EEG_Research

March 20, 2024

1 Installing Dependencies

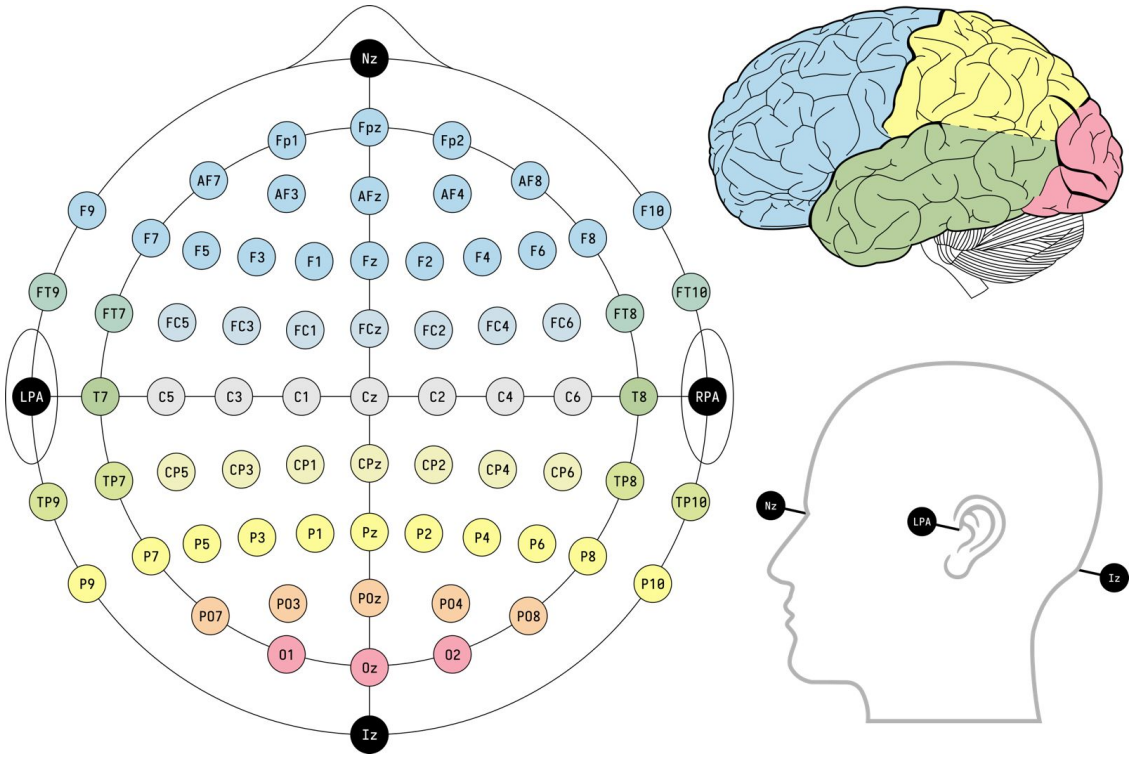
[1]:
```
import pandas as pd
import seaborn as sns
import numpy as np
from scipy import fftpack
from scipy.interpolate import BarycentricInterpolator
import matplotlib.pyplot as plt
```

2 Visualization for EEGs

The EEG files are each given a shape of around 18000x20.

The columns names are provided below. They correspond to the nodes present on the patient's head, illustrated by the naming convention used in the picture.

The only exception is the column named EKG, which stands for the readings of the heart.



The X axis corresponds to time in seconds (s), while the Y axis corresponds to the micro volts observed in brain (μV)

This data is useful for the data model, however, doctors use spectrograms to make more sense, and get a better feel of the data.

[2]:
```
# Load the data, see the column names
eeg_visualization = pd.read_parquet(r'train_eegs/1628180742.parquet',␣
 ↪engine='pyarrow')
eeg_visualization.columns
```

[2]:
```
Index(['Fp1', 'F3', 'C3', 'P3', 'F7', 'T3', 'T5', 'O1', 'Fz', 'Cz', 'Pz',
       'Fp2', 'F4', 'C4', 'P4', 'F8', 'T4', 'T6', 'O2', 'EKG'],
      dtype='object')
```

[3]:
```
# How the data is inputed, it's all float valeus, positive and negative both
eeg_visualization.head()
```

[3]:

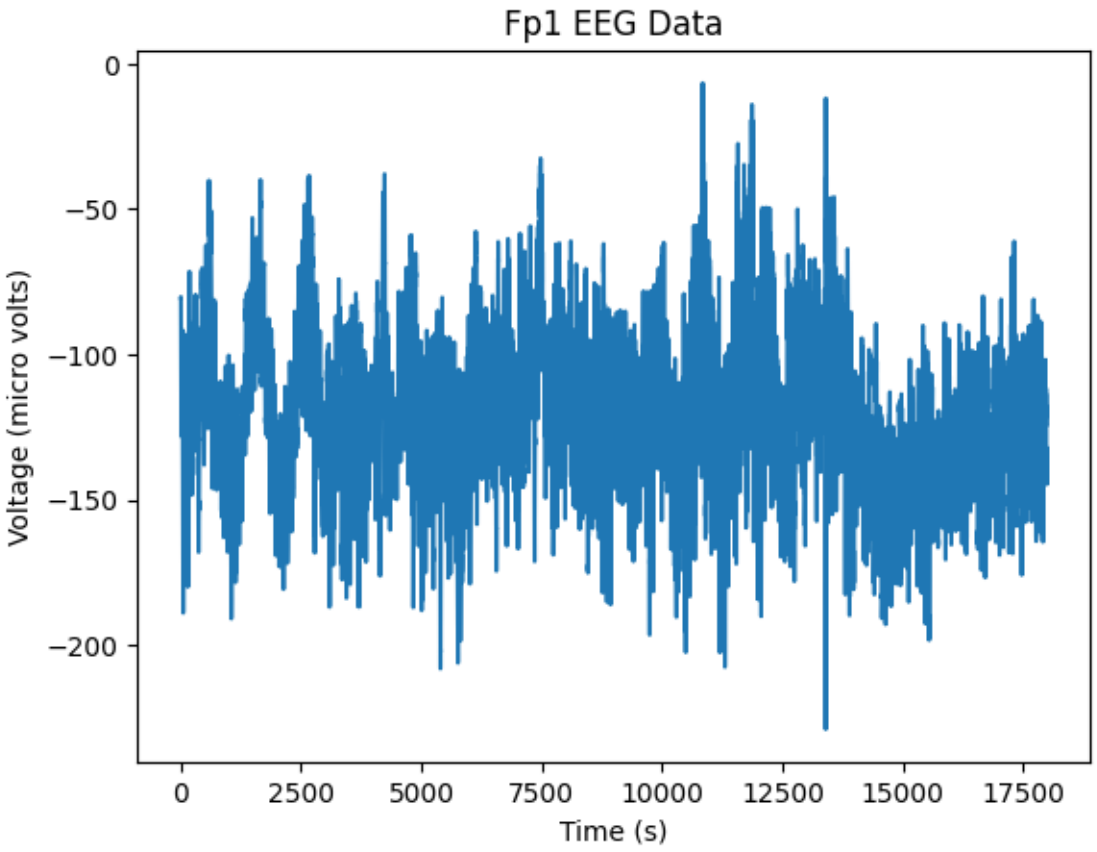
	Fp1	F3	C3	P3	F7	T3	\
0	-80.519997	-70.540001	-80.110001	-108.750000	-120.330002	-88.620003	
1	-80.449997	-70.330002	-81.760002	-107.669998	-120.769997	-90.820000	
2	-80.209999	-75.870003	-82.050003	-106.010002	-117.500000	-87.489998	
3	-84.709999	-75.339996	-87.480003	-108.970001	-121.410004	-94.750000	
4	-90.570000	-80.790001	-93.000000	-113.870003	-129.960007	-102.860001	

	T5	O1	Fz	Cz	Pz	Fp2	\

```
0 -101.750000 -104.489998  -99.129997  -90.389999 -97.040001 -77.989998
1 -104.260002  -99.730003  -99.070000  -92.290001 -96.019997 -84.500000
2  -99.589996  -96.820000 -119.680000  -99.360001 -91.110001 -99.440002
3 -105.370003 -100.279999 -113.839996 -102.059998 -95.040001 -99.230003
4 -118.599998 -101.099998 -107.660004 -102.339996 -98.510002 -95.300003

           F4          C4          P4         F8         T4          T6  \
0  -88.830002 -112.120003 -108.110001 -95.949997 -98.360001 -121.730003
1  -84.989998 -115.610001 -103.860001 -97.470001 -89.290001 -115.500000
2 -104.589996 -127.529999 -113.349998 -95.870003 -96.019997 -123.879997
3 -101.220001 -125.769997 -111.889999 -97.459999 -97.180000 -128.940002
4  -88.930000 -115.639999  -99.800003 -97.500000 -88.730003 -114.849998

           O2        EKG
0 -106.449997   7.920000
1 -102.059998  29.219999
2 -105.790001  45.740002
3 -109.889999  83.870003
4 -100.250000  97.769997
```

[4]:
```
# Taking a sample column to visualize how the data is like, usually we would␣
 ↪look at a smaller range of time values, to see the data better
eeg_visualization.loc[:,"Fp1"].plot(xlabel="Time (s)", ylabel="Voltage (micro␣
 ↪volts)", title="Fp1 EEG Data")
```

[4]:
```
<Axes: title={'center': 'Fp1 EEG Data'}, xlabel='Time (s)', ylabel='Voltage
(micro volts)'>
```

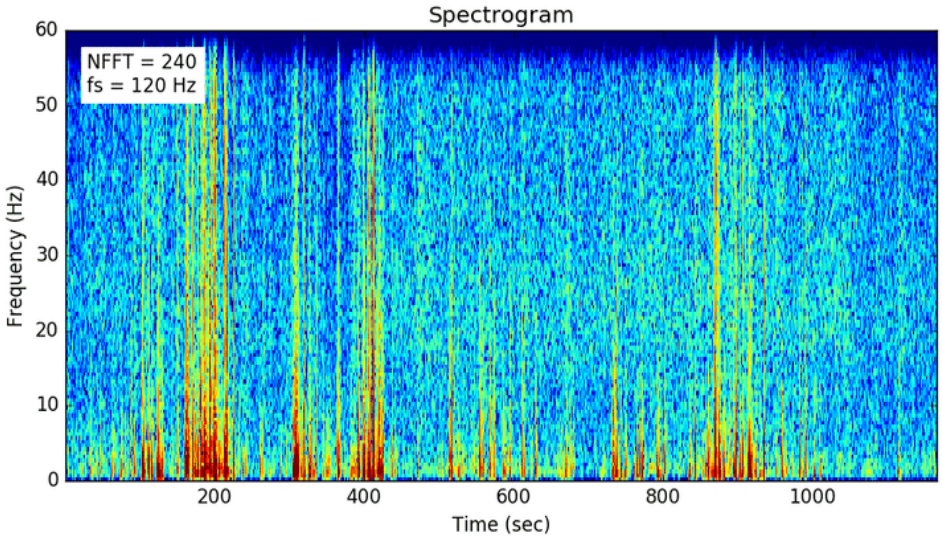


3 Visualizing for Spectrograms

The Spectrograms gives a much in depth analysis into an EEG report. It is a segment of the EEG report, where we see a fourier-transformed analysis of frequencies and their corresponging amplitudes.

I research a few ideal cases, where certain patterns in spectrograms can be used to infer results.

Usually, spectrogram data is plotted in a heat map, where our main focus is on the red portion (called flames)



The deeper colour of red is a query of interest as well. It corresponds to the amplitude/absolute value of the EEG report at a particular point, with a particular given frequency.

In our given spectrogram dataset, we are given a dataset of size 320x401 at an average. 1 of those columns is what defines actual time in seconds, so we have 400 columns of interest.

Columns are divided into 4 parts, LL, RL, LP, RP (Left Lateral, Right Lateral, Left Parasagittal, Right Parasagittal).

Each of the 4 parts have their own range of frequencies. In the above image imagine horizontal slices, which correspond to a particular frequency. That is what is meant by something like LL_0.59. It's a left lateral data at frequency of 0.59 Hz.

Each data point is an absolute value of the amplitude, and the shade of red matters. I will discuss below common cases that help us.

[5]:
```
spectrogram_visualization = pd.read_parquet(r'train_spectrograms/353733.
 ↪parquet', engine='pyarrow')
spectrogram_visualization.columns
```

[5]:
```
Index(['time', 'LL_0.59', 'LL_0.78', 'LL_0.98', 'LL_1.17', 'LL_1.37',
       'LL_1.56', 'LL_1.76', 'LL_1.95', 'LL_2.15',
       …
       'RP_18.16', 'RP_18.36', 'RP_18.55', 'RP_18.75', 'RP_18.95', 'RP_19.14',
       'RP_19.34', 'RP_19.53', 'RP_19.73', 'RP_19.92'],
      dtype='object', length=401)
```

[6]:
```
spectrogram_visualization.head()
```

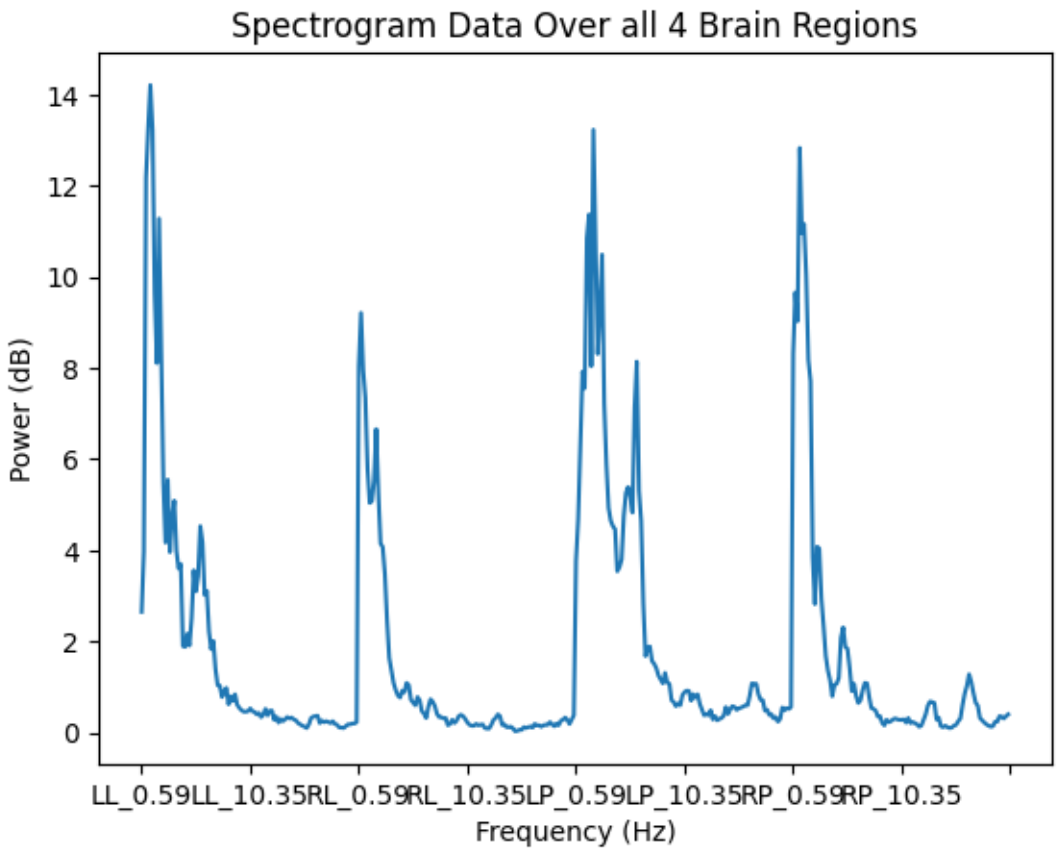
[6]:

	time	LL_0.59	LL_0.78	LL_0.98	LL_1.17	LL_1.37	LL_1.56	LL_1.76	\
0	1	4.26	10.98	9.05	13.65	11.49	8.930000	18.840000	
1	3	2.65	3.97	12.18	13.26	14.21	13.230000	9.650000	
2	5	4.18	4.53	8.77	14.26	13.36	16.559999	19.219999	

```
3     7     2.41     3.21     4.92     8.07     5.97  12.420000  10.820000
4     9     2.29     2.44     2.77     4.62     5.39   7.080000   9.840000

   LL_1.95    LL_2.15  …  RP_18.16  RP_18.36  RP_18.55  RP_18.75  RP_18.95  \
0    19.26  19.240000  …      0.31      0.17      0.28      0.19      0.24
1     8.11  11.280000  …      0.15      0.13      0.14      0.24      0.24
2    17.51  22.650000  …      0.29      0.21      0.16      0.25      0.28
3    14.96  21.809999  …      0.33      0.51      0.49      0.64      0.58
4    12.27  14.410000  …      0.44      0.38      0.48      0.63      0.45

   RP_19.14  RP_19.34  RP_19.53  RP_19.73  RP_19.92
0      0.27      0.29      0.16      0.22      0.19
1      0.36      0.35      0.31      0.36      0.40
2      0.28      0.34      0.48      0.44      0.48
3      0.42      0.32      0.31      0.32      0.33
4      0.45      0.49      0.33      0.31      0.34

[5 rows x 401 columns]
```

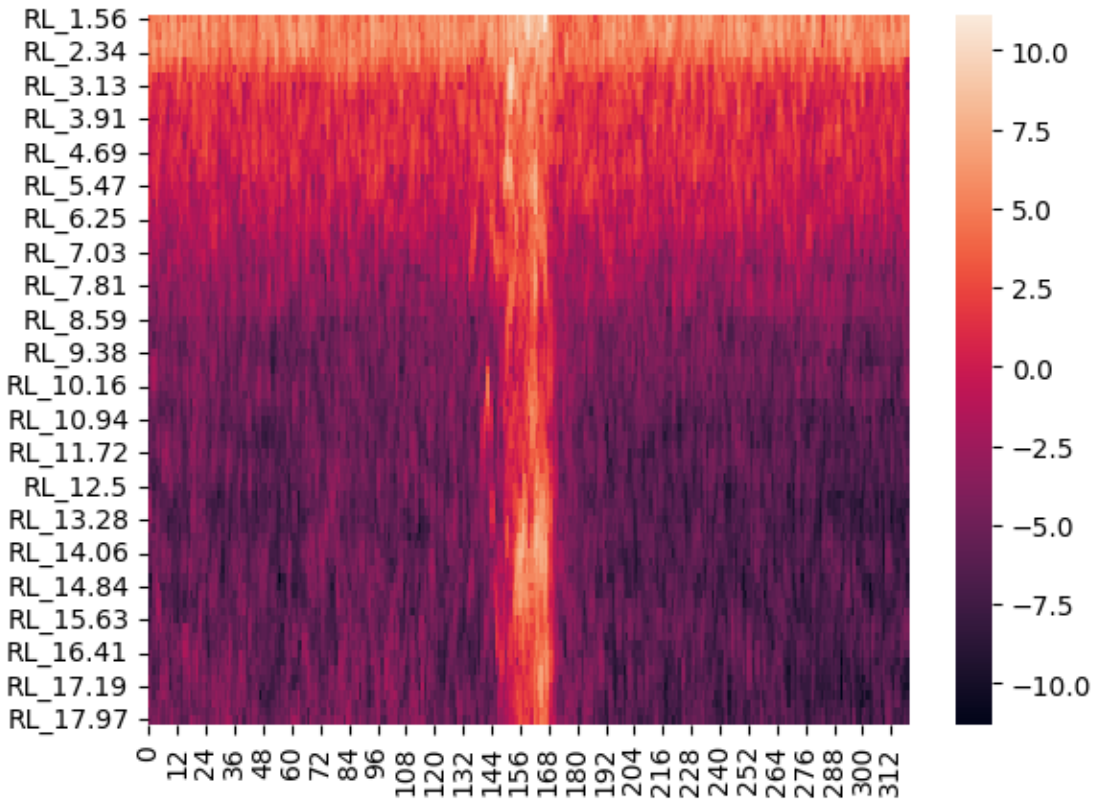
[7]:
```
spectrogram_visualization.loc[1,"LL_0.59":].plot(xlabel="Frequency (Hz)",␣
↪ylabel="Power (dB)", title="Spectrogram Data Over all 4 Brain Regions")
```

[7]:
```
<Axes: title={'center': 'Spectrogram Data Over all 4 Brain Regions'},
xlabel='Frequency (Hz)', ylabel='Power (dB)'>
```



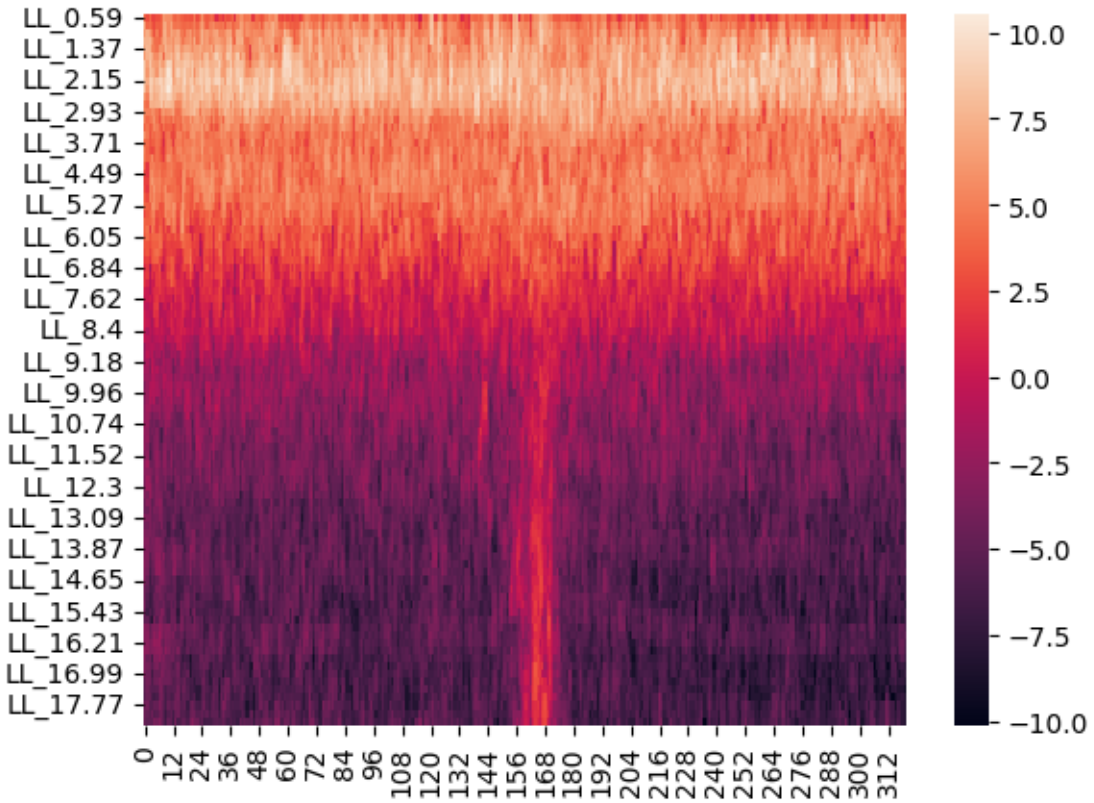
[8]:
```
# Note, I have made the graphs upside down, the lower frequencies should be at␣
↪the bottom
sns.heatmap(2*np.log2(spectrogram_visualization.loc[:,"RL_1.56":"RL_17.97"].T))
```

[8]: <Axes: >



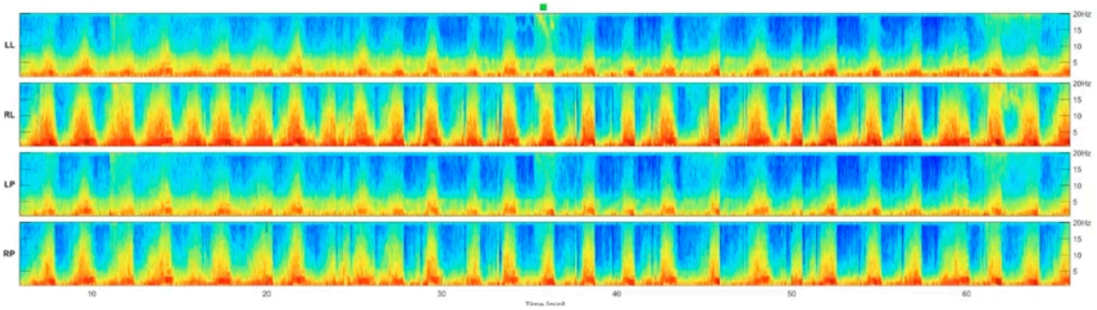
[9]:
```
sns.heatmap(2*np.log2(spectrogram_visualization.loc[:,"LL_0.59":"LL_18.16"].T))
```

[9]: <Axes: >

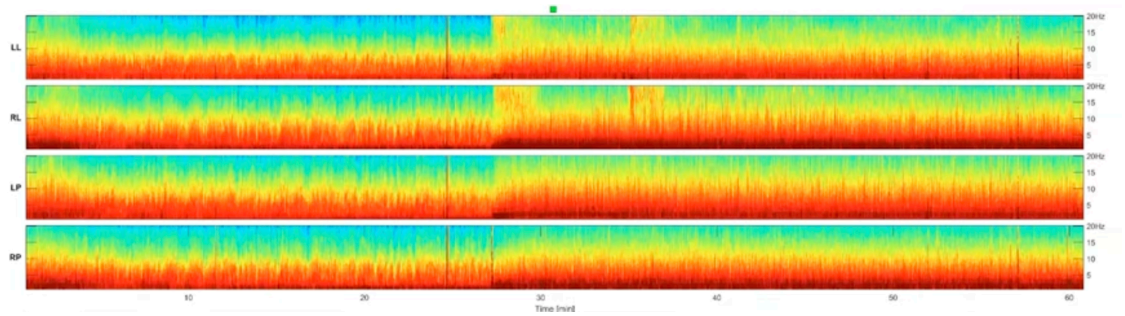


Spectograms can have flames (red patches that show growth and decline), and based on their behaviour, we can predict:

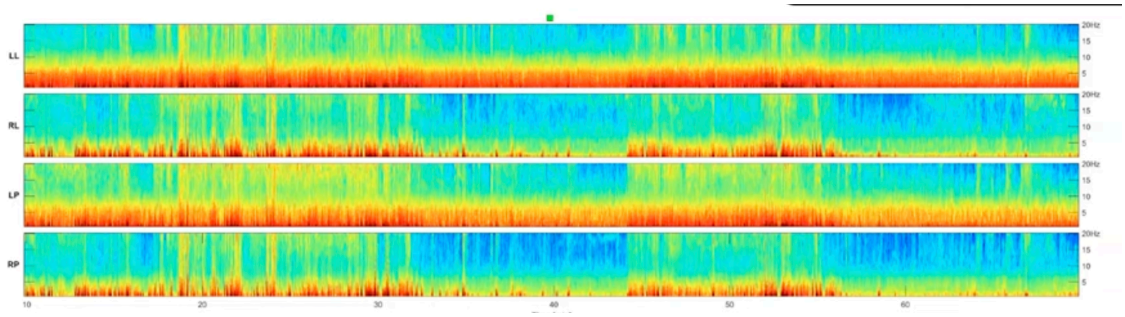
- Regular flames: Seizures



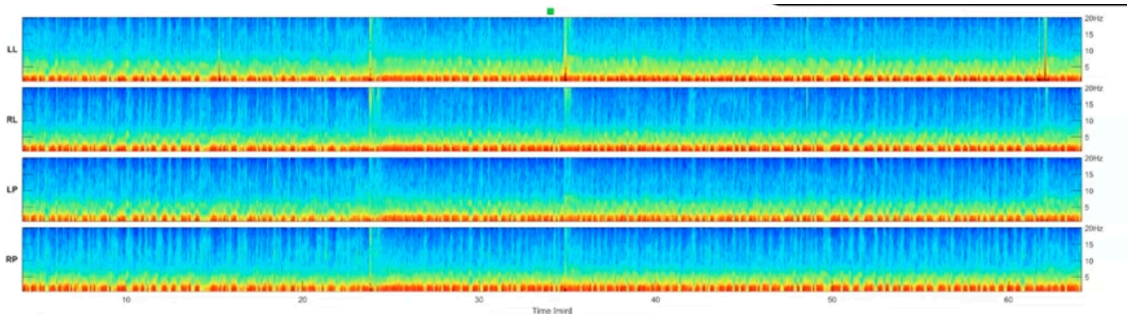
- High Power Flames all over the data set: Something Generalized
- Spikey: Something Periodic
- Long Stretches/Broach Band Monotonous: GPDs



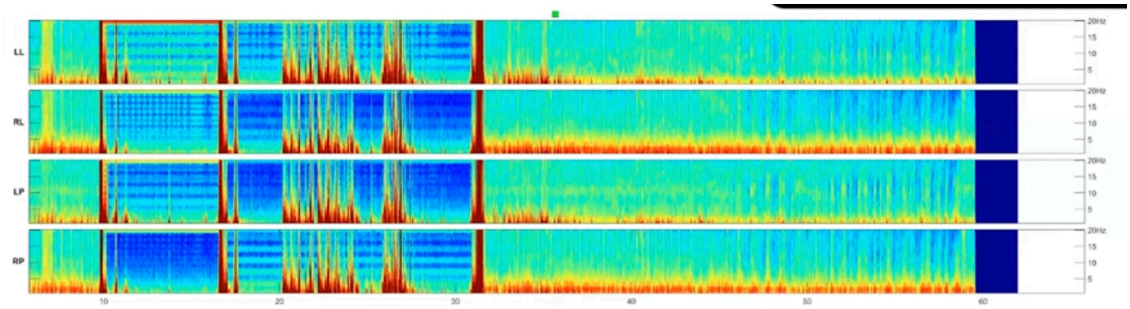
- Long Streches/Broach Band Monotonous but only on one side (left or right): LPDs



- Narrow Band Monotonous: GRDA



- Narrow Band Monotonous but only on one side: LRDA



4 What should be the input data in our model?

According to train.csv, we are given the following data at the very least

- eeg_id, [int]
- eeg_sub_id, [int]
- eeg_label_offset_seconds, [int]
- spectrogram_id, [int]
- spectrogram_sub_id, [int]
- spectrogram_label_offset_seconds, [int]
- label_id, [int]
- patient_id, [int]
- expert_consensus, [string]
- seizure_vote, [int]
- lpd_vote, [int]
- gpd_vote, [int]
- lrda_vote, [int]
- grda_vote, [int]
- other_vote [int]

Which makes a total of 15 data points. But there will be more, considering eeg and spectrogram data.

5 Transforming the EEG by Fourier Transforms

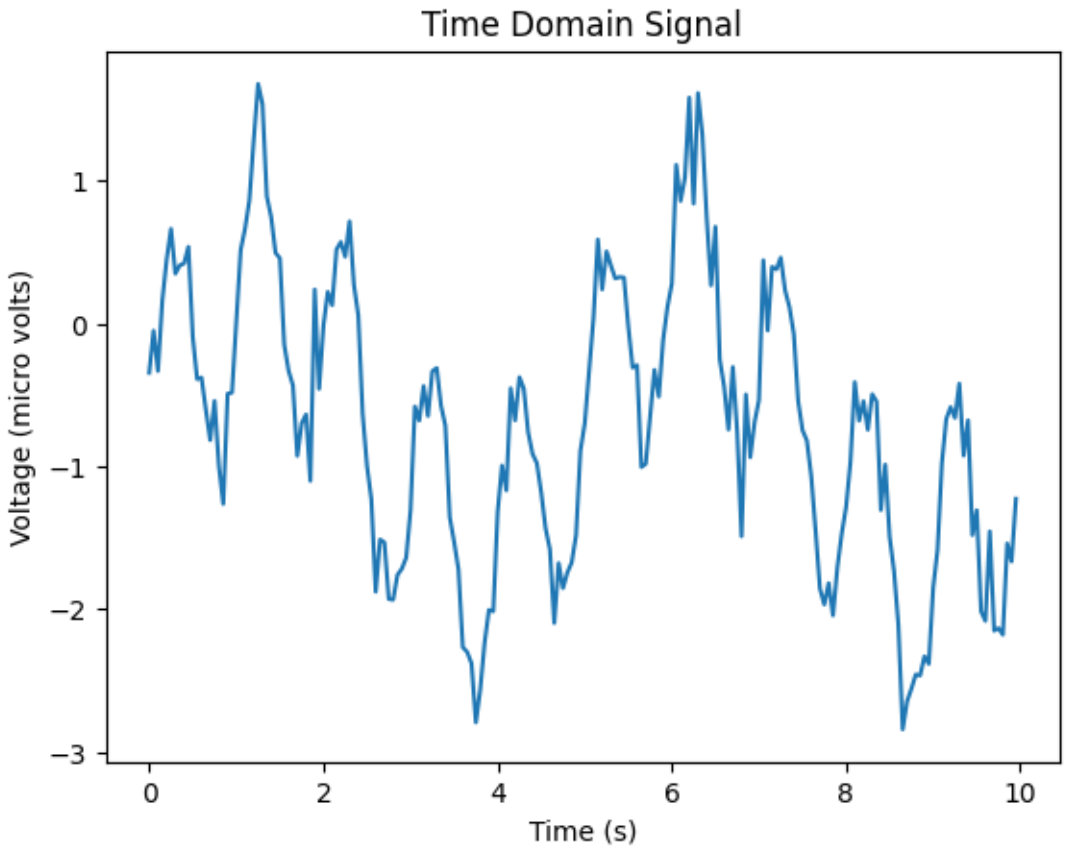
How FFT works

[10]:
```
time_step = 0.05
time_vec = np.arange(0, 10, time_step)
period = 5

# Creating a signal, adding random sin waves and adding some noise
sig = (
        np.sin(2 * np.pi * time_vec / period)
        + 0.25 * np.random.randn(time_vec.size)
        + np.sin(10 * np.pi * time_vec / period)
        + np.sin(1000 * np.pi * time_vec / period + 90000000)
)

print(np.round(sig, 2))
```

```
[-0.67 -0.78  0.03  0.41  0.43  0.55  0.25  0.6   0.08  0.35 -0.25 -0.59
 -0.61 -0.57 -0.69 -1.08 -0.65 -0.3  -0.66 -0.4  -0.02  0.61  0.74  1.55
  1.19  1.26  0.85  1.01  0.61  0.03  0.41 -0.15 -0.31 -0.93 -0.96 -1.14
 -0.99 -0.76 -0.52 -0.11 -0.02 -0.02  0.55  0.38  0.61  0.42  0.73  0.65
  0.37 -0.68 -0.83 -0.91 -1.8  -1.8  -2.01 -1.97 -1.66 -2.07 -1.62 -1.22
 -0.98 -0.93 -1.11 -0.32 -0.3  -0.76 -0.94 -0.13 -0.65 -1.62 -1.61 -2.56
 -1.8  -2.35 -2.5  -2.14 -2.79 -2.22 -1.9  -1.88 -1.71 -1.59 -1.23 -0.87
 -0.84 -0.48 -0.78 -0.36 -0.57 -0.9  -1.16 -1.45 -1.39 -1.57 -1.66 -2.39
 -2.21 -1.93 -0.95 -1.32 -0.71 -0.38 -0.04  0.41  0.6   0.42  0.61  1.03
  0.35  0.32 -0.25  0.16 -0.66 -0.59 -0.6  -0.74 -0.9  -0.21 -0.66 -0.3
  0.58  0.47  0.52  1.22  1.23  1.39  0.77  1.24  0.97  0.4   0.27 -0.3
```

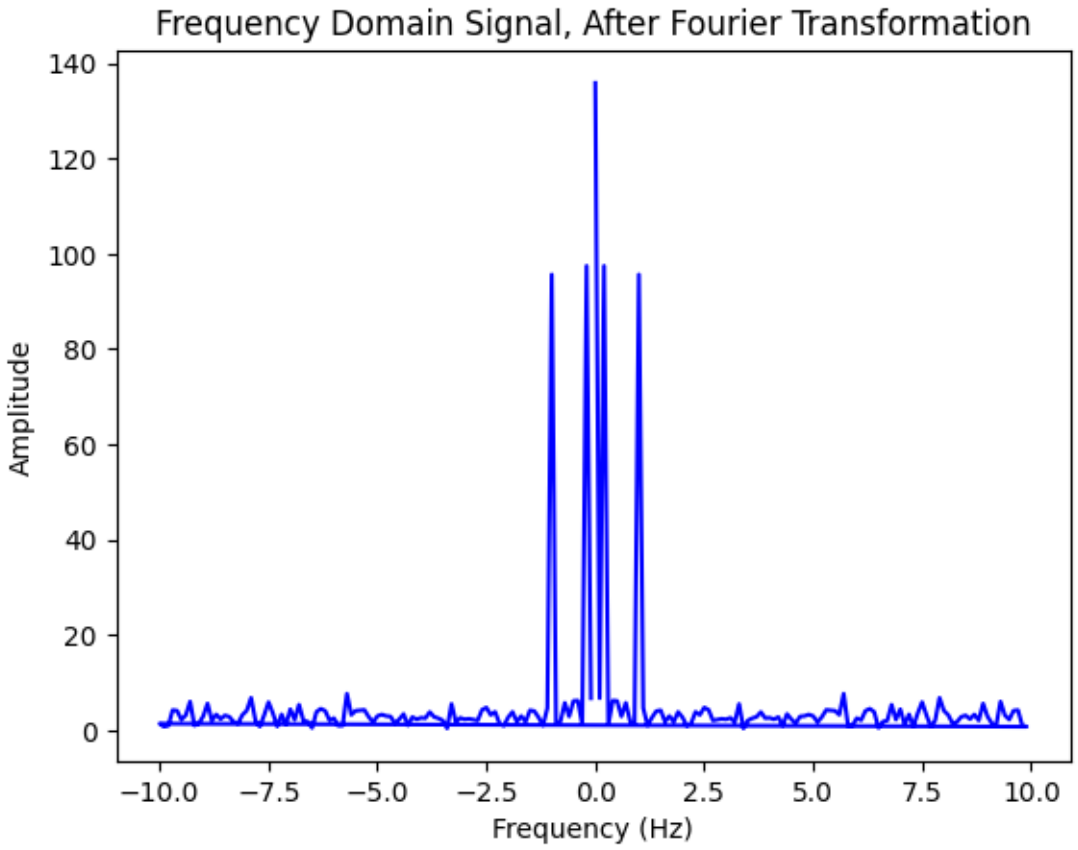
```
-0.31 -0.42 -0.87 -1.   -0.55 -1.22 -0.49 -0.57 -0.21  0.2   0.57  0.7
 0.25  0.52  0.52  0.26  0.03 -0.2  -0.98 -1.24 -0.83 -1.2  -2.32 -2.06
-1.97 -1.74 -1.92 -1.33 -1.49 -1.23 -0.63 -0.27 -0.52 -0.51 -0.54 -1.41
-1.54 -1.24 -1.39 -2.14 -2.18 -2.36 -2.32 -2.71 -2.89 -2.76 -2.57 -1.68
-1.66 -1.4  -0.77 -0.68 -0.94 -0.58 -0.62 -0.5  -0.6  -1.39 -1.71 -1.6
-1.73 -2.11 -2.39 -2.31 -1.96 -1.46 -1.4  -1.16]
```

[11]:
```
plt.plot(time_vec, sig)
plt.xlabel("Time (s)")
plt.ylabel("Voltage (micro volts)")
plt.title("Time Domain Signal")
plt.show()
```



[12]:
```
# Taking the fourier transform of the signal
sig_fft = fftpack.fft(sig)

# Finding the ampltiudes, powers and angles
Amplitude = np.abs(sig_fft)
Power = Amplitude**2
Angle = np.angle(sig_fft)
```

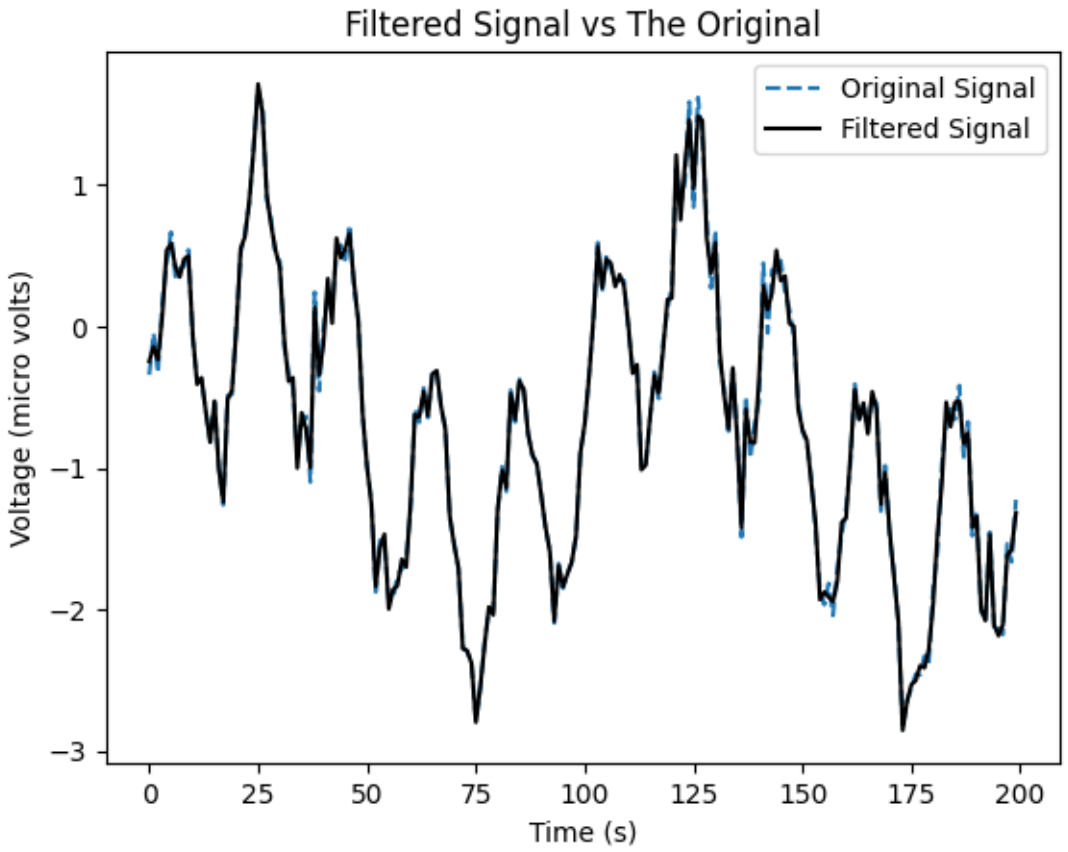
```
# Finding the sample frequency and the peak frequency
sample_freq = fftpack.fftfreq(sig.size, d=time_step)
Amp_freq = np.array([Amplitude, sample_freq])
peak_frequency_list = np.sort(Amp_freq[1])[-10:]

plt.plot(sample_freq, Amplitude, label="Fourier Transform", color="blue")
plt.xlabel("Frequency (Hz)")
plt.ylabel("Amplitude")
plt.title("Frequency Domain Signal, After Fourier Transformation")
plt.show()
```



[13]:
```
# Filtering the signal
high_freq_fft = sig_fft.copy()
high_freq_fft[np.abs(sample_freq) > np.min(peak_frequency_list)] = 0
filtered_sig = fftpack.ifft(high_freq_fft)

plt.plot(sig, label="Original Signal", linestyle="--")
plt.plot(filtered_sig, label="Filtered Signal", color="black")
```

```
plt.legend()
plt.xlabel("Time (s)")
plt.ylabel("Voltage (micro volts)")
plt.title("Filtered Signal vs The Original")
plt.show()
```

```
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-
packages/matplotlib/cbook.py:1699: ComplexWarning: Casting complex values to
real discards the imaginary part
  return math.isfinite(val)
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-
packages/matplotlib/cbook.py:1345: ComplexWarning: Casting complex values to
real discards the imaginary part
  return np.asarray(x, float)
```



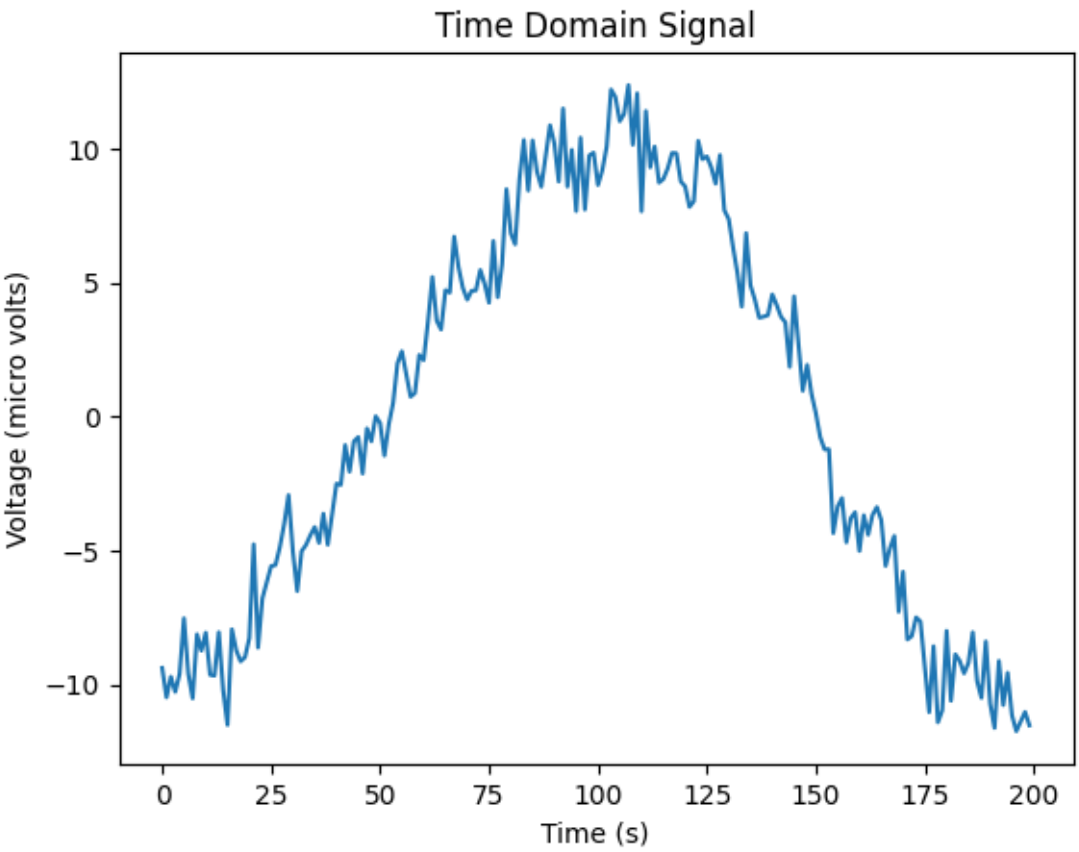
6 Making a General Function

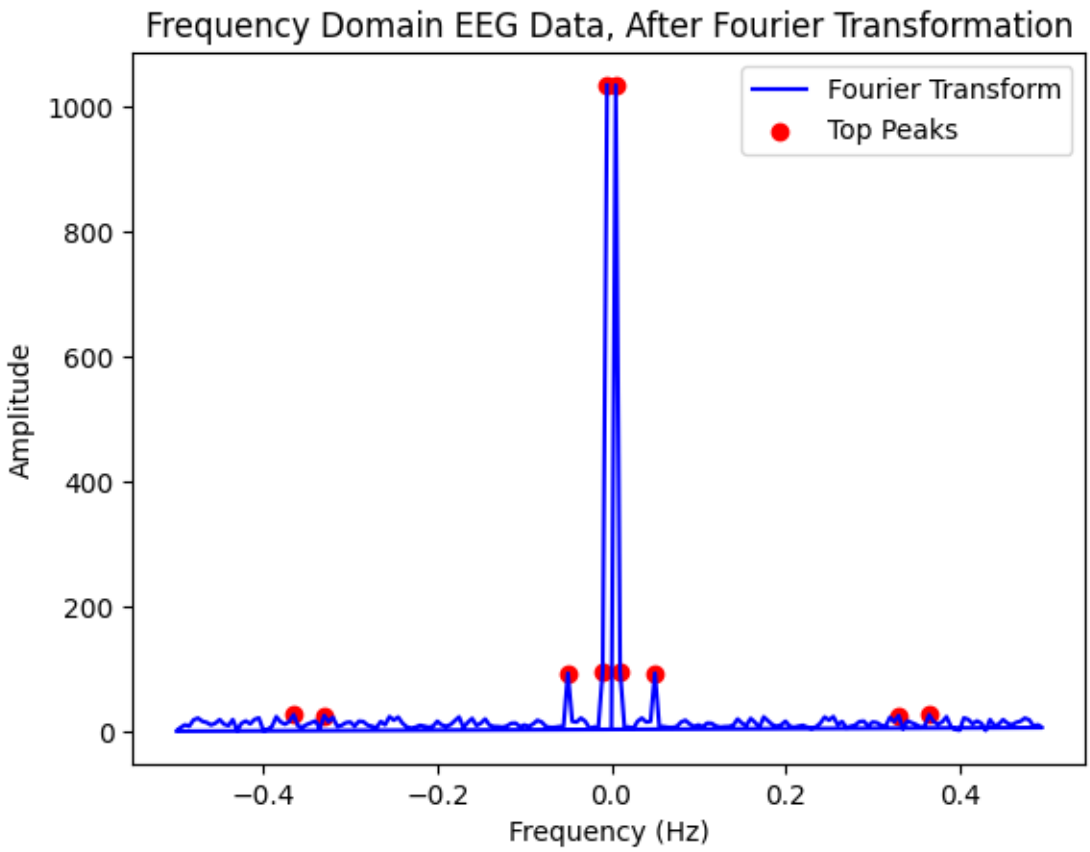
[14]:
```
# Assumption:   - The eeg_data is a 1D numpy array
#               - The eeg_data is a time series data
#               - The eeg_data comprises of sums of sine or cosine waves (with␣
 ↪a little bit of noise)

# This function takes in an EEG data and plots the original signal, calculates␣
 ↪the fourier transform and plots the
# filtered signal.

# fourier_transform_eeg(eeg_data :: np.array, level_of_detail :: int):
#                           np.array, int -> np.array(2, level_of_detail)

def fourier_transform_eeg(eeg_data, level_of_detail=10):
    # Input validation
    if not isinstance(eeg_data, np.ndarray) or eeg_data.ndim != 1:
        raise ValueError("Input 'eeg_data' must be a 1D NumPy array.")

    # Plotting the time domain signal of the Original EEG data
    plt.plot(eeg_data)
    plt.xlabel("Time (s)")
    plt.ylabel("Voltage (micro volts)")
    plt.title("Time Domain Signal")
    plt.show()

    # Taking the Fourier transform of the signal
    eeg_data_fft = fftpack.fft(eeg_data)

    # Finding the amplitudes and frequencies
    amplitudes = np.abs(eeg_data_fft)
    sample_freq = fftpack.fftfreq(eeg_data.size)

    # Finding the top peaks in the frequency domain
    peak_indices = np.argsort(amplitudes)[::-1][:level_of_detail]
    output_amplitude_and_frequency = np.array([amplitudes[peak_indices],␣
 ↪sample_freq[peak_indices]])


    # Plotting the Fourier Transform
    plt.plot(sample_freq, amplitudes, label="Fourier Transform", color="blue")
    plt.scatter(sample_freq[peak_indices], amplitudes[peak_indices],␣
 ↪color='red', label='Top Peaks')
    plt.xlabel("Frequency (Hz)")
    plt.ylabel("Amplitude")
    plt.title("Frequency Domain EEG Data, After Fourier Transformation")
    plt.legend()
```

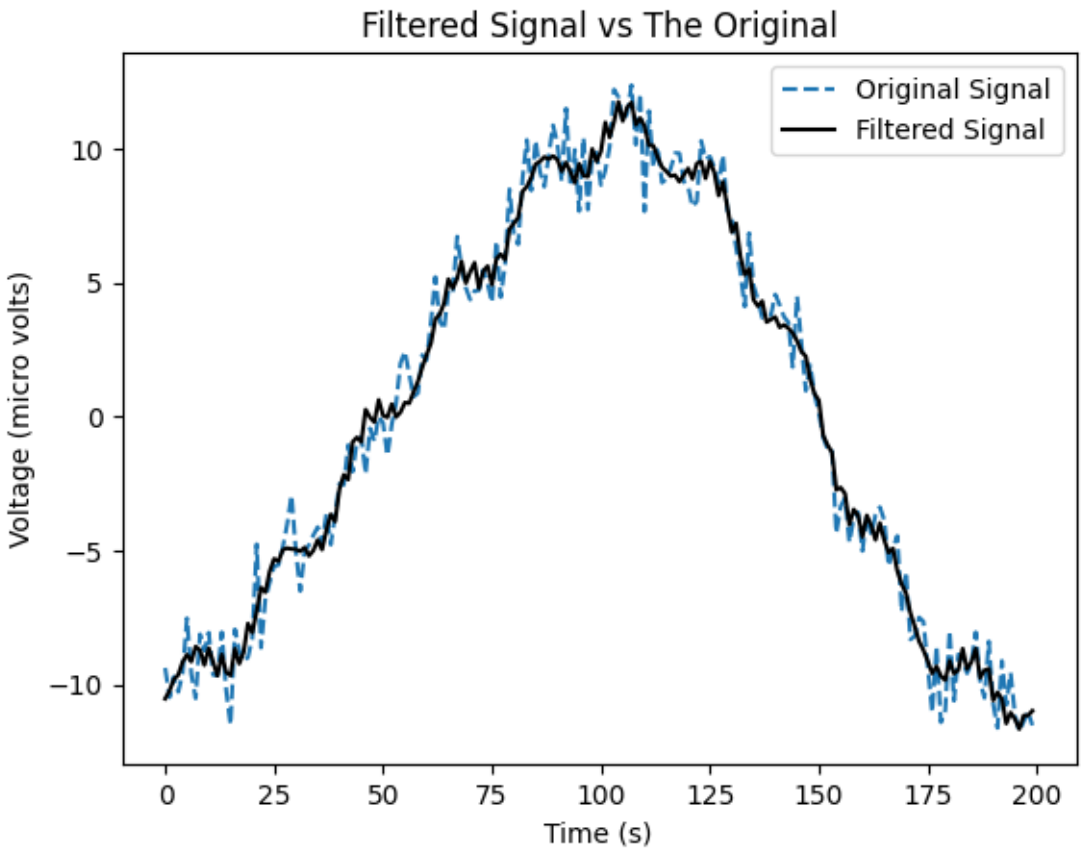
```
    plt.show()

    # Reconstructing the filtered signal using the inverse Fourier transform
    filtered_eeg_data = fftpack.ifft(np.where(np.isin(sample_freq,␣
 ↪output_amplitude_and_frequency[1]), eeg_data_fft, 0))

    # Plotting the filtered signal
    plt.plot(eeg_data, label="Original Signal", linestyle="--")
    plt.plot(filtered_eeg_data.real, label="Filtered Signal", color="black")
    plt.legend()
    plt.xlabel("Time (s)")
    plt.ylabel("Voltage (micro volts)")
    plt.title("Filtered Signal vs The Original")
    plt.show()

    return output_amplitude_and_frequency
```

7 Looking how it works in other models

[15]:
```
# It works for other models too!

sig = (
        np.sin(2 * np.pi * time_vec / period)
        + np.random.randn(time_vec.size)
        + np.sin(10 * np.pi * time_vec / period)
        + -10 * np.sin(np.pi * time_vec / period + np.pi / 2)
        + 0.5 * np.sin(10000 * np.pi * time_vec / period)
)

sig_fourier = fourier_transform_eeg(sig)
print("Transformed Sig: ", sig_fourier)
```


Time Domain Signal
Voltage (micro volts)
10
5
0
−5
−10
0
25
50
75
100
125
150
175
200
Time (s)


Frequency Domain EEG Data, After Fourier Transformation
Fourier Transform
Top Peaks
Amplitude
1000
800
600
400
200
0
−0.4
−0.2
0.0
0.2
0.4
Frequency (Hz)

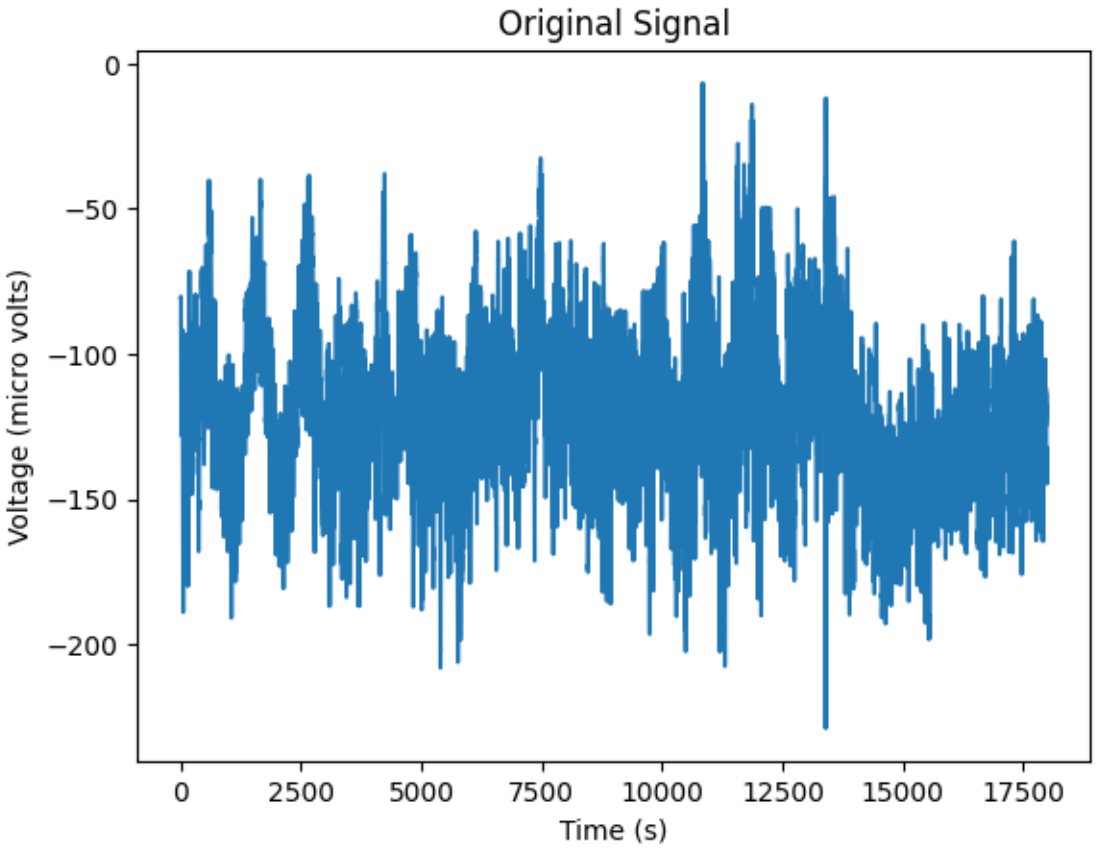


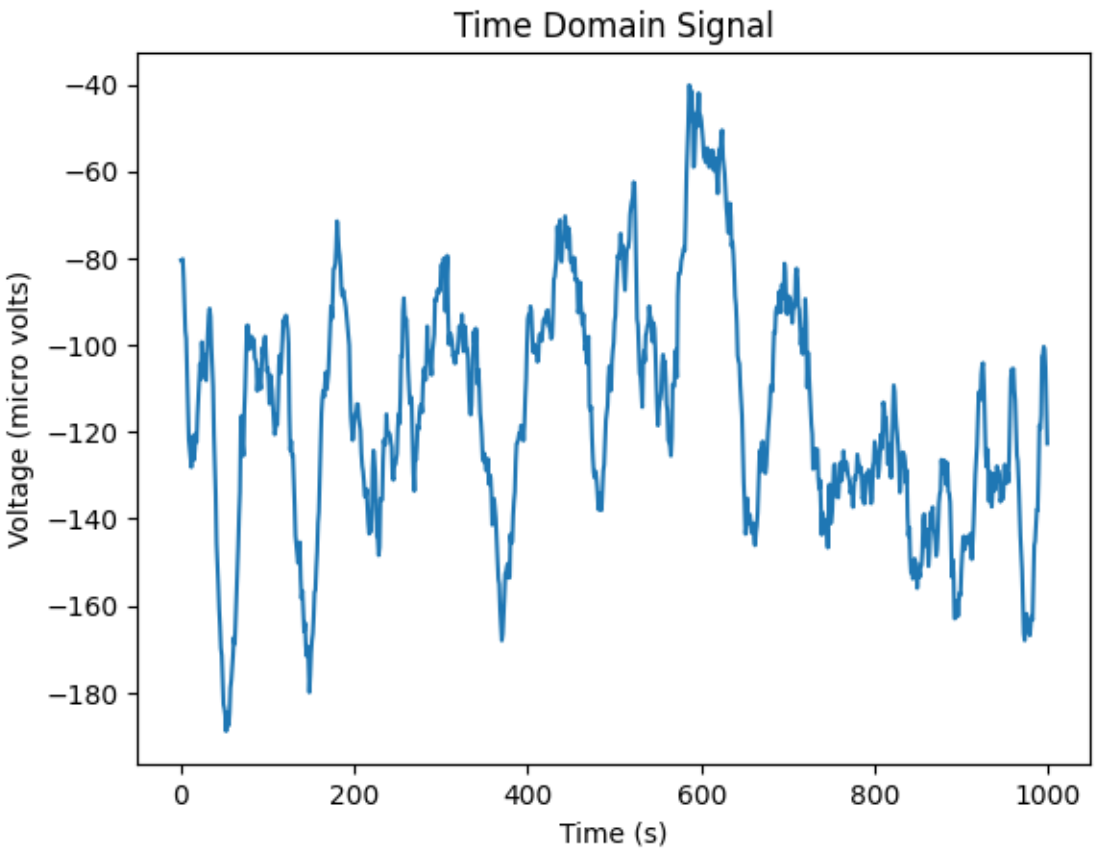

```
Transformed Sig:  [[ 1.01379981e+03  1.01379981e+03  1.21103295e+02
1.21103295e+02
   1.03609664e+02  1.03609664e+02  2.99204286e+01  2.99204286e+01
   2.95183434e+01  2.95183434e+01]
 [-5.00000000e-03  5.00000000e-03 -5.00000000e-02  5.00000000e-02
  -1.00000000e-02  1.00000000e-02 -3.50000000e-01  3.50000000e-01
  -2.15000000e-01  2.15000000e-01]]
```

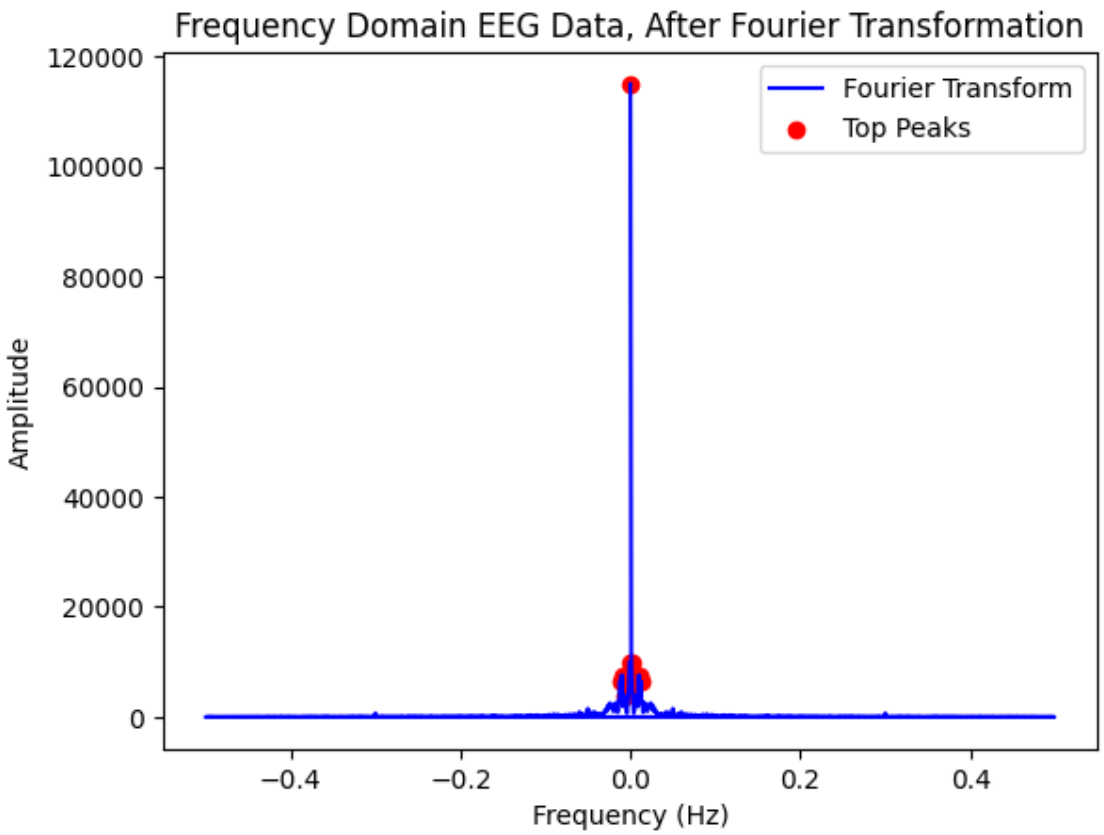
8 Applying Fourier Transforms to our EEG Samples

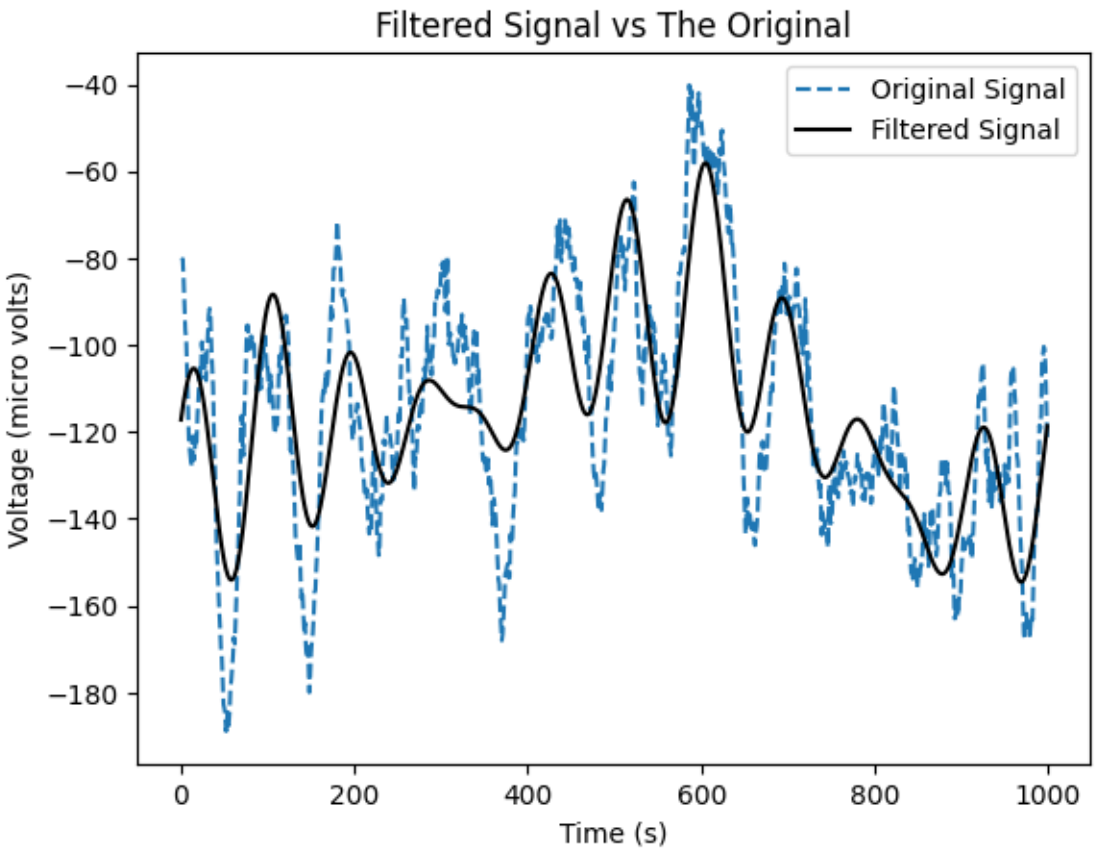
[16]:
```
eeg_visualization_Fp1 = np.array(eeg_visualization.loc[:,"Fp1"])
plt.plot(eeg_visualization_Fp1, label="Original Signal")
plt.xlabel("Time (s)")
plt.ylabel("Voltage (micro volts)")
plt.title("Original Signal")
plt.show()

# Let's take smaller portion of the signal
eeg_visualization_Fp1 = eeg_visualization_Fp1[0:1000]
```

```
eeg_fourier = fourier_transform_eeg(eeg_visualization_Fp1)
print("Transformed EEG: ", eeg_fourier)
```




Time Domain Signal
Voltage (micro volts)
−40
−60
−80
−100
−120
−140
−160
−180
0
200
400
600
800
1000
Time (s)

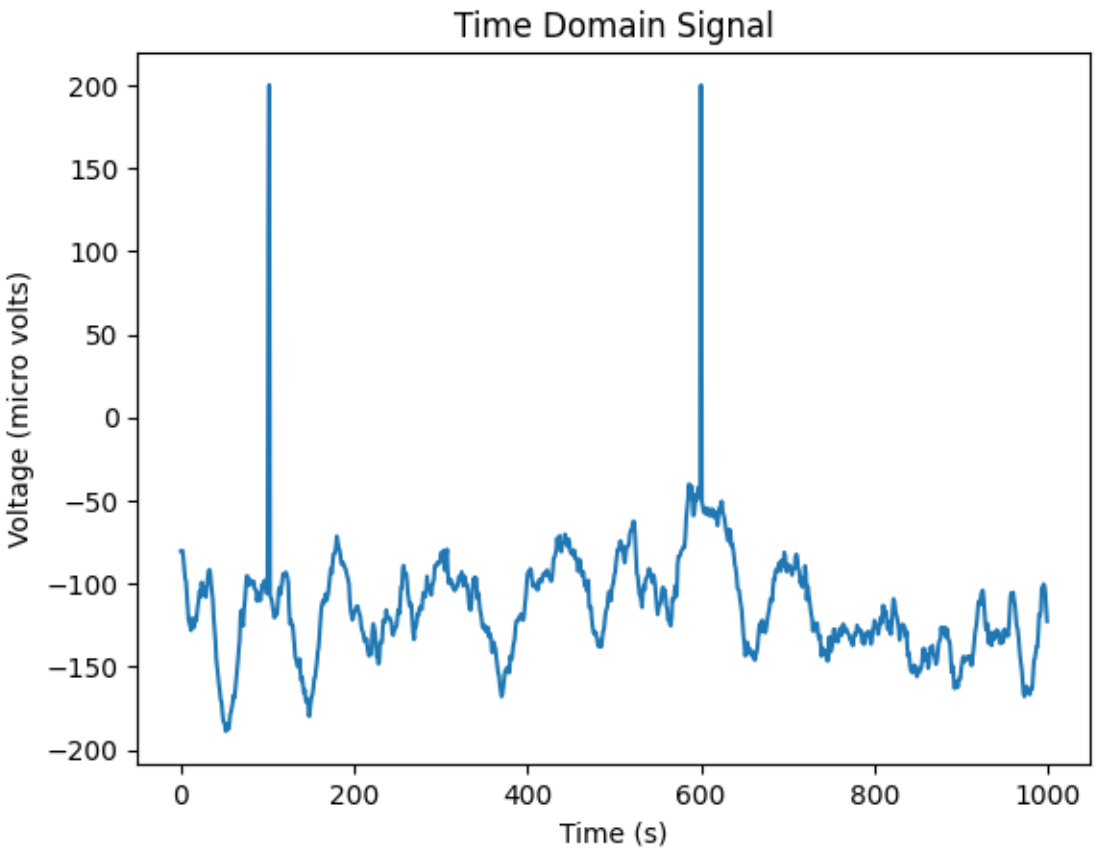

Frequency Domain EEG Data, After Fourier Transformation
Fourier Transform
Top Peaks
Amplitude
120000
100000
80000
60000
40000
20000
0
−0.4
−0.2
0.0
0.2
0.4
Frequency (Hz)

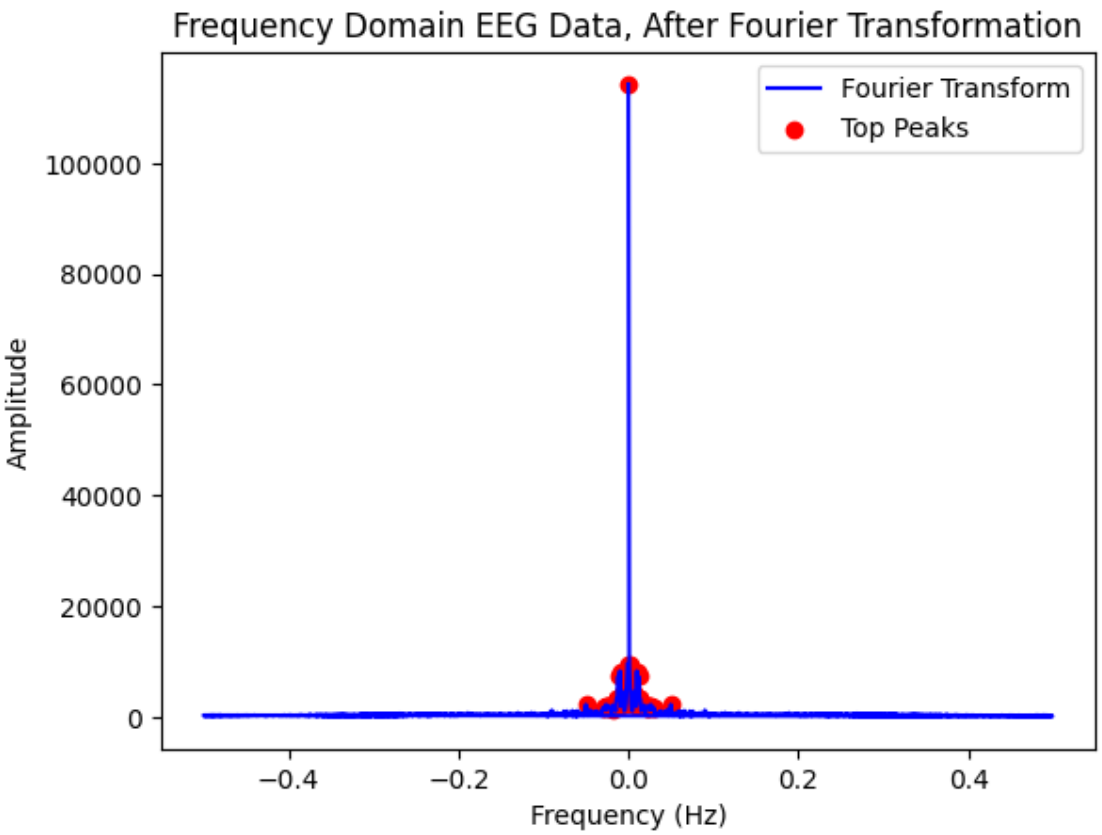


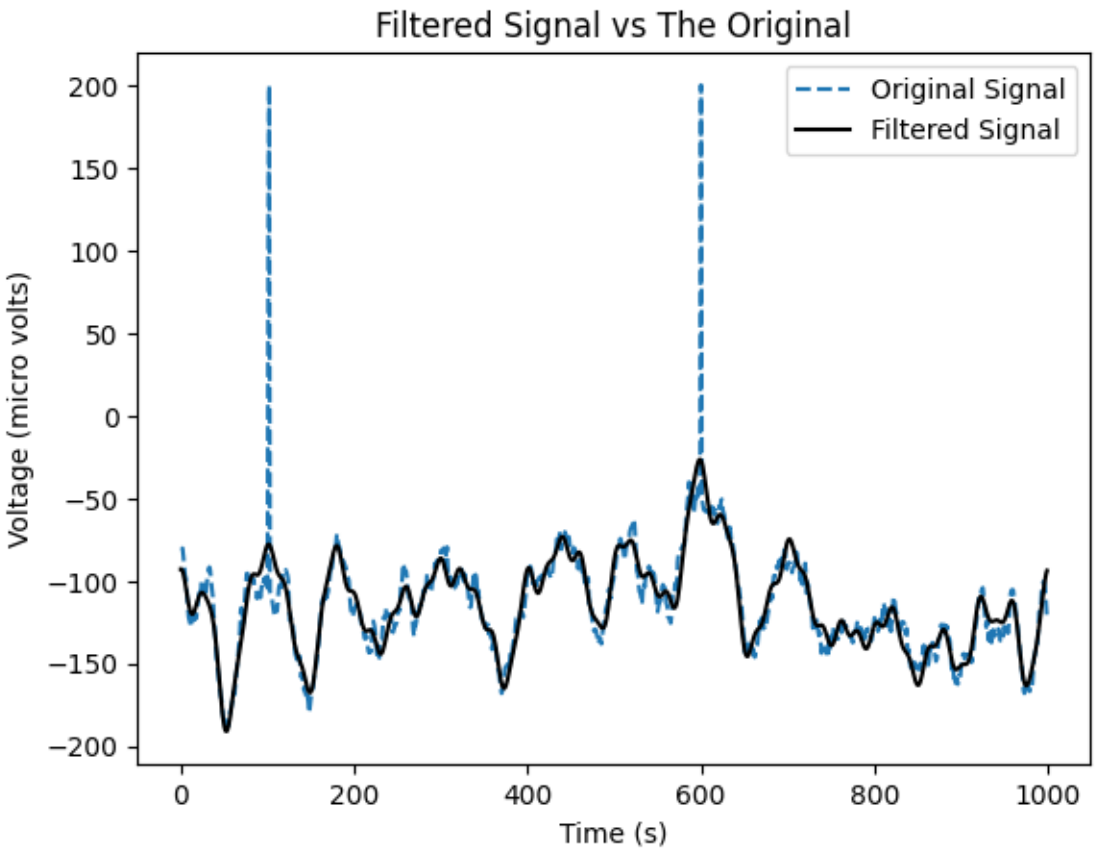

```
Transformed EEG:  [[ 1.15003258e+05  9.77734961e+03  9.77734961e+03
7.62491211e+03
   7.62491211e+03  6.55427344e+03  6.55427344e+03  5.27791992e+03
   5.27791992e+03  3.89158447e+03]
 [ 0.00000000e+00  1.00000000e-03 -1.00000000e-03 -1.00000000e-02
   1.00000000e-02 -1.20000000e-02  1.20000000e-02  2.00000000e-03
  -2.00000000e-03 -6.00000000e-03]]
```

9 Adding Artifacts

[17]:
```
# Adding artifcats
eeg_visualization_Fp1[101] = 100
eeg_visualization_Fp1[102] = 200
eeg_visualization_Fp1[600] = 200
eeg_2_fourier = fourier_transform_eeg(eeg_visualization_Fp1, 50)

print("Transformed EEG 2: ", eeg_2_fourier)
```


Time Domain Signal
Voltage (micro volts)
200
150
100
50
0
−50
−100
−150
−200
0
200
400
600
800
1000
Time (s)


Frequency Domain EEG Data, After Fourier Transformation
Fourier Transform
Top Peaks
Amplitude
100000
80000
60000
40000
20000
0
−0.4
−0.2
0.0
0.2
0.4
Frequency (Hz)

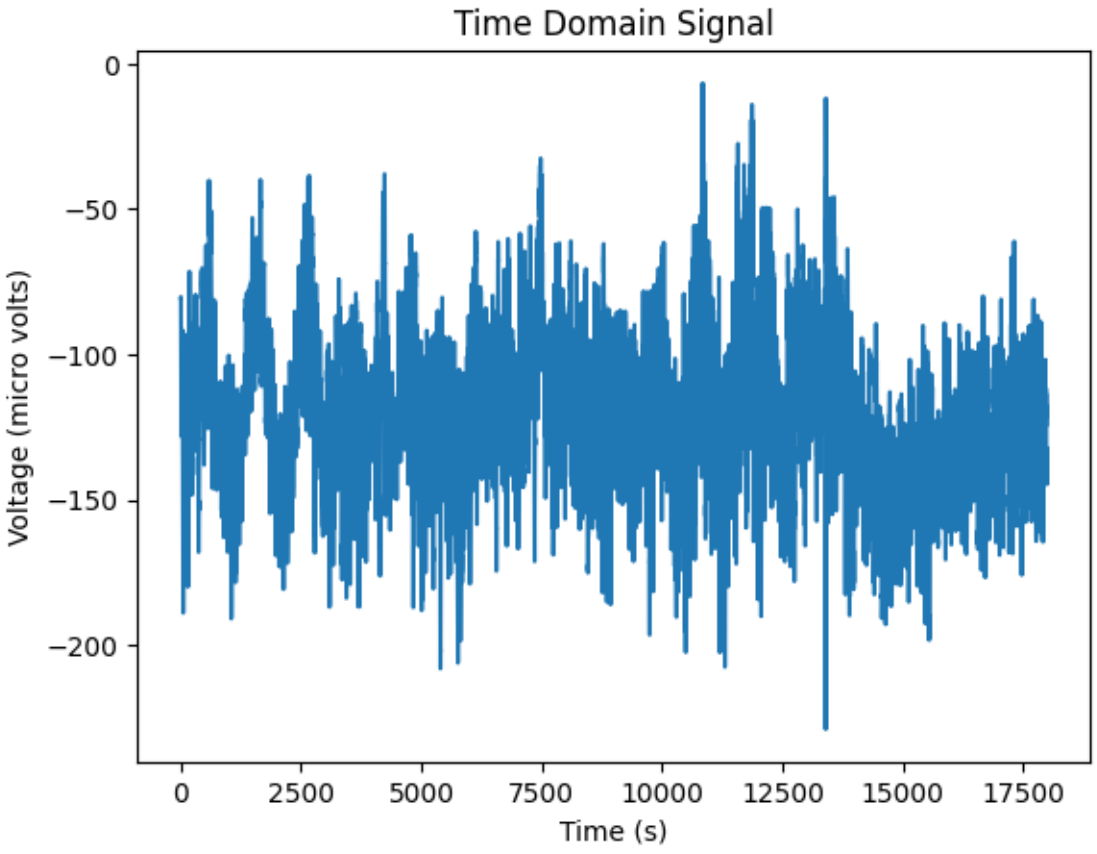


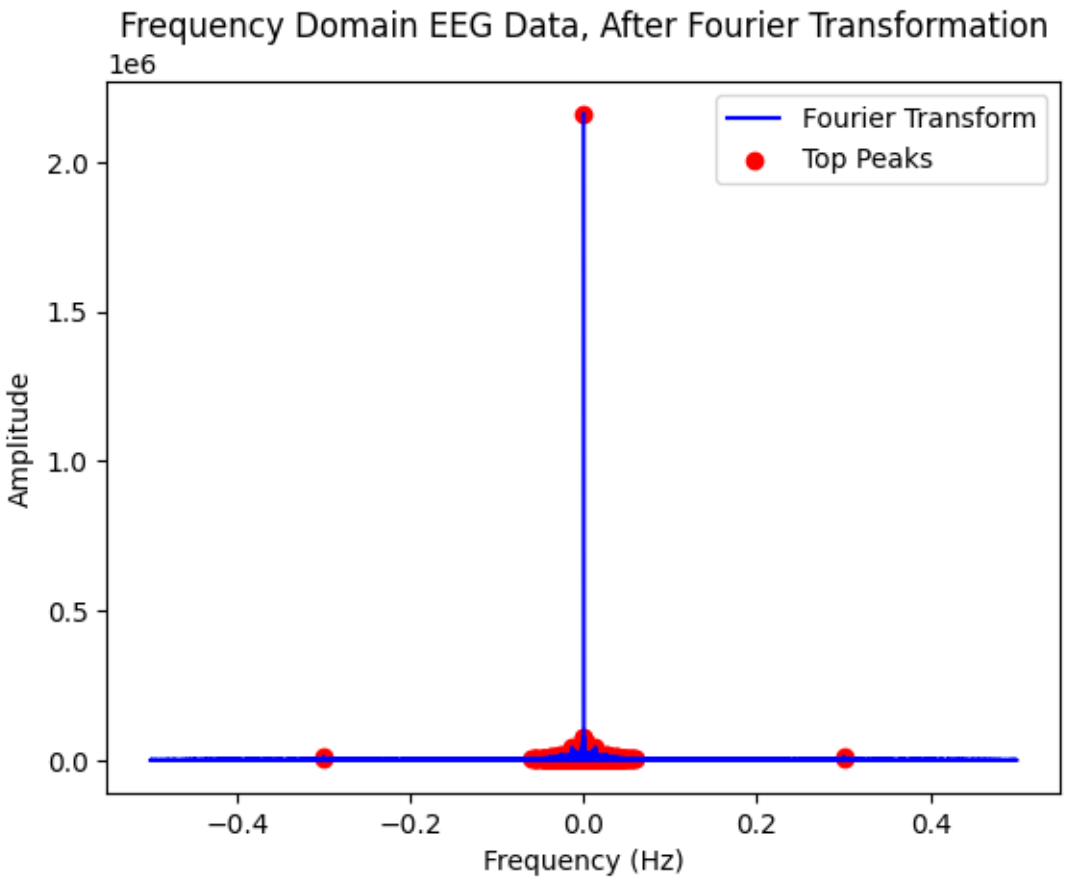

```
Transformed EEG 2:  [[ 1.14233188e+05  9.56493457e+03  9.56493457e+03
8.31597852e+03
   8.31597852e+03  7.32023584e+03  7.32023584e+03  6.04667676e+03
   6.04667676e+03  4.56502148e+03  4.56502148e+03  3.66887744e+03
   3.66887744e+03  3.63344141e+03  3.63344141e+03  3.35441602e+03
   3.35441602e+03  3.32546289e+03  3.32546289e+03  2.99065552e+03
   2.99065552e+03  2.51312842e+03  2.51312842e+03  2.25112231e+03
   2.25112231e+03  2.19697559e+03  2.19697559e+03  2.16395703e+03
   2.16395703e+03  2.13425903e+03  2.13425903e+03  2.10681226e+03
   2.10681226e+03  1.97249512e+03  1.97249512e+03  1.93759692e+03
   1.93759692e+03  1.89755737e+03  1.89755737e+03  1.70578357e+03
   1.70578357e+03  1.68249646e+03  1.68249646e+03  1.66652734e+03
   1.66652734e+03  1.61454211e+03  1.61454211e+03  1.59433215e+03
   1.59433215e+03  1.41623743e+03]
 [ 0.00000000e+00  1.00000000e-03 -1.00000000e-03  1.00000000e-02
  -1.00000000e-02  1.20000000e-02 -1.20000000e-02  2.00000000e-03
  -2.00000000e-03 -6.00000000e-03  6.00000000e-03 -8.00000000e-03
   8.00000000e-03 -7.00000000e-03  7.00000000e-03 -1.30000000e-02
   1.30000000e-02  1.40000000e-02 -1.40000000e-02  1.10000000e-02
  -1.10000000e-02  3.00000000e-03 -3.00000000e-03  5.00000000e-02
```

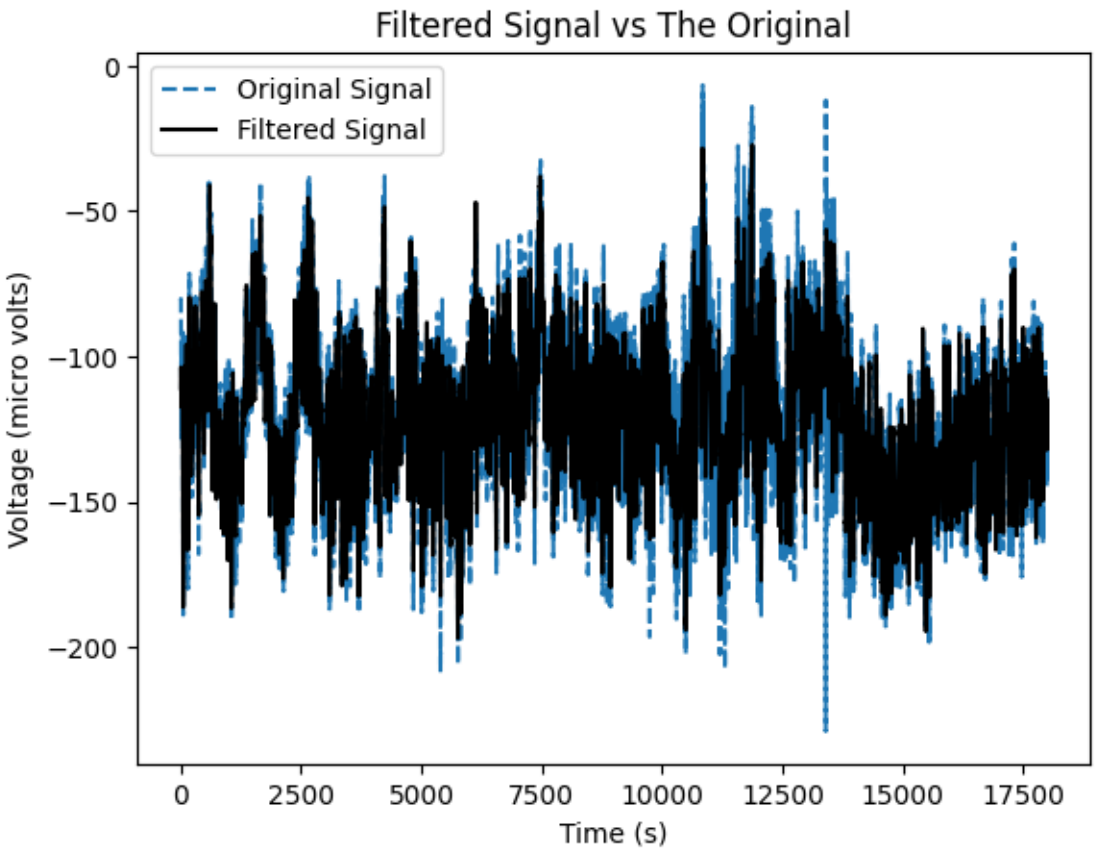
```
-5.00000000e-02  2.70000000e-02 -2.70000000e-02 -2.50000000e-02
 2.50000000e-02 -1.90000000e-02  1.90000000e-02  1.70000000e-02
-1.70000000e-02  2.40000000e-02 -2.40000000e-02 -9.00000000e-03
 9.00000000e-03  1.60000000e-02 -1.60000000e-02 -2.30000000e-02
 2.30000000e-02 -2.10000000e-02  2.10000000e-02 -3.00000000e-02
 3.00000000e-02 -2.20000000e-02  2.20000000e-02  2.80000000e-02
-2.80000000e-02 -1.80000000e-02]]
```

10 Looking at the bigger picture

[18]:
```
# Looking at the bigger picture
eeg_3_fourier = fourier_transform_eeg(np.array(eeg_visualization.loc[:,"Fp1"]),␣
↪1000)

print("Transformed EEG 3: ", eeg_3_fourier)
```




Frequency Domain EEG Data, After Fourier Transformation
1e6
Fourier Transform
Top Peaks
2.0
1.5
1.0
0.5
0.0
Amplitude
−0.4
−0.2
0.0
0.2
0.4
Frequency (Hz)




```
Transformed EEG 3:  [[ 2.15990050e+06  7.84746641e+04  7.84746641e+04 …
4.97831201e+03
   4.97831201e+03  4.97555273e+03]
 [ 0.00000000e+00 -1.66666667e-04  1.66666667e-04 …  4.72222222e-03
  -4.72222222e-03 -2.50555556e-02]]
```

11 Who gives a damn?

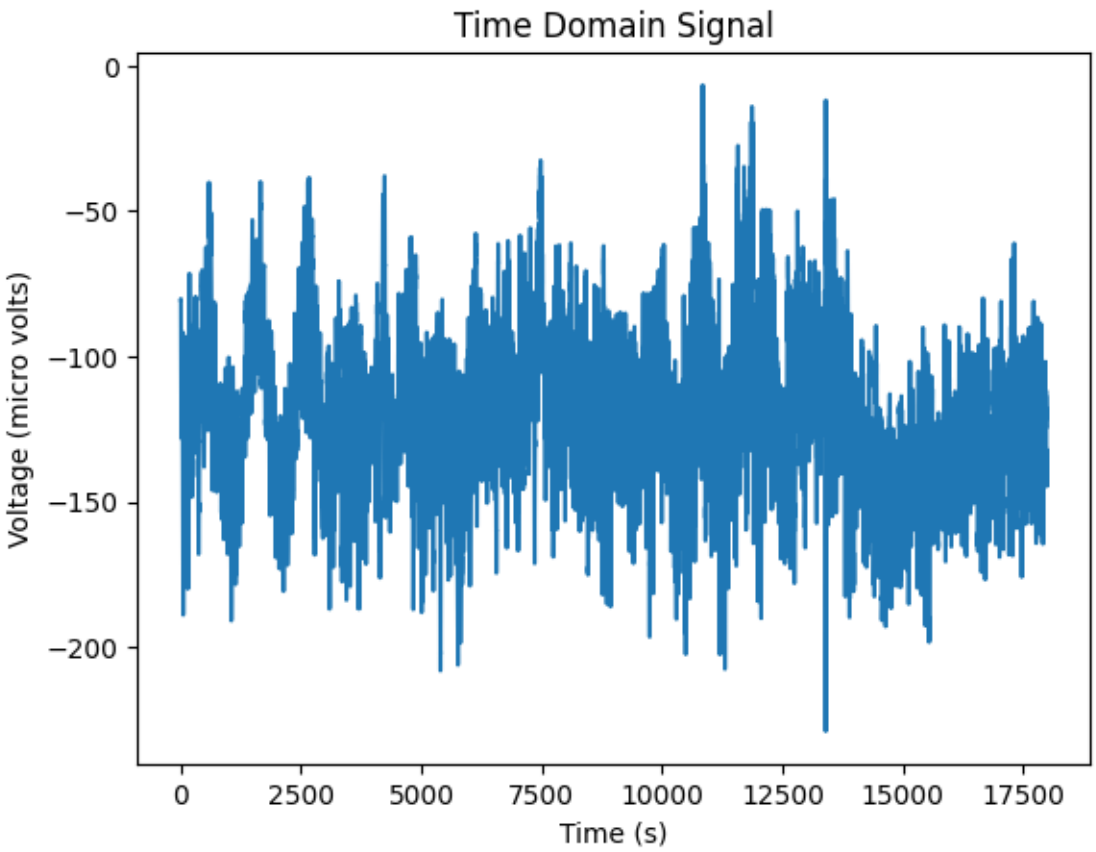
Instead of dealing with a size of

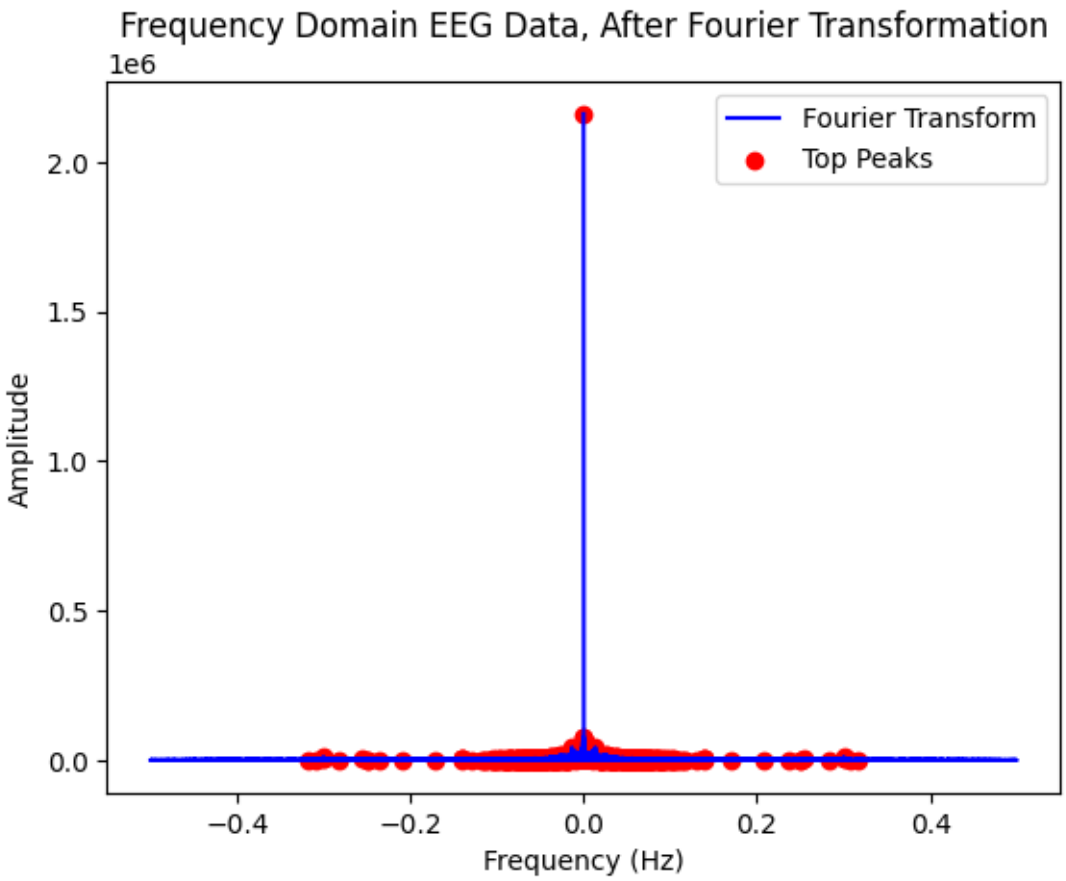
[19]:
```
print(eeg_visualization.loc[:,"Fp1"].size)
```

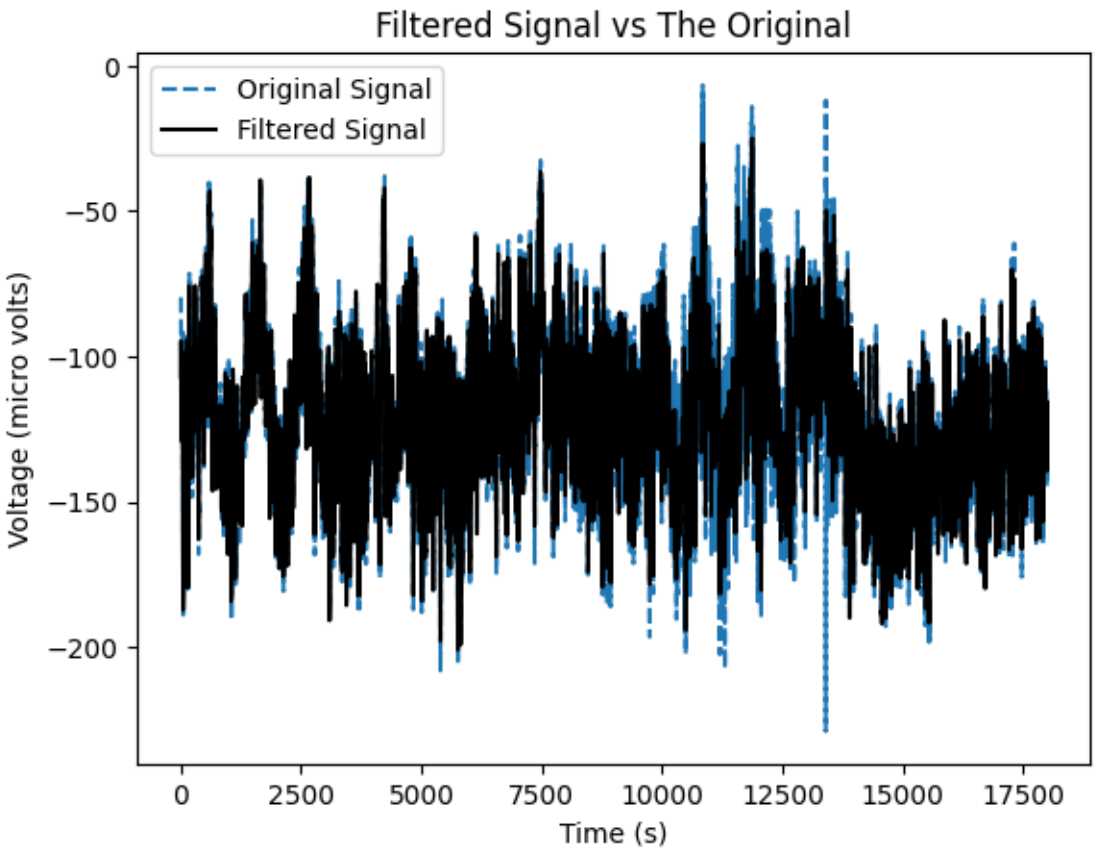
```
18000
```

We can deal with a smaller size of

[20]:
```
print(fourier_transform_eeg(np.array(eeg_visualization.loc[:,"Fp1"]), 2000).
↪size)
```


Time Domain Signal
0
−50
−100
−150
−200
Voltage (micro volts)
0
2500
5000
7500
10000
12500
15000
17500
Time (s)


Frequency Domain EEG Data, After Fourier Transformation
1e6
Fourier Transform
Top Peaks
2.0
1.5
1.0
0.5
0.0
Amplitude
−0.4
−0.2
0.0
0.2
0.4
Frequency (Hz)



4000

We have also dealt with Data Standardization and Data Imputation using fourier series

Plus, now we have the complete information only using the frequencies and ampltidues in sin waves.