LAB 4 Write Up:

6.1

The code starts by creating a sin wave of 50 Hz and 1000 Hz, and then plots them individually as well as together in one plot. Then, the code makes a new figure and plots the combined 50 Hz and 1000 Hz sine waves (as in, adding the two plots mathematically). Then, the code will take the newly combined data plot and find the frequencies present within it using the Fast Fourier Transform provided by Numpy. In doing this, the code will create two new datasets: A non-normalized set from the FFT, and a normalized set from the FFT. The normalized set is plotted to visualize the frequencies, and the non-normalized set is used for later. After this, an array of 0's is created, and filled with the data from the normalized array using the 500-1500 range of frequencies (to filter out the 50 Hz frequency). This is then plotted. The same is done with the non-normalized dataset, except this is not plotted. Next, the code will use the inverse Fast Fourier Transform to convert the dataset from the frequency plane to the time plane. This data is then plotted, along with the original sine wave and again the combined wave.

Something of note is that the converted wave is shifted by pi/2 and becomes a cosine wave. I believe this is due to the loss of information as the data is converted back and forth.

6.2

First, the code plots the signal from the trumpet.wav file. Then, a subsection of the signal, from the 0-48000 samples range is saved to be the segment of use. This data is then transformed using FFT to find the frequencies within the signal and then this is plotted. Next, the code will find the 3 highest peaks in the frequency plane, and then display the frequency values that they occur at. It does this twice, once for the non-filtered dataset (i.e., the dataset that contains all the found frequencies) and once for the filtered dataset, and plots both. Then, to filter out the very high frequencies, a subsection from 1->8000 is taken (this was the range that Audacity presented as well). This range is then plotted and converted back to the time plane using the IFFT function. The code then writes two files, one being the segment file, and the other being the filtered segment file.

Something of note is that the filtered and converted dataset does not seem to play correctly when converted to a wave file (despite my best effort). My thought on this is that the converted data loses a lot of information and becomes very noisy.

6.3

Using the code from the sine() function, 3 different sine waves are made. These are then converted into numpy arrays for wave addition. Then, the resulting wave is plotted, and it's FFT result is plotted as well (showing the frequencies that are supposed to be in the wave). Something interesting is that the frequencies reported are exactly half of those generated, as in, the 1000 Hz frequency becomes 500 Hz, and so on. This combined wave is then written to a .WAV file

6.4

First, the function checks to make sure that the inputted file exists, and if so, continues. Then, it makes sure that the factor of stretching is a floating point (and not a string or something). Next, the wave file is read in (using the code from get_freq()). Then, a wave file is made, and the length is adjusted by changing the number of samples to write (which is 48000 * the factor inputted). The code then generates a new .WAV file called modified_wave, which will pitch up and be less duration depending on if it's squished, or slow down and be longer duration is it's stretched.

6.5

This function starts by defining a set sampling rate, and then gets a indexer value by dividing the original sampling rate of 1000 Hz by the input one and converting it to an integer to ensure it is a real value. Then, the code will iterate through the original signal at the calculated indexer and plot the data of the signal and its frequencies as well.

In general, it's good practice to sample at 10x the rate at which the data is being generated. So, a 5 Hz signal should be sampled at <= 50 Hz to get smooth data. The limit, I think, is sampling at twice the rate at which the data is being generated. This is why when you set the sampling rate to that less than 50 Hz, you start to see less and less smooth data sampled. Regardless of the correct ratio of data rate to sampling rate, the slower you sample the data, the less smooth and unusable it will be. If you sampled at 5 Hz, you would not get enough data to understand how it is changing or be able to make predictions, do operations, etc.