Lab 7 ReadMe

In lab 7 we are using FFT functions to provide health and audio solutions.

Required Libraries:

1) FFT

```
import numpy as np
from scipy import fftpack, signal
from matplotlib import pyplot as plt
import pandas as pd
from scipy.io import wavfile
import pandas as pd
import sys, os, os.path
```

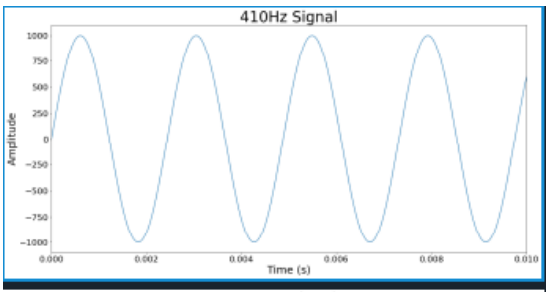
2) Heartpy

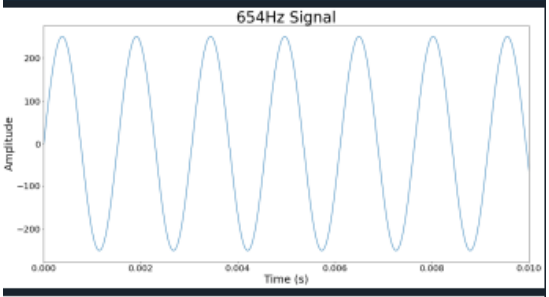
```
import heartpy as hp
import matplotlib.pyplot as plt
```

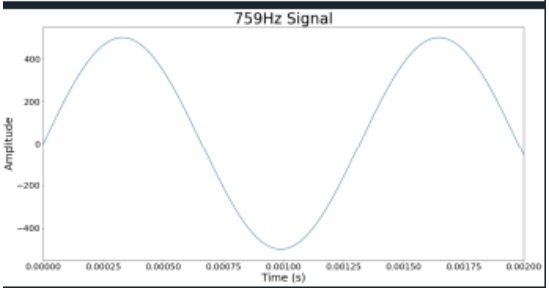
3) Heart disease analysis

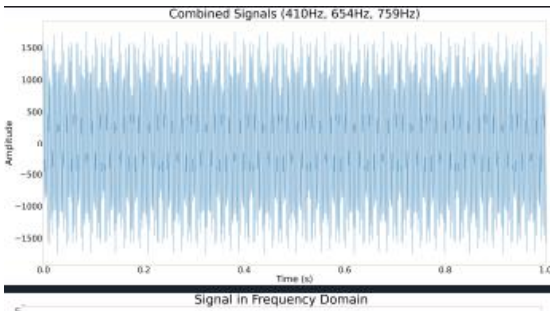
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import math
from scipy import signal
from scipy.io import wavfile
import glob
import seaborn as sns
import os.path
from os import walk
import heartpy as hp
from scipy import fftpack
from pathlib import Path
from scipy.stats import linregress, chi2_contingency
from sklearn import preprocessing,linear_model
from multiprocessing.dummy import Pool as ThreadPool
pool=ThreadPool(12)
```

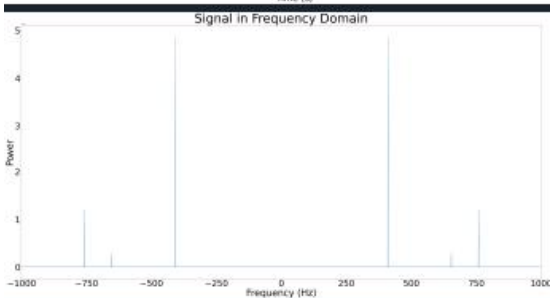
1) FFT audio signal processing
   The code will output a series of filtered and unfiltered graphs that show a signal in both the frequency and time domain. It will display the designated waveforms, waveforms combined, unfiltered and filtered waveforms, all in the time and frequency domain.

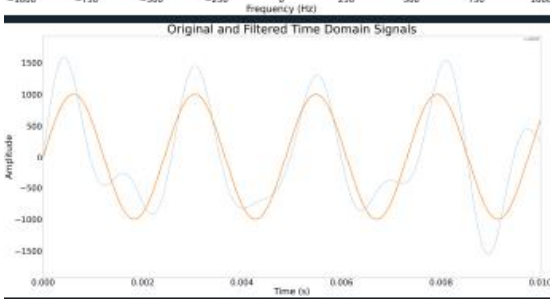


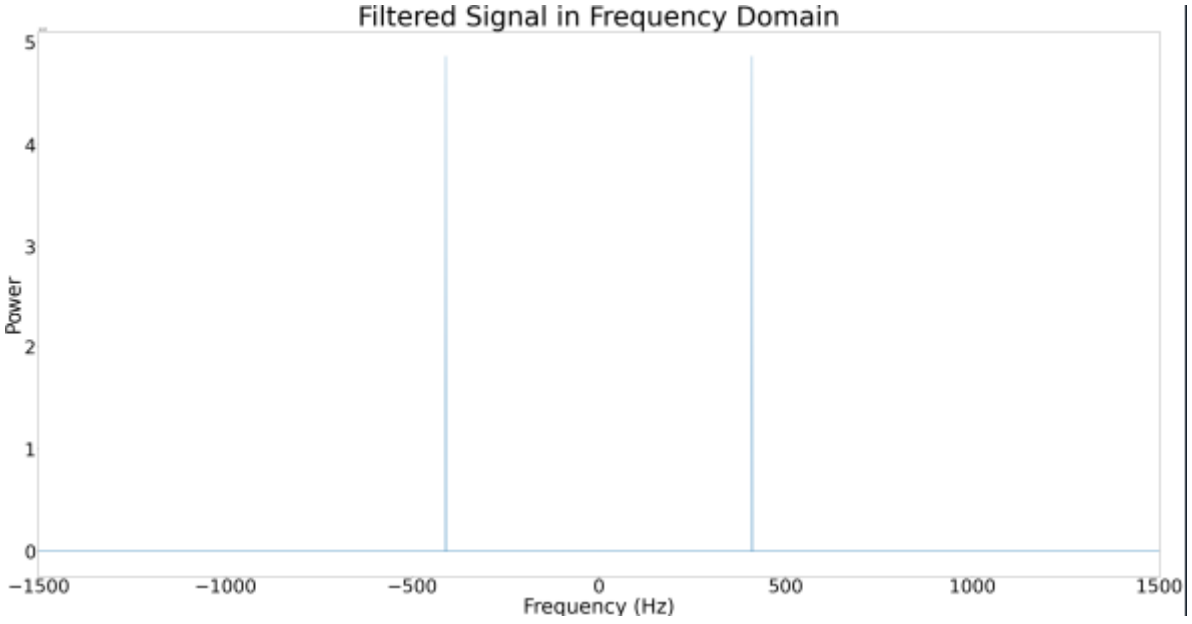










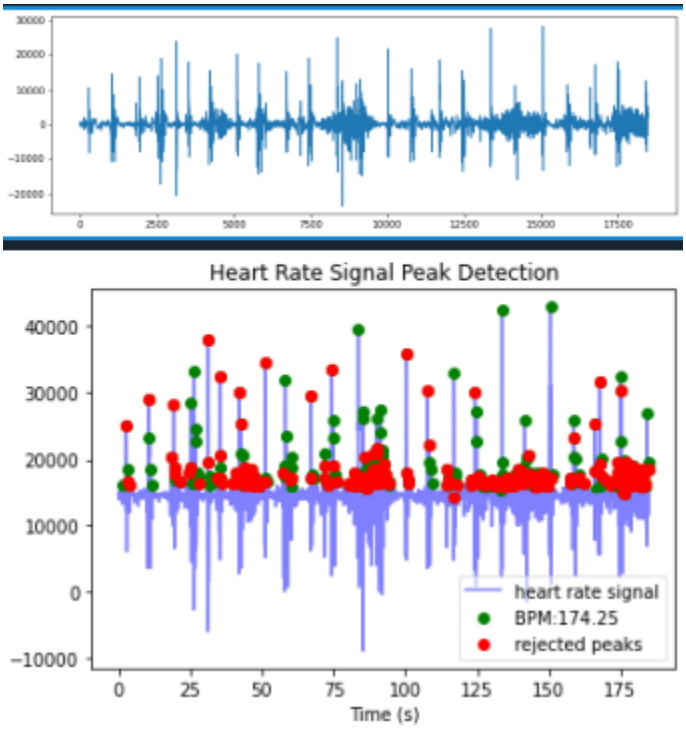


The code works by combining the signals then filtering the signal with a LowPass like filter which removes the high frequencies. Additionally, 2 .wav files will be outputted to show the difference.

2) Heartpy

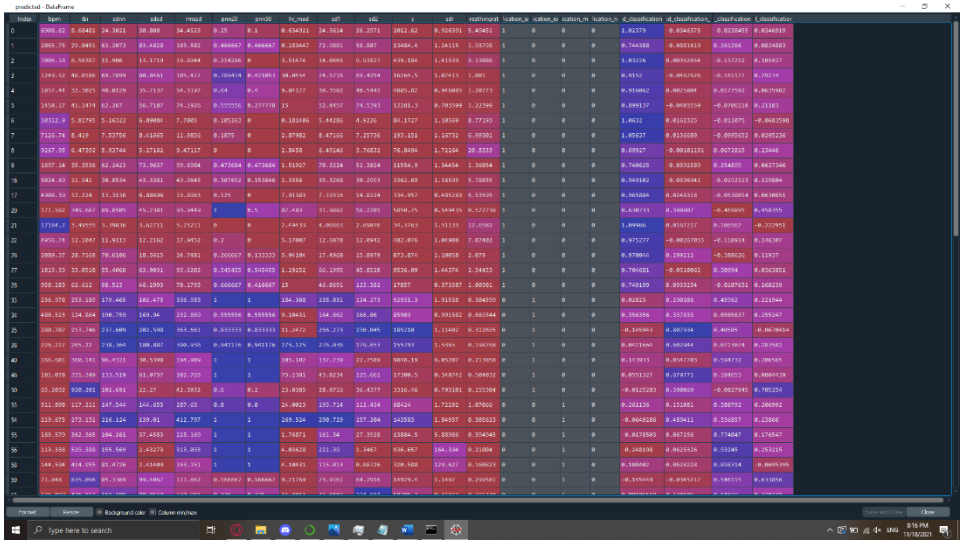
The purpose of this code is to load a heartbeat csv file and have the code read the csv and determine cardiology numbers for expert analysis. The code will display the signal and the output characteristics of the signal in both a graph and chart form.

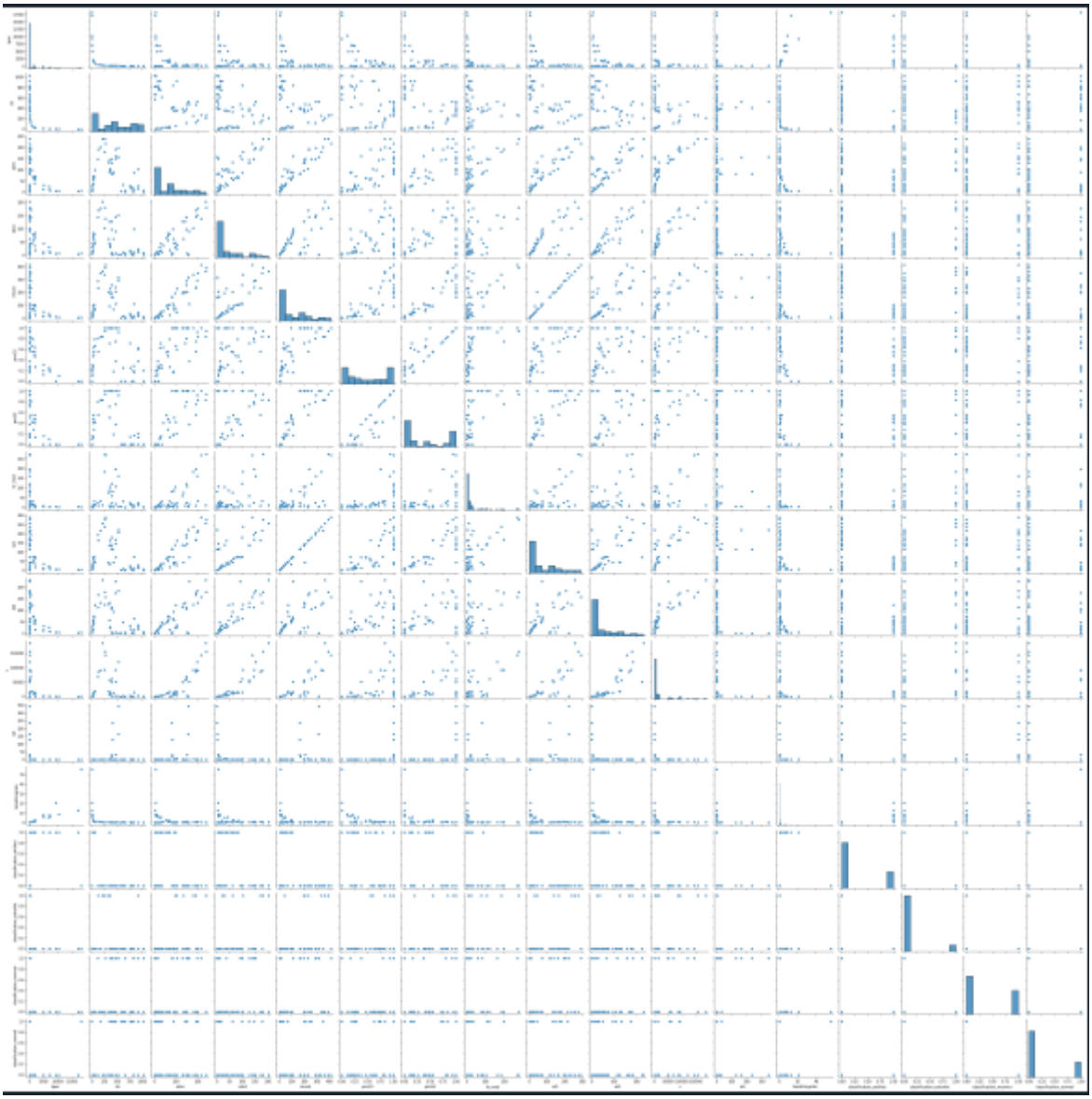
```
bpm: 174.254317
ibi: 344.324324
sdnn: 142.516771
sdsd: 111.123084
rmssd: 184.923971
pnn20: 0.875000
pnn50: 0.750000
hr_mad: 105.000000
sd1: 127.312784
sd2: 153.252339
s: 61295.557267
sd1/sd2: 0.830740
breathingrate: 0.196232
```



3) Heart disease classification

The purpose of this code is to decode a series of heartbeat.wav files into csv files and determine if the csv row has a certain heart disease using linear regression. The code will output a table with prediction of what the row's heart condition is, and a scatter lot that displays the correlation between all the parameters.





Documentation:

1) FTT analysis:
   The code first lets the users define what frequency the program should develop. By changing lines 18,33,48, the user can input their own frequencies. The following function then combines the signals:

```
sig = sig1 + sig2 + sig3
plt.figure(figsize=(60,30))
plt.title('Combined Signals (410Hz, 654Hz, 759Hz)', fontsize=80)
plt.ylabel('Amplitude', fontsize=60)
plt.xlabel('Time (s)', fontsize=60)
plt.xticks(fontsize=55)
plt.yticks(fontsize=55)
plt.xlim(0,1)
plt.plot(time_vec, sig)
plt.savefig('Combined_Signal')
```

It will then display the FFT of the three signals along with the power:

```
sig_fft = fftpack.fft(sig)
power = np.abs(sig_fft)**2
sample_freq = fftpack.fftfreq(sig.size, d=time_step)
plt.figure(figsize=(60, 30))
plt.title('Signal in Frequency Domain', fontsize=80)
plt.ylabel('Power', fontsize=60)
plt.xlabel('Frequency (Hz)', fontsize=60)
plt.xticks(fontsize=55)
plt.yticks(fontsize=55)
plt.xlim(-1000,1000)
plt.plot(sample_freq, power)
plt.savefig('FFT_Unfiltered')
```

By finding the high frequencies, the filter can filter the signals out and leave the lowest frequency behind:

```
#%% finding peak frequency

pos_mask = np.where(sample_freq > 0)
freqs = sample_freq[pos_mask]
peak_freq = freqs[power[pos_mask].argmax()]
```

```
#%% filter out high frequencies

high_freq_fft = sig_fft.copy()
high_freq_fft[np.abs(sample_freq) > peak_freq] = 0
filtered_sig = fftpack.ifft(high_freq_fft)
```

2) Heartpy:
   Th code runs by simply putting a csv into the code in line 9.

```
data=hp.get_data('4.csv')
plt.figure(figsize=(12,4))
plt.plot(data)
plt.show
```

3) Heart Disease Classification:
   The code uses several functions to run.
   - A dataloader:

```
#%%
classfolder = "D:\EE104\lab 7\heart classification\classificatior
classpath = Path(classfolder)
paths = [Path(dir[0]) for dir in walk(classpath)][1:]
```

   - .wav file reader:

```
#%%
def readwav(file:str):
    filepath = Path(file).absolute()
    samplerate, data = wavfile.read((filepath))
    # print(f"samplerate = {samplerate}")
    return data,samplerate
```

-a file processor to turn the .wav into a csv:

```
database = []
def processFiles(wav,classification):
    data, samplerate= generateSignal(wav)
    try:
        wd, m = hp.process(data, samplerate)
        points = [m[measure] for measure in m.keys()]
        database.append([classification,*points])
    except:
        return
```

-signal generation from the .wav file reader and signal filter:

```
lengths = []
def generateSignal(file:str,plot:bool=False,loglevel:str=None):
    data,samplerate = readwav(file)
    length = data.shape[0] / samplerate
    lengths.append(length)
    sig = filterSignal(data)
    return sig, samplerate
#%%

def filterSignal(data:np.ndarray,filter=True):
    ## normalize input
    sig = data/np.amax(data)
    norm_heart = data/np.amax(data)
    sos = signal.butter(10, [.2,195], 'bp', fs=1000, output='sos'
    filtered_heart = signal.sosfilt(sos, sig)
    ## Removing noise
    noise_heart = signal.signaltools.wiener(filtered_heart,300)
    noise_heart = filtered_heart
    if(not filter):
        noise_heart = norm_heart
    return (noise_heart)
```

- Generating a dataframe and plotting a scatterplot

```
df = pd.DataFrame(database,columns=["classification","bpm","ib
df = df.dropna()
print("columns",list(df.columns.values))
database=df[df.columns.values]
sns.pairplot(df,kind="scatter")
plt.show()
```

- Move the classification column into its own column so it can be compared with other columns and generate the correct scatterplot

```
df = pd.get_dummies(df,columns=["classification"])

print("columns",list(df.columns.values))
sns.pairplot(df,kind="scatter")
plt.show()
```

- Clean and perform linear regression on the data

```
cleandf = df.fillna(df.median())
pear_columns = ["bpm","ibi","sdnn","sdsd","rmssd","pnn20","pnn5
X = cleandf[pear_columns]# here we have 2 variables for multipl
#Y = cleandf["classification_normal"]
#A = cleandf["classification_extrahls"]
#B = cleandf["classification_murmur"]
#C = cleandf["classification_artifact"]
# with sklearn
#regr = linear_model.LinearRegression()
#model = regr.fit(X,Y)
#model = regr.fit(X,A)
#model = regr.fit(X,B)
classifications = cleandf.columns[-4:]
predicted = cleandf.copy()
Predictions =[]
for classification in classifications:
    regr = linear_model.LinearRegression()
    fit = regr.fit(X,cleandf[classification]).predict(X)
    predicted["predicted_"+classification] = fit
    #Predictions.append([fit])
```

Conclusions for Heart disease classification:

We noticed a few correlations inside the scatterplot.

-SD1 SD2 and breathing rate were somewhat correlated to a heart having artifact. An artifact is defined by simple having a heart anomaly.

-having a high ibi correlates with having heart murmurs

-having high RMSSD and low SDSD correlated to having extrahls also known as extra heart sounds.

-having a 3% higher than normal PNN50 would correlate to having a heart problem.