\_filename = 'RECORDS'  
 records\_dir = os.path.join(base\_dir, directory.replace('/', os.sep)) # Normalize path separator  
 records\_file\_path = os.path.join(records\_dir, records\_filename+'.txt')  
  
 if os.path.exists(records\_file\_path):  
 # Read the list of file names from the RECORDS file  
 with open(records\_file\_path, 'r') as records\_file:  
 for line in records\_file:  
 file\_name = line.strip()  
 mat\_file\_path = os.path.join(base\_dir, directory, file\_name + '.mat').replace('/', os.sep)  
 hea\_file\_path = os.path.join(base\_dir, directory, file\_name).replace('/', os.sep)  
  
 # Check if both .mat and .hea files exist  
 if os.path.exists(mat\_file\_path):  
 #Process the file  
 process\_file(mat\_file\_path, hea\_file\_path, result\_dir,file\_name)  
  
 # Recursively process subdirectories  
 for subdir in os.listdir(records\_dir):  
 subdir\_path = os.path.join(records\_dir, subdir)  
 if os.path.isdir(subdir\_path):  
 process\_directory(subdir, base\_dir, result\_base\_dir)  
  
  
# Directory containing the first RECORDS file (which contains paths to subfolders)  
records\_directory = 'data'  
first\_records\_filename = 'RECORDS.txt'  
  
# Directory to store processed results  
result\_directory = 'results\_denoised'  
  
# Read the list of subfolder paths from the first RECORDS file  
with open(os.path.join(records\_directory, first\_records\_filename), 'r') as first\_records\_file:  
 subfolder\_paths = first\_records\_file.read().splitlines()  
  
# Iterate over each subfolder path from the first RECORDS file  
for subfolder\_path in subfolder\_paths:  
 process\_directory(subfolder\_path, records\_directory, result\_directory)Code for PCA (Dimensionality Reduction)  
Code for PCA(Dimensionality Reduction)  
import pandas as pd  
import numpy as np  
from sklearn.decomposition import PCA  
import matplotlib.pyplot as plt  
  
# Load ECG data from the CSV file  
filename = 'JS00101'  
data = pd.read\_csv('CSV/' + filename + '.csv')  
  
# Extract ECG signal data (excluding the 'time' column)  
X = data.iloc[:, 1:].values  
  
# Store original column names for labeling  
original\_column\_names = data.columns[1:]  
  
# Apply PCA  
n\_components = 6  
pca = PCA(n\_components=n\_components) # Reduce to specified number of principal components  
X\_pca = pca.fit\_transform(X)  
  
# Get the indices of the principal components with the highest variance  
component\_indices = np.argsort(pca.explained\_variance\_ratio\_)[::-1][:n\_components]  
  
# Get the principal axes in feature space  
principal\_axes = pca.components\_  
  
# Keep track of selected leads  
selected\_leads = set()  
  
# Create a new DataFrame with the principal components and time  
df\_pca = pd.DataFrame({'time': data['time']})  
for i, idx in enumerate(component\_indices):  
 # Find the original attribute name corresponding to the principal component  
 original\_attr\_index = np.argmax(np.abs(principal\_axes[idx]))  
 original\_attr\_name = original\_column\_names[original\_attr\_index]  
  
 # Check if the lead is already selected  
 while original\_attr\_name in selected\_leads:  
 # If lead is repeated, choose the next best one  
 principal\_axes[idx, original\_attr\_index] = 0 # Exclude this lead from consideration  
 original\_attr\_index = np.argmax(np.abs(principal\_axes[idx]))  
 original\_attr\_name = original\_column\_names[original\_attr\_index]  
  
 # Add the selected lead to the set  
 selected\_leads.add(original\_attr\_name)  
  
 df\_pca[f'{original\_attr\_name}'] = X\_pca[:, idx]  
  
# Save the DataFrame to a new CSV file  
output\_file\_name = 'Reduced/' + filename + '\_reduced.csv'  
df\_pca.to\_csv(output\_file\_name, index=False)  
  
# Plot the principal components against time  
plt.figure(figsize=(12, 6))  
for col in df\_pca.columns[1:]:  
 plt.plot(df\_pca['time'], df\_pca[col], label=col)  
plt.xlabel('Time')  
plt.ylabel('Amplitude')  
plt.title('ECG Signals Projected onto the First Four Principal Components')  
plt.legend()  
plt.show()

**Code of visualization of noise**

import pandas as pd  
import matplotlib.pyplot as plt  
  
# Read the CSV file into a DataFrame  
df = pd.read\_csv('JS00004.csv')  
  
# Extract the necessary columns for visualization  
time = df['time']  
lead\_I = df['I']  
lead\_II = df['II']  
lead\_III = df['III']  
lead\_AVF = df['aVF']  
lead\_AVL = df['aVL']  
lead\_AVR = df['aVR']  
lead\_V1 = df['V1']  
lead\_V2 = df['V2']  
lead\_V3 = df['V3']  
lead\_V4 = df['V4']  
lead\_V5 = df['V5']  
lead\_V6 = df['V6']  
  
  
# Create the plot  
plt.figure(figsize=(10, 6))  
plt.plot(time, lead\_I, label='Lead I')  
plt.plot(time, lead\_II, label='Lead II')  
plt.plot(time, lead\_III, label='Lead III')  
plt.plot(time, lead\_AVF, label='Lead AVF')  
plt.plot(time, lead\_AVL, label='Lead AVL')  
plt.plot(time, lead\_AVR, label='Lead AVR')  
plt.plot(time, lead\_V1, label='Lead V1')  
plt.plot(time, lead\_V2, label='Lead V2')  
plt.plot(time, lead\_V3, label='Lead V3')  
plt.plot(time, lead\_V4, label='Lead V4')  
plt.plot(time, lead\_V5, label='Lead V5')  
plt.plot(time, lead\_V6, label='Lead V6')  
  
  
# Customize the plot  
plt.xlabel('Time')  
plt.ylabel('Voltage (mV)')  
plt.title('ECG Measurements')  
plt.legend()  
  
# Display or save the plot  
plt.show()

**Code of dimensionality reduction using PCA**

import pandas as pd  
import numpy as np  
from sklearn.decomposition import PCA  
import matplotlib.pyplot as plt  
  
# Load ECG data from the CSV file  
data = pd.read\_csv('JS00004.csv')  
  
# Extract ECG signal data (excluding the 'time' column)  
X = data.iloc[:, 1:].values  
  
# Apply PCA  
pca = PCA(n\_components=2) # Reduce to 2 principal components  
X\_pca = pca.fit\_transform(X)  
  
# Create a new DataFrame with the principal components and time  
df\_pca = pd.DataFrame({'time': data['time'], 'PC1': X\_pca[:, 0], 'PC2': X\_pca[:, 1]})  
  
# Plot the principal components against time  
plt.figure(figsize=(12, 6))  
plt.plot(df\_pca['time'], df\_pca['PC1'], label='Principal Component 1')  
plt.plot(df\_pca['time'], df\_pca['PC2'], label='Principal Component 2')  
plt.xlabel('Time')  
plt.ylabel('Amplitude')  
plt.title('Projected ECG Signals onto the Initial Two Principal Components')  
plt.legend()  
plt.show()

**Code of dimensionality reduction using PCA**

import csv  
from biosppy.signals import ecg  
import matplotlib.pyplot as plt  
  
# Data points  
signal = []  
  
# Read ECG signal from the CSV file (assuming the signal is in the second column)  
with open('JS00004.csv', 'r') as file:  
 reader = csv.reader(file)  
 next(reader) # Skip the first row  
 for row in reader:  
 signal.append(float(row[8])) #Lead V1 (heartbeat)  
  
# Process the ECG signal and plot  
out = ecg.ecg(signal=signal, sampling\_rate=1000., show=True)  
  
# Plot the ECG signal with R-peaks  
plt.figure(figsize=(12, 6))  
# Plot the filtered ECG signal with R-peaks  
plt.subplot(1, 2, 1)  
plt.plot(out['ts'], out['filtered'], label='Filtered ECG')  
plt.plot(out['ts'][out['rpeaks']], out['filtered'][out['rpeaks']], 'ro', label='R-peaks')  
plt.xlabel('Time (s)')  
plt.ylabel('Amplitude')  
plt.title('Filtered ECG Signal with R-peaks')  
plt.legend()  
  
# Plot the templates on the right side  
plt.subplot(1, 2, 2)  
for i, template in enumerate(out['templates']):  
 plt.plot(out['templates\_ts'], template, label=f'Template {i+1}')  
plt.xlabel('Time (s)')  
plt.ylabel('Amplitude')  
plt.title('Templates')  
plt.legend()  
  
plt.tight\_layout()  
plt.show()

**Link of Medium Blog**

**https://medium.com/@junaidfarooq0427/unveiling-arrhythmia-patterns-a-deep-dive-into-ecg-data-analysis-and-dimensionality-reduction-c8192555967e**