Data cleaning process

The data collected using the OpenBCI device is completely raw. This means that there are no filters applied on the EEG data that is being stored in the .txt file. However the data displayed in the GUI in the form of images, graphs and animations during the data collection experiment was filtered. Some research in the context of GUI proved that the data being rendered in the GUI is passed through filtering and cleaning programs in the backend before visualizing it on our desktop screens. Thus there emerged a need to find a process that can clean the data as closely as the cleaned data in the GUI. One of my assumption was that "if we follow the same code that was used to visualize the graphs in the GUI, we can certainly get our data cleaned". Further research pointed out towards a package called as 'brainflow' that included all the necessary tools to clean and pre-process our data. This package contains APIs and functions to collect data from various BCI devices, pre-processing functionalities, visualizing data using plots and graphs and also consists of machine learning algorithms to help build models. So using this package was the best choice. But the package was laid out in a more general way and hence the packages for filtering and de-noising were available in a single file/ code snippet. Thus there were ambiguities regarding which to apply first, filtering or de-noising. And also there are several filtering and de-noising options available that can be applied on the data. This added up several other questions such as 'which filter to apply and with what parameters?', 'which de-noising techniques are best for our project?' etc.

Thorough research in finding answers to the questions and referring to several discussions in the OpenBCI forum helped me find a workflow to apply pre-processing steps. The required workflow was:

1. Apply bandpass filter (1-51Hz, 3rd order)
2. Apply bandstop filter (59.5-60.5Hz, 4th order)
3. Denoising
4. Convert voltage values to RMS voltage values.

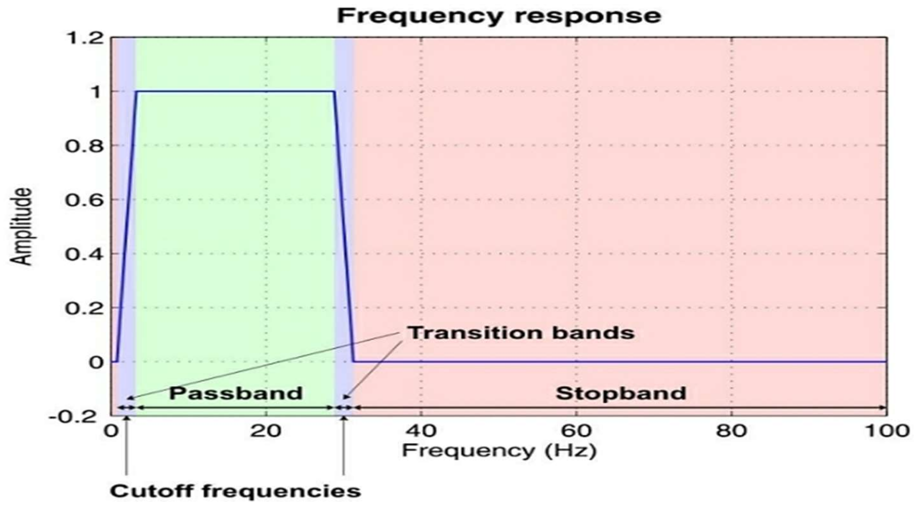
After getting the workflow, the next step was to select the best filters and methods for the project. Researching again in the field of filters resulted out finding a resource that contained experiment which was very closely related to ours. The webpage blog was titled as EEG basics and was residing in the website of CiMeC (Centre for Mind/Brain Sciences). I have noted some of the points that were helpful to me:

1. **What is sampling rate and how much sampling rate is considered effective for our project?**
    - Sampling rate (in Hz) is defined as the number of samples (or time-points) acquired in one second.
    - In EEG data each time-point represents a voltage sample (usually measured in µV).
    - Sampling rate is important because the recorded discrete samples will be used to represent the continuous voltages. And to support this statement sampling theory states that "A continuous signal can be accurately reconstructed from its samples, if the highest frequency present in the continuous signal is lower than half the sampling rate". This value of half the sampling rate is commonly known as the 'Nyquist Frequency'.
    - Typical sampling rate values ranges from 250Hz (i.e. 1 sample every 4ms) to 2000Hz (i.e. 1 sample every 0.5ms). Higher the sampling rate better the resolution of signals temporal dynamics.
    - Higher sampling rate will bring problems related to data storage and processing. Lower sampling rate will typically have the worst resolution and also non-existent signals can be introduced. Thus choosing the correct sampling rate will ensure the quality of the data.
    - It was also noted that "if the goal of our research is to characterize ERP components then sampling rate of 250Hz is more adequate. If the goal is to characterize high frequency oscillatory dynamics then values of 1000-2000Hz are desirable". This concludes that sampling rate of 250Hz is sufficient for our project, and the data collection experiment perfectly maintained this criteria.
2. **What is downsampling and is it necessary for our project?**
    - As we know that high sampling rate induces burden on processing speed and time, hence the EEG data is downsampled from high sampling rate to a lower sampling rate which is easier to process.

- Down sampling aggregates two or more consecutive rows of timestamps to reduce the number of samples per second, which in turn reduces the number of rows of the dataset.
- For our project, as concluded earlier, sampling rate of 250Hz is good and hence no downsampling is required.

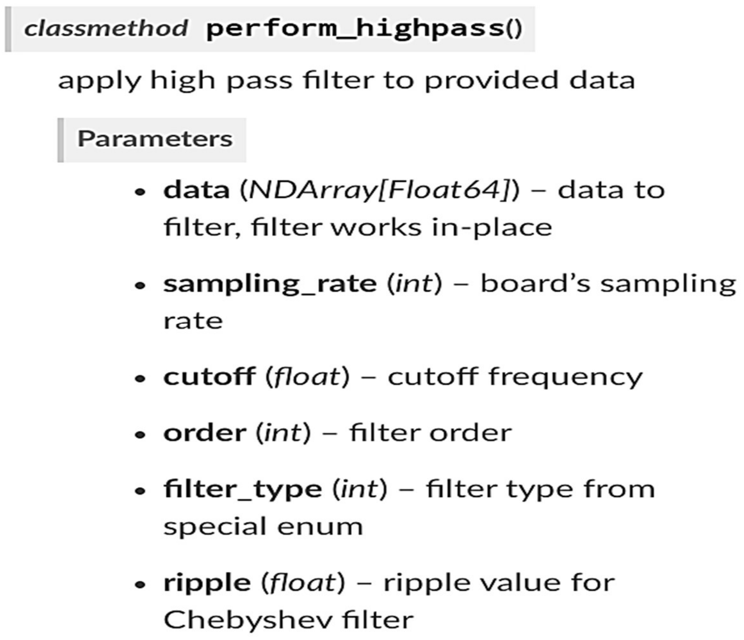
3. **Why filtering is necessary?**
    - Filtering is typically done to improve the signal to noise ratio and reduce signal distortions by attenuating those frequencies that are thought to be noisy (examples: low frequency skin potentials, high frequency EMG activities, line noise at 50/60 Hz).
    - Each filters add up some unintended adverse effects (signal distortions) so the parameters need to be set according to the application. So selection of filters is essential and equally important is selection of parameters for filters.
    - The figure below shows frequency response of a band-pass filter, but every filter has similar characteristics with respect to frequency response:

Figure 2.3: Frequency response of a band-pass filter



    - The cutoff frequencies are the frequencies that we set to filter.
    - But no filter actually is accurate enough to attenuate frequencies just outside of the cutoff frequencies. Hence the transition bands is the actual range of filter frequencies. Hence to get the perfect signal filtering we need to keep in mind the transition bands and cutoff frequencies and apply the filters accordingly (i.e. the cutoff frequencies should be such that it should also maintain the transition bands for filters). In our case I kept the frequency at 0.3Hz as the transition band may restrict to 0.1Hz which is the least frequency of recorded EEG signals.
    - Lastly the stopband is the frequency limit that will not be exceeded by the transition bands in any case.
    - According to above workflow, bandpass filter (1-51Hz 3rd order) needs to be applied. A bandpass filter allows frequencies in a certain range to pass through it and attenuates frequencies above and below the specified range.
    - But instead of using bandpass filter it is recommended to use lowpass and highpass filter as the combination of HP and LP filters is effective than the BP filter.
    - It is generally recommended for a HP filter to use lower cutoff frequency than 0.5Hz. So cutoff frequency of 0.3Hz is effective and also the transition bands of such filter will lie in the range of 0.1 to 0.5Hz.
    - The notch filters are stop band filters with a very narrow band, which are sometimes used to remove line noise. The application of notch filters is not recommended because they may introduce dramatic distortions of the original signal in frequency bands that are well outside the notch. Hence I rejected step 2 from the workflow and verified that the signal doesn’t have distortions as mentioned.
4. **What are the different parameters that are set for filters?**

- I followed and used brainflow APIs for filtering. These APIs consisted of filters that had a certain syntax and a set of parameters that needs to be set in order to apply the filters.
- The syntax for HP filter as mentioned in the docs is:



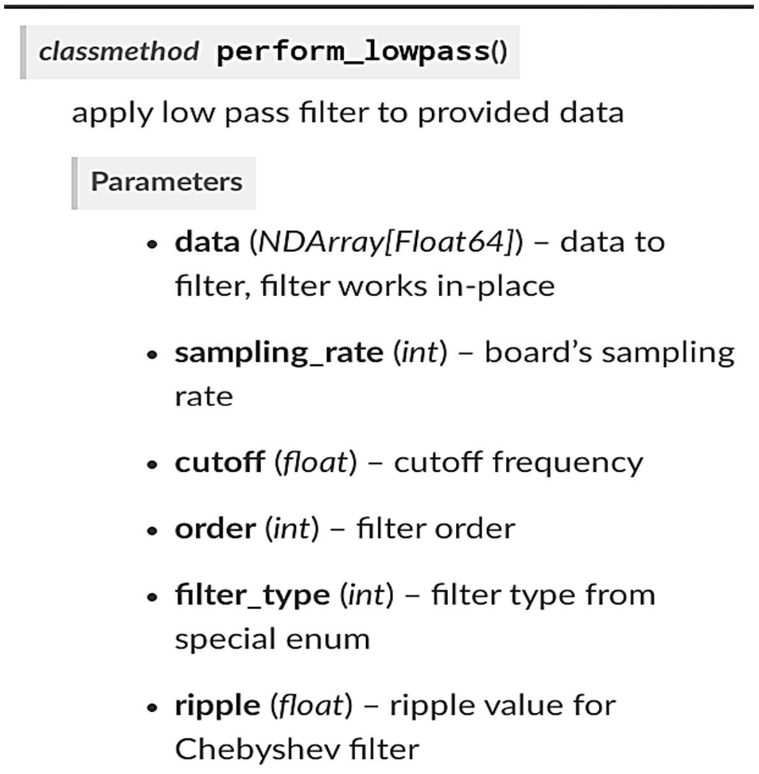
classmethod **perform_highpass()**

apply high pass filter to provided data

Parameters

- **data** (*NDArray[Float64]*) – data to filter, filter works in-place
- **sampling_rate** (*int*) – board's sampling rate
- **cutoff** (*float*) – cutoff frequency
- **order** (*int*) – filter order
- **filter_type** (*int*) – filter type from special enum
- **ripple** (*float*) – ripple value for Chebyshev filter

- The syntax for LP filter as mentioned in the docs is:



classmethod **perform_lowpass()**

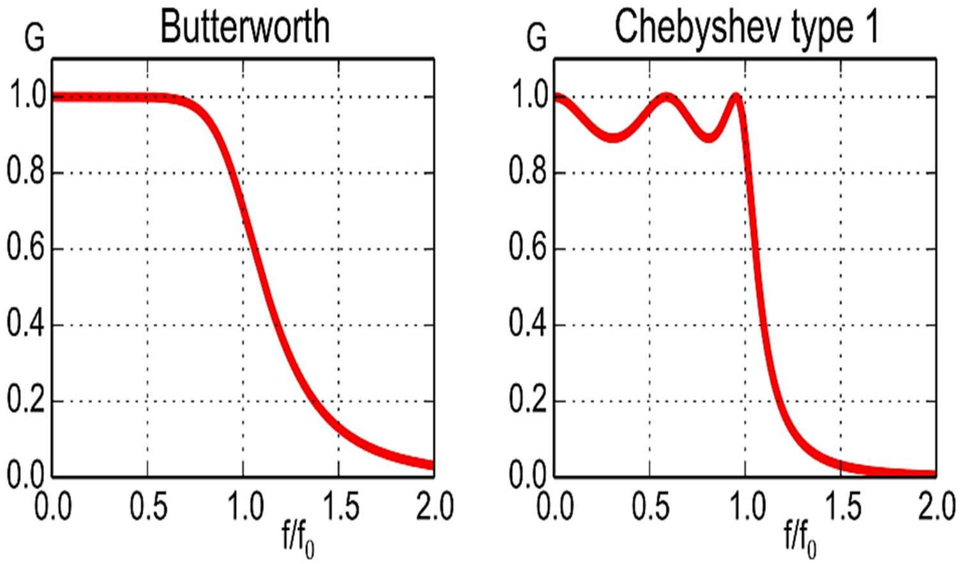
apply low pass filter to provided data

Parameters

- **data** (*NDArray[Float64]*) – data to filter, filter works in-place
- **sampling_rate** (*int*) – board's sampling rate
- **cutoff** (*float*) – cutoff frequency
- **order** (*int*) – filter order
- **filter_type** (*int*) – filter type from special enum
- **ripple** (*float*) – ripple value for Chebyshev filter

- The sampling rate in our case is 250Hz and cutoff frequency for HP filter is set to 0.1Hz and for LP is 95.0Hz I referred the order and filter type from sample code snippets available in brainflow documentation and confirmed their usefulness. The ripple value is 0 for HP and 0.1 for LP.
- The filter type used for HP filter is Butterworth filter and for LP filter is Chebyshev type 1 filter. The reason for using the filters is that Butterworth filter has a frequency response that almost has zero ripple and it is useful because at the start of collection we will get a lot of noise added which is not useful and Butterworth filter helps eliminates this. Chebyshev filters has some amount of ripple and is helpful to

retain some of the ERP components at the end of the experiment. This can be visualized from the diagram below:

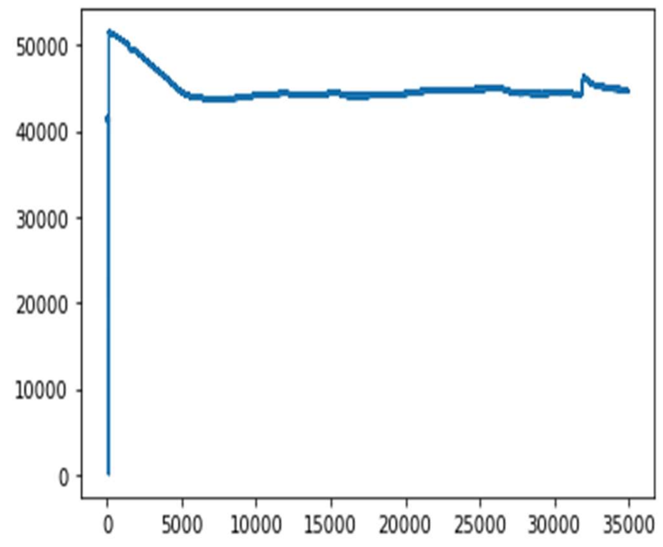


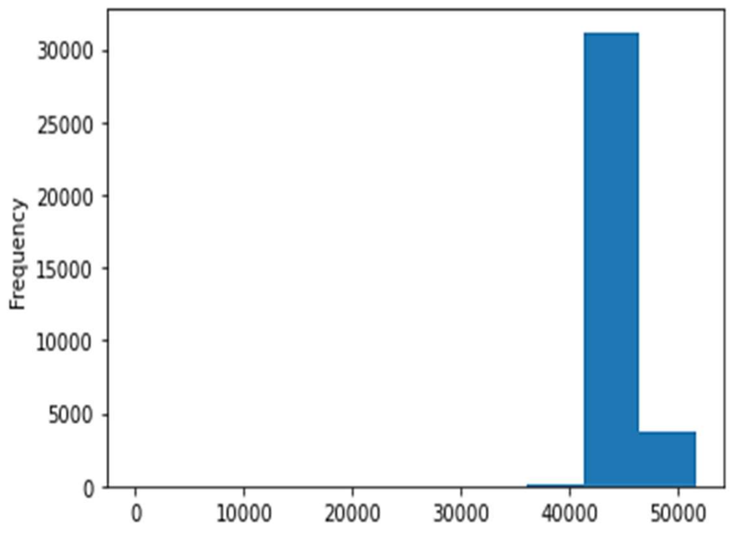
- Finally we just need to convert the voltage values in µV to RMS voltage values in µV.

5. **What is RMS value and why is it important?**
   - RMS stands for 'Root Mean Square'. It is the square root of the time average of the voltage squared.
   - The RMS value is the effective value of a varying voltage or current. It is equivalent steady DC value which gives the same effect. For Example: a lamp connected to 6V RMS AC supply will shine with the same brightness when connected to a steady 6V DC supply.
   - Hence the requirement of RMS voltage is justified and also the voltages for channels shown in the GUI are RMS values.
   - For converting voltage to RMS voltage we just need to multiply the voltage by 0.707.

These steps helped me clean my data and now the values are perfectly in the range of -100 to +100µVolts. Below graphs help visualize the data before and after cleaning. Note that this is the graph of a single channel i.e. channel 1.

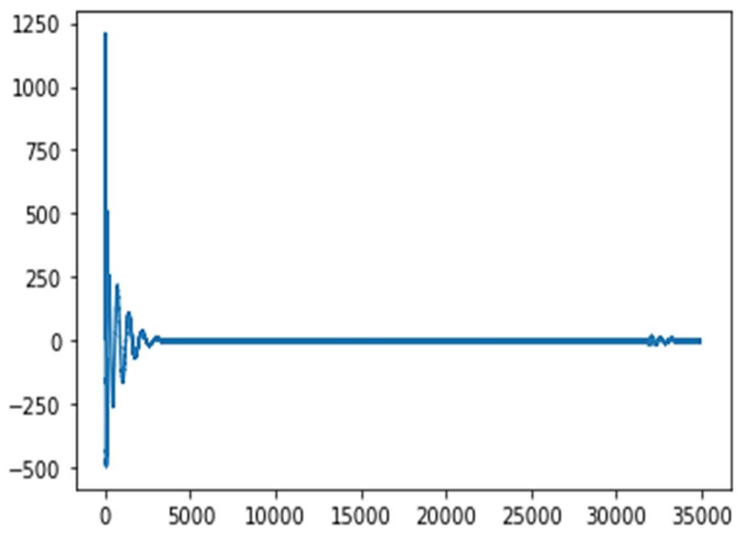
Values before cleaning

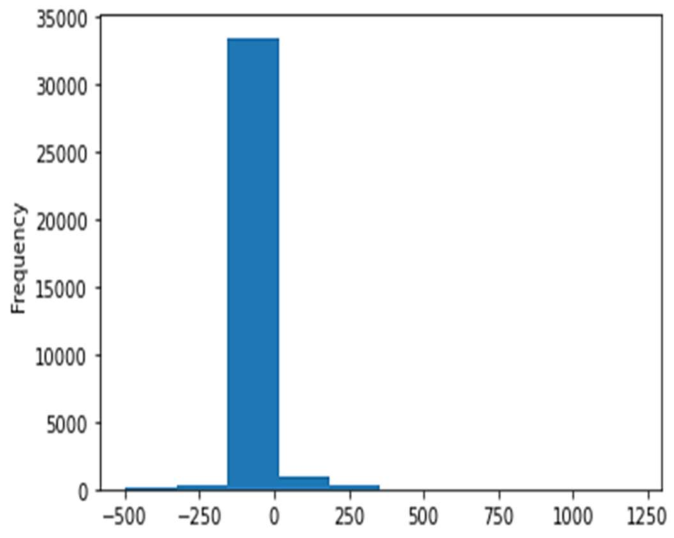




Notice that the values are very high compared to the limit of EEG signals. This is due to addition of noise and also due to the gain of the amplifier used in ADS1299 board used in OpenBCI Ultracortex Mark IV.

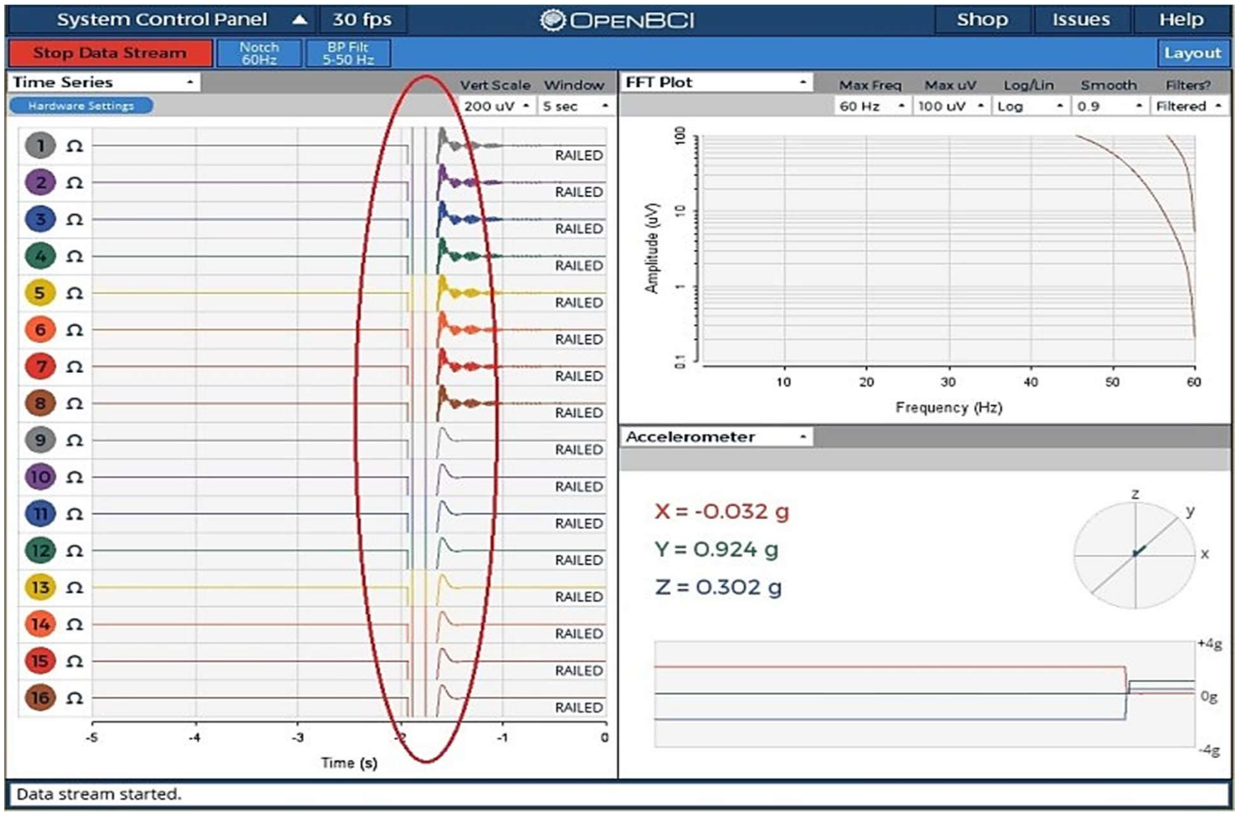
Values after cleaning





Now the values are in the correct range of EEG signals i.e. -100 to +100µV.

The initial spike is due to channel initialization that is performed by the cyton board once we start recording. This phenomenon can be easily observed in the screenshot below.



Vert Scale: 200 uV
Window: 5 sec
Max Freq: 60 Hz
Max uV: 100 uV
Log/Lin: Log
Smooth: 0.9
Filters?: Filtered

X = -0.032 g
Y = 0.924 g
Z = 0.302 g

Data stream started.

We can eliminate the spike by removing 3000-3500 rows from the dataset, which equals time period of about 13 seconds. Lastly we can remove the rows which has voltages below -100µV and above +100µV, as these are either EMG, EKG, EOG artifacts or are simply some other source of external noise like movement of the device, jerks etc.

References Workflow: https://openbci.com/forum/index.php?p=/discussion/2478/brainflow-replicating-the-pre-processing-steps-from-the-gui
EEG basics: https://wiki.cimec.unitn.it/tiki-index.php?page=Data+pre-processing
Brainflow docs: https://brainflow.readthedocs.io/en/stable/UserAPI.html#python-api-reference
My code: https://github.com/SEMANTICK/Datasets/blob/main/filteringAndDenoising.ipynb