Building and applying a brain-computer interface through electroencephalography

Olivier Berg & Hu Peng

Building and applying a brain-computer interface through electroencephalography

Document information

Title	Building and applying a brain-computer interface through electroencephalography
Authors	Olivier Berg & Hu Peng
Education	Pre-university education (‘VWO’)
School	Christelijk Gymnasium Utrecht
Profile	Nature and technology & nature and health
Subject	Physics
Supervisor	Rudolf Burggraaf-Wingens
Date	07-03-2024

Abstract

This paper presents the design, construction, and application of a non-invasive EEG-based brain-computer interface beyond traditional medical diagnostics. An EEG (electroencephalogram) is a recording of brain waves. For the EEG-machine we built a circuit which filters and amplifies the voltage measured across the scalp. Utilizing this self-made, cost-effective EEG-machine, we successfully captured and analysed alpha waves (8-12 Hz) in the brainwave data. With this, we were able to discern whether the subject had their eyes open and was concentrated, or had their eyes closed and was in a relaxed state of mind by calculating the root mean square voltage of the brain waves. With this brain-computer-interface (BCI) we implemented applications for communication and interaction, exemplified by a Flappy Bird game controlled via blinking. Analysis of external EEG datasets further extended our investigation. As these datasets contained multiple independent electric signals, rather than just one, the task of predicting the eye state was more complicated. We employed various machine learning models to do this: a K-Nearest Neighbors model, a support vector machine, and a neural network. The neural network was able to achieve the highest accuracy of 90%. Our research shows the feasibility of developing affordable, EEG-based BCIs for assistive technologies for, for instance, people with motor impairments.

Contents

1. Introduction

The brain consists of neurons which communicate through the transmission of electrical signals. We wanted to capture and apply these electrical signals. Therefore, for our PWS at Christelijk Gymnasium Utrecht we built an EEG (electroencephalogram) machine. This machine is designed to measure the brain waves, which are voltage differences that oscillate and represent synchronized activity over a network of neurons. The machine focuses in particular on the alpha waves which originate from the occipital lobe, as this is a very clear signal.[1] Alpha waves reflect activities in the visual cortex; when someone closes their eyes or relaxes, their magnitude increases when someone closes their eyes or relaxes and decreases with open eyes and concentration. The captured alpha waves are portrayed using a graph. This data can be used and applied to numerous activities. For further analysis we used external EEG datasets. This led us to formulate two connected research questions.

- How does an electroencephalogram (EEG) machine function in capturing and detecting the electrical signals within the brain?
- In what manners can the electrical signals derived from the brain be analysed and applied?

We were inspired by the Libet experiment for this project. He wanted to test whether humans have free will through the use of an EEG-machine. Participants were asked to perform a simple action, like pressing a button, while their brain activity was recorded. Results showed that brain activity associated with the action occurred before participants reported being aware of their intention to act, suggesting that unconscious processes precede conscious decision-making. This experiment makes use of an EEG-machine to record the brain activity, we found this so riveting that we wanted to attempt to build an EEG-machine ourselves.

We would like to thank our supervisor, Rudolf Burggraaf, for his guidance and support throughout the project. Moreover, we would like to thank Pannos Chatzoudis for his advice and assistance with documenting our project. Lastly, we thank Jan Wessels for being our test subject and demonstrating remarkable skill at playing Flappy Bird by blinking.

This paper has to be viewed as a manual for making and processing a basic EEG, and to analyse and apply EEG data. The concepts used to do this will not be discussed in detail. Refer to the footnotes whenever questions arise.

2. Wiring

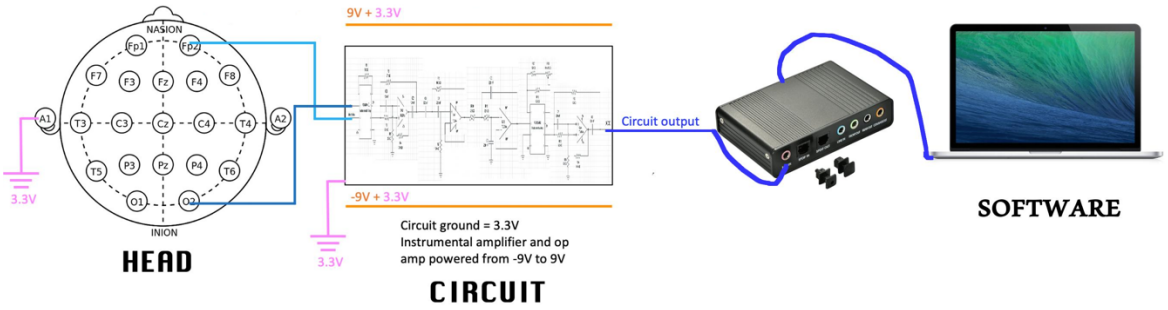


Figure 1: The entire wiring of the EEG machine.

This diagram describes the EEG machine. Three electrodes are taped to the subject's scalp at O2, Fp2, and A1 according to the 10-20 system.[1] These electrodes are attached to the circuit, which amplifies the signal coming from the scalp and filters all signals except for the alpha waves. The filtered signal is an analog signal, which the computer cannot understand. Therefore, the analog signal still has to be translated to a digital signal. This is done through an external USB-soundcard. The soundcard has an analog-to-digital converter.[2] After the signal is filtered, amplified, and digitized, it will be treated as a microphone by the computer.

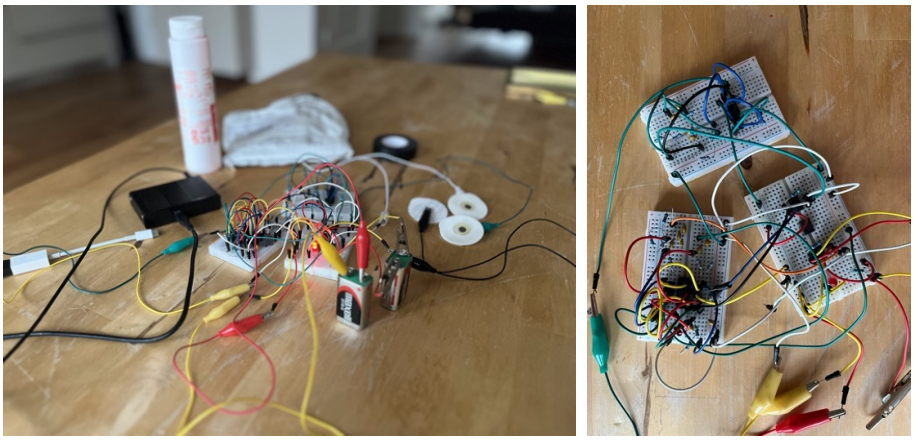


Figure 2: The actual wiring.

2.1 Electrode placement for alpha waves

We used one-time use ECG (electrocardiogram) electrodes, which can be used for an EEG as well. To measure alpha waves, three electrodes are needed: one at the left mastoid (the bone at the back of the left ear) which is connected to the ground of the circuit. This electrode helps with noise cancelling; one located one inch above and one inch to the right of the nasion (the midline bony

[1] '10–20 System (EEG)', in *Wikipedia*, 16 March 2023, https://en.wikipedia.org/w/index.php?title=10%E2%80%9320_system_(EEG)&oldid=1144970353.
[2] 'Analog-to-Digital Converter', in *Wikipedia*, 9 October 2023, https://en.wikipedia.org/w/index.php?title=Analog-to-digital_converter&oldid=1179394589.

depression between the eyes where the frontal and two nasal bones meet); the last one located one inch above and one inch right of the inion (the projecting part of the occipital bone at the base of the skull). The 2nd and 3rd electrode placements are approximately in O2 and Fp2 regions in the below diagram. The voltage differences between these electrodes are the alpha waves.[3]

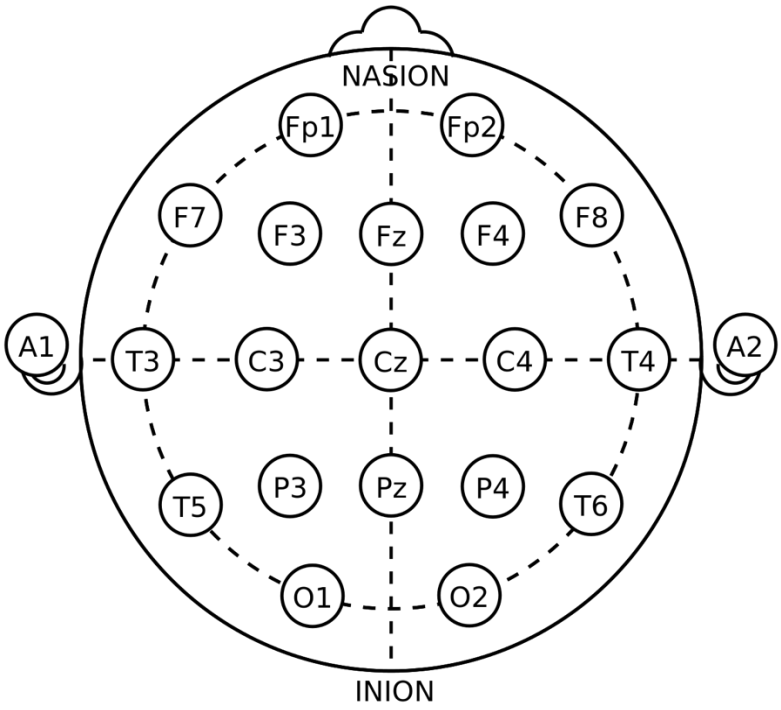


Figure 3: The 10-20 system for electrode placement.

[3] 'Einthoven's Triangle', in *Wikipedia*, 22 November 2023, https://en.wikipedia.org/w/index.php?title=Einthoven%27s_triangle&oldid=1186304522.

3. Hardware

3.1 Materials

This is the materials list. Every item has a small explanation on what their purpose is and tips about what to be cautious of when buying the materials. Everyone located in the Netherlands are in luck because everything should be available. Every item has a hyperlink to where we bought them. For everyone located outside the Netherlands, Amazon should have most items and some items need to be bought from an online store like Digikey or Mouser, which might have unexpected tax and delivery fees.

Amazon

- Breadboards, try to buy one big breadboard. We on the other hand bought three small breadboards because they were cheaper.
- Alligator clips with wires, do not forget that wires have to be attached to the alligator clips.
- 3.5 mm audio jack male to male, headphones use the same wire, both ends have to be a jack.
- Electrical tape, tape that does not conduct electricity.
- Two 9V batteries
- ECG electrodes, they can also be used for an EEG.[4]
- Capacitor set non polarized, metalized polyester film is recommended, because the values have to be 10 nF to 470 nF.
- Resistor set, a big range of values is recommended.
- Jumper wires male to male, these are the wires on the breadboard, however these Jumper wires look less cluttered.
- External USB soundcard, easy way to translate the analog signal to digital as it has an analogue to digital converter inside.
- Breadboard LED, it does not matter which colour, however the required resistance should be determined.
- Mini USB to USB A cable, the cable used to connect the soundcard to the laptop/computer.

Mouser

- Two AD620ANZ or AD620BNZ (DIP 8). This is the instrumental amplifier and make sure you buy DIP 8 (!) because any other type does not fit on the breadboard. The B-type is a little better than the A-type, so if you have enough funds, go for the B type.
- SPST (single pole single throw) switch, used to turn off the circuit.
- Four TL084CN, this is the quad operational amplifier (4 op-amps in one chip). Only one is required, however it can be damaged quite easily so buying multiple is safer.
- Four TL082ACP, this is the dual operational amplifier (2 op-amps in one chip). Only one is needed, however it can be damaged quite easily so buying multiple is safer.

[4] E. Seitsonen, A. Yli-Hankala, and K. Korttila, 'Are Electrocardiogram Electrodes Acceptable for Electroencephalogram Bispectral Index Monitoring?', *Acta Anaesthesiologica Scandinavica* 44, no. 10 (November 2000): 1266–70, https://doi.org/10.1034/j.1399-6576.2000.441014.x.

All these items are needed to build a DIY EEG machine. Make sure to buy everything in the right size, look at the datasheets of every chip in order to understand which type is which and ask your teachers if there are any items from the list available from school, that will save you some money.

3.2 Circuit

Amplifying and filtering are the main purposes of the circuit. The signal coming from the brain is 15 µV to 50 µV, which is too small to measure. To solve that, an AD620 is used for amplifying. There will also be a lot of noise coming from the alternating current. In Europe, the AC is 50 Hz and in America, it is 60 Hz. This circuit focuses on the alpha waves. Thus, only 8-12 Hz is needed.

The term ground is always used when building any electrical circuit.[5] In this circuit, a dual rail power supply is required to power the operational amplifiers.[6] Just remember that every ground needs to be connected to each other. Look at the pictures below to see how the batteries are wired on a breadboard. In the schematic, the positive voltage terminal is connected to pin 7, and the negative terminal is connected to pin 4 of the AD620. There is also an LED used for remembering whether the circuit is turned on or off.

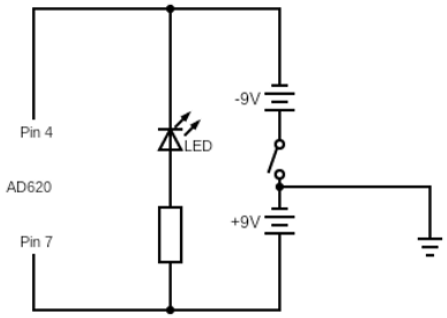


Figure 4: Battery supply.

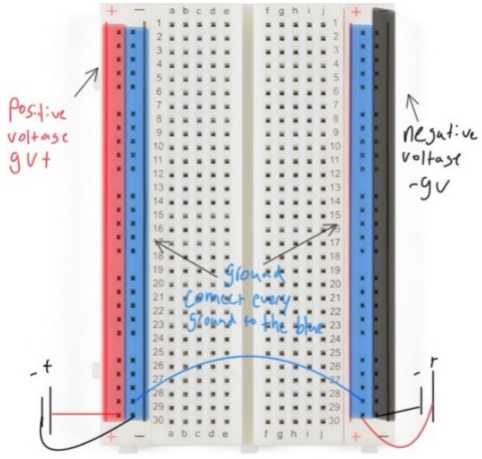


Figure 5: The breadboard ports to use.

[5] Øyvind Nydal Dahl, 'What Is Ground in Electronic Circuits?', Build Electronic Circuits, 18 September 2020, https://www.build-electronic-circuits.com/what-is-ground/.
[6] *Electronics Single versus Dual or Split Power Supply Explained*, 2020, https://www.youtube.com/watch?v=jiWKQEOSgIs.

After building the battery supply the main circuit needs to be built. Read everything about the circuit first and start building only after that. The following image shows the entire circuit.

- Two AD620 instrumental amplifiers
- Two notch filters
- Low pass filter
- High pass filter

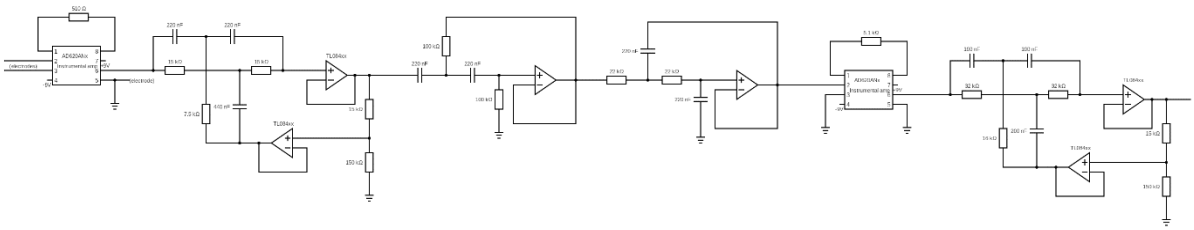


Figure 6: The entire circuit.

First amplifier (gain ≈ 100)

The minuscule signal that comes from the brain will go through the instrumental amplifier.[7] The resistance value between pin 1 and 8 determines the gain. Refer to the datasheet of the AD620 for the gain.[8] We used a 510 Ω resistor for a gain of around 100 times the original voltage. Connect electrode O2 to pin 2, Fp2 to pin 3, and A1 to pin 5. Pin 5 also needs to be connected to the ground. Lastly, pin 6 is the output of the signal, which will go to the notch filter.

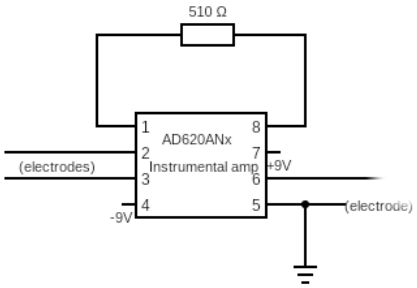


Figure 7: The first amplifier.

First notch filter (50 Hz)

In this circuit, a twin-T notch filter configuration is used.[9] This filter is great at filtering the 50 Hz noise. The values of the resistors and capacitors can be different than what we used.[10] There are lots of online notch filter calculators.[11] Try to avoid using many resistors in series or capacitors in parallel, as it will produce worse results. Try to get as close as possible to 50 Hz using the formula $f = \frac{1}{4\pi RC}$.

[7] *Introduction to Instrumentation Amplifiers*, 2020, https://www.youtube.com/watch?v=NvyDw8ZpLd0.
[8] 'Datasheet of the AD620', accessed 22 February 2024, https://www.analog.com/media/en/technical-documentation/data-sheets/ad620.pdf.
[9] Wayne Storr, 'Band Stop Filter and Notch Filter Design Tutorial', *Basic Electronics Tutorials* (blog), 20 October 2015, https://www.electronics-tutorials.ws/filter/band-stop-filter.html.
[10] *Resistors - Ohm's Law Is Not a Real Law*, 2015, https://www.youtube.com/watch?v=G3H5lKoWPpY; *Capacitors and Capacitance: Capacitor Physics and Circuit Operation*, 2016, https://www.youtube.com/watch?v=f_MZNsEqyQw.
[11] Alexander C. Frank Changpuak aka, 'Online Engineering Calculator :: Active Twin - T - Notch Filter Calculator', www.changpuak.ch, accessed 26 November 2023, https://www.changpuak.ch/electronics/Active_Notch_Filter.php; 'Twin-T Filter', accessed 26 November 2023, https://www.falstad.com/circuit/e-twint.html; 'Notch Filter Calculator', accessed 26 November 2023, https://www.learningaboutelectronics.com/Articles/Notch-filter-calculator.php#answer1.

The notch filter consists of two parts: a high pass filter and a low pass filter. The values of the high pass filter are calculated using the formula, and the values of the low pass filter are double the values of the high pass filter.

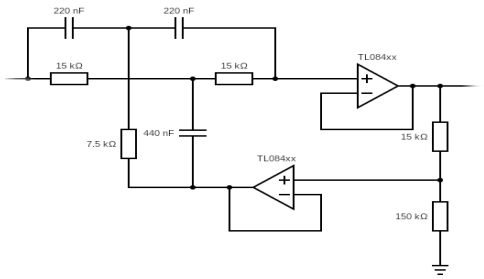


Figure 8: The first notch filter.

High pass filter (cut-off frequency ≈ 7.2 Hz)

A passive high pass filter will let current with frequencies higher than the cut-off frequency through.[12] Every filter is a voltage divider, however, instead of two resistors, one resistor is replaced by a capacitor. The filter uses the capacitive reactance of a capacitor to do that.[13] The first picture is the voltage divider, and the second and third are the same high pass filter but drawn differently.

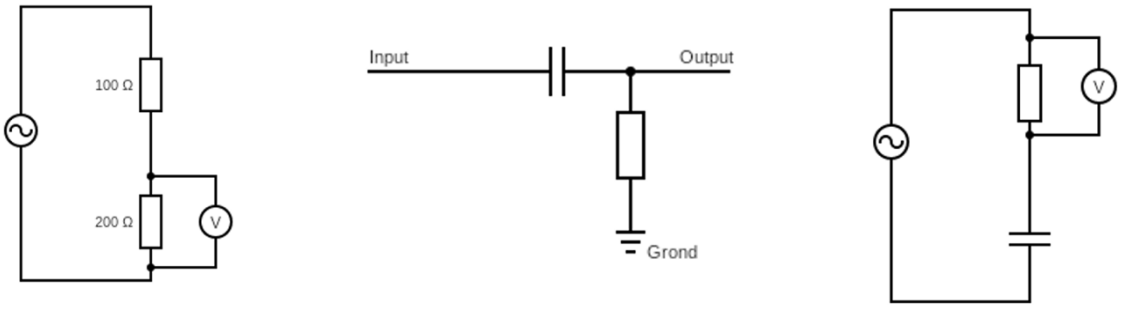


Figure 9: The workings of a high pass filter.

There is also a phenomenon called attenuation, which causes the output voltage to be lower than the input voltage.[14] This problem can be solved using an active high pass filter.[15] The active filter uses a unity gain filter.[16] In this circuit, an operational amplifier (TL084CN) is utilized.[17] We used the Sallen key topology because it was the most efficient.[18]

[12] Wayne Storr, 'High Pass Filter - Passive RC Filter Tutorial', *Basic Electronics Tutorials* (blog), 14 August 2013, https://www.electronics-tutorials.ws/filter/filter_3.html.
[13] Wayne Storr, 'Capacitive Reactance - The Reactance of Capacitors', *Basic Electronics Tutorials* (blog), 14 August 2013, https://www.electronics-tutorials.ws/filter/filter_1.html.
[14] Lynnette Reese, 'What Is Signal Attenuation?', Analog IC Tips, 5 May 2020, https://www.analogictips.com/what-is-signal-attenuation/.
[15] Wayne Storr, 'Active High Pass Filter - Op-Amp High Pass Filter', *Basic Electronics Tutorials* (blog), 14 August 2013, https://www.electronics-tutorials.ws/filter/filter_6.html.
[16] *Op Amp Circuits: Analog Computers from Operational Amplifiers*, 2016, https://www.youtube.com/watch?v=_o4ScgRZtNI.
[17] 'Tl082a.Pdf', accessed 26 November 2023, https://www.ti.com/lit/ds/symlink/tl082a.pdf?ts=1699103154231&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FTL082A%252Fpart-details%252FTL082ACP.
[18] Wayne Storr, 'Sallen and Key Filter Design for Second Order RC Filters', *Basic Electronics Tutorials* (blog), 7 October 2021, https://www.electronics-tutorials.ws/filter/sallen-key-filter.html.

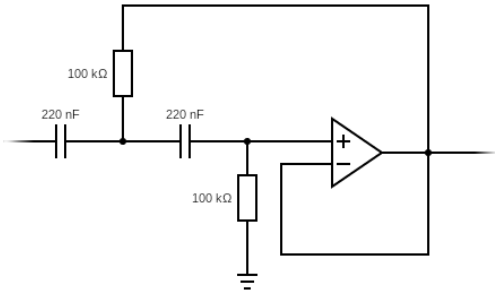


Figure 10: The high pass filter.

Low pass filter (cut-off frequency ≈ 32.9 Hz)

The passive low pass filter is almost the same as the passive high pass filter.[19] The only difference is that the capacitor is switched. The cut-off frequency is a lot higher than 8-12 Hz because we did not want to let the slope of the filter affect our signal. For the same reason as above, an active low pass filter with the Sallen key topology is utilized.[20]

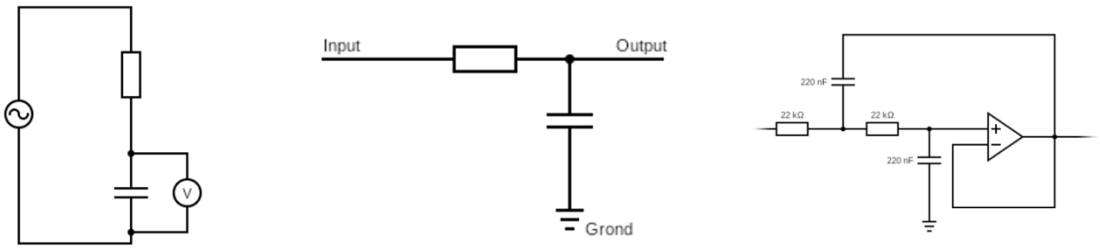


Figure 11: The workings of the low pass filter.

Second instrumental amplifier (gain = 10)

This is the last amplifier in the circuit. It has a gain of 10, so the original electrical signal will be amplified 100 x 10 = 1000 times. The output voltage is then high enough to be picked up by the mic in port in the external USB sound card. Before the signal is translated by the analog-to-digital converter in the sound card, it must be filtered one last time.[21]

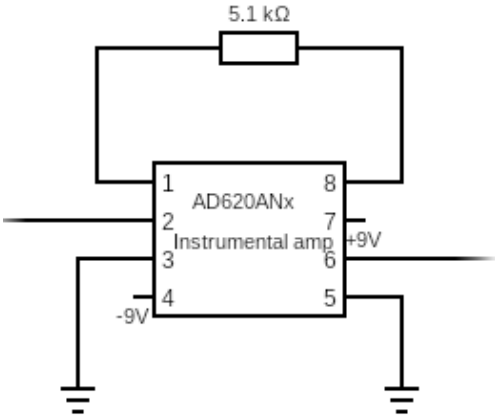


Figure 12: The second instrumental amplifier.

[19] Wayne Storr, 'Low Pass Filter - Passive RC Filter Tutorial', *Basic Electronics Tutorials* (blog), 14 August 2013, https://www.electronics-tutorials.ws/filter/filter_2.html.
[20] Wayne Storr, 'Active Low Pass Filter - Op-Amp Low Pass Filter', *Basic Electronics Tutorials* (blog), 14 August 2013, https://www.electronics-tutorials.ws/filter/filter_5.html.
[21] 'Analog-to-Digital Converter'.

Second Notch filter (cut-off frequency = 50 Hz)
Exactly the same as the notch filter above. To get a better output signal, use different resistor and capacitor values.

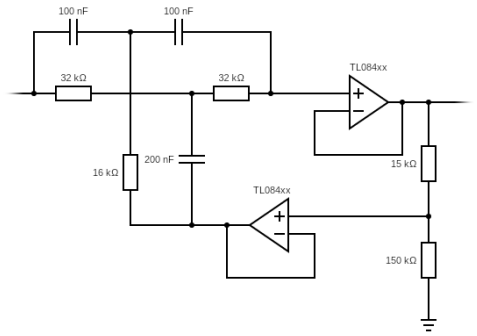


Figure 13: The second notch filter.

Computer connection
Connect the output signal to the tip of the audio cable and connect the sleeve to the ground in the circuit using alligator clips. Put the other side of the audio cable in the external USB sound card and connect that to the laptop. Open up Audacity and make sure that the input is the mic in audio input from the sound card. It is possible to play with the value of the input sensitivity.

Figure 14: The headphone jack.

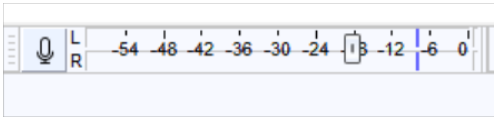


Figure 15: Audacity input sensitivity.

4. Software

As this was a practical PWS, not every concept is entirely explained. However, links to websites which explain the concepts are provided in the footnotes.

4.1 Recording the data

We record the data similar to a microphone and save it to a .wav file:

```
import sounddevice as sd
from scipy.io.wavfile import write

SAMPLE_RATE = 44100  # This is the sample rate for aux in
SECONDS = 10
DEVICE = "USB Sound Device        , Core Audio"  # to list all
available devices: python -m sounddevice


input(f"Press [enter] to start recording for {SECONDS} seconds.")


recording = sd.rec(int(SECONDS * SAMPLE_RATE),
samplerate=SAMPLE_RATE, device=DEVICE, channels=1)
sd.wait()


write("brain_data.wav", SAMPLE_RATE, recording)
```

4.2 Processing the data

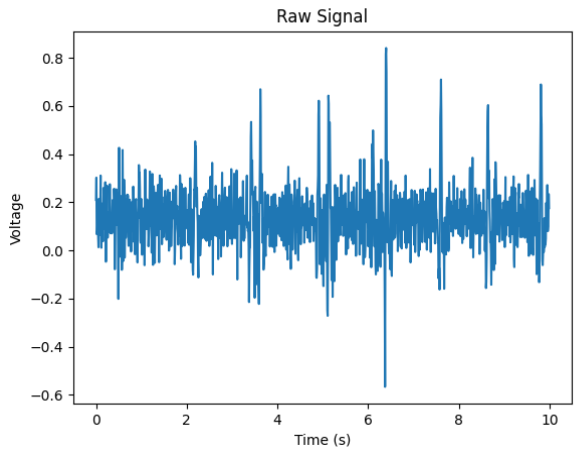


Figure 16: The plotted raw signal of 10 seconds of brain activity recorded by the EEG machine.

To extract the alpha waves (8-12Hz) from the raw data, which still contains some noise, we first need to calculate a power spectrogram of the data using a Fourier transform. The power spectrum shows the frequency distribution of the signal.[22]

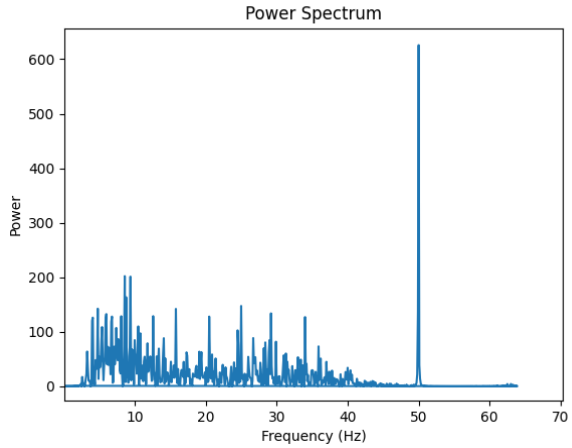


Figure 17: A power spectrum showing the frequency distribution of 10 seconds of raw EEG signal.

The spectrum shows clear peaks at 8-12 Hz, however there is still a lot of noise in the data (especially at 50 Hz as Europe's electricity network operates at 50 Hz).[23] We can filter this by setting all frequency components outside of 8-12 Hz to 0, and by taking the inverse Fourier it is possible to transform the data into a reconstructed signal. The reconstructed signal will only contain 8-12 Hz waves.

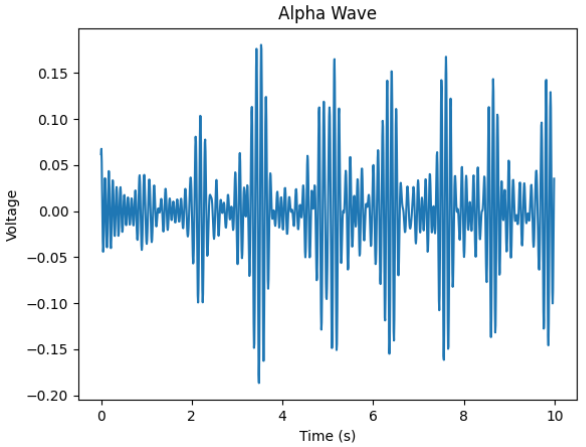


Figure 18: The reconstructed signal showing only the 8-12 Hz alpha waves.

The data shows the alpha waves whenever the subject closed and opened his eyes. Closing his eyes corresponds to the peaks in the graph above.

[22] *But What Is the Fourier Transform? A Visual Introduction.*, 2018, https://www.youtube.com/watch?v=spUNpyF58BY; Real Python, 'Fourier Transforms With Scipy.Fft: Python Signal Processing – Real Python', accessed 24 November 2023, https://realpython.com/python-scipy-fft/.

[23] 'Netspanning', in *Wikipedia*, 10 February 2024, https://nl.wikipedia.org/w/index.php?title=Netspanning&oldid=67012357.

4.3 Analysing the data

```
"""
This code was taken from:
https://github.com/ryanlopezzzz/EEG/blob/main/analysis_tools.py
The only difference is using the fourier transform from scipy
instead of numpy: https://scipy.org/faq/#what-is-the-difference-
between-numpy-and-scipy
"""

import math
import time

import numpy as np
import scipy
from scipy.special import erf


def get_power_spectrum(time_series):
    """
    Applies window function and fourier transform to time_series
data and returns power spectrum of FFT: ps[freq]=|FFT(x)|^2[freq]

    :param time_series: Numpy array containing the most recent ADC
voltage difference measurements.
    """
    time_series_zero_mean = time_series - np.mean(time_series)
    time_series_fft = scipy.fft.fft(time_series_zero_mean)
    ps = np.abs(time_series_fft) ** 2  # Gets square of complex
number
    return ps

def get_brain_wave(time_series, freq_min, freq_max, freq):
    """
    Eliminates the components of time_series which have frequency
above freq_max or below freq_min giving
    brain waves.
    """
    time_series_fft = scipy.fft.fft(time_series)
    freq_abs = np.abs(freq)
    time_series_fft[
        (freq_abs > freq_max) | (freq_abs < freq_min)
    ] = 0  # set frequencies outside of [freq_min,freq_max] to 0
    brain_wave_complex = scipy.fft.ifft(time_series_fft)
    brain_wave = np.real(brain_wave_complex)  # get rid of zero
imaginary component
    return brain_wave
```

Using those functions to plot the data:

```
import matplotlib.pyplot as plt
import numpy as np
import scipy
import soundfile as sf
from analyze_functions import (get_brain_wave, get_power_spectrum,
                               get_rms_voltage)

FILE = ""
SECONDS = 10
RESAMPLE_RATE = 128
nsamples = int(SECONDS * RESAMPLE_RATE)
sinterval = 1 / RESAMPLE_RATE
freq_min = 8  # minimum frequency for alpha waves
freq_max = 12  # maximum frequency for alpha waves


data, _ = sf.read(FILE, dtype="float32")
data = scipy.signal.resample(data, nsamples)

xpoints = np.arange(0, SECONDS, sinterval)


f1, ax1 = plt.subplots()
ax1.plot(xpoints, data)
ax1.set(xlabel="Time (s)", ylabel="Voltage", title="Raw Signal")
f1.show()

ps = get_power_spectrum(data)
freq = np.fft.fftfreq(nsamples, d=sinterval)  # frequencies for FFT
of data
f2, ax2 = plt.subplots()
ax2.plot(freq, ps)
ax2.set(xlabel="Frequency (Hz)", ylabel="Power", title="Power
Spectrum")
f2.show()

brain_wave = get_brain_wave(data, freq_min, freq_max, freq)
f3, ax3 = plt.subplots()
ax3.plot(xpoints, brain_wave)
ax3.set(xlabel="Time (s)", ylabel="Voltage", title="Alpha Wave")
f3.show()

rms = get_rms_voltage(ps, freq_min, freq_max, freq, nsamples)
print("Alpha wave root mean square (rms) voltage is: ", rms)
input("\nPress <Enter> to exit...\n")
```

4.4 Predicting whether someone's eyes are open or closed (our data)

Predicting whether we had our eyes open or closed was not very difficult with our own data. There is only one feature, the difference between the O2 and the Fp2 channel, so there is no need for complicated models. For the ensuing external dataset, as it contains the data of multiple EEG channels, these complicated models are required. For our own data, we only need to calculate the Root Mean Squared (RMS) voltage from the Fourier series.[24]

```
"""
This code was taken from:
https://github.com/ryanlopezzzz/EEG/blob/main/analysis_tools.py
"""

def get_rms_voltage(ps, freq_min, freq_max, freq, time_series_len):
    """
    Gets the Root-Mean-Square (RMS) voltage of waves with .
frequency between freq_min and freq_max. Parseval's Theorem says:
    \sum_{i=0}^{N-1} x[i]^2 = \frac{1}{N} \sum_{i=-(N-1)}^{N-1}
|FFT(x)[i]|^2
    Using this, the RMS voltage is (first formula is definition,
second is implemented evaluation):
    \sqrt{ \frac{1}{N} \sum_{i=0}^{N-1} x[i]^2 } = \frac{1}{N}
\sqrt{ \sum_{freq in range} |FFT(x)[freq]|^2}

    :param ps: FFT power spectrum of time_series, ps[freq] =
|FFT(x)|^2[freq].
    :param freq_min: Number corresponding to minimum alpha wave
frequency in Hz.
    :param freq_max: Number corresponding to maximum alpha wave
frequency in Hz.
    :param freq: Numpy array of negative and positive FFT freq,
fft.fftfreq of time_series
    :param time_series_len: Integer length of time_series
    """
    freq_abs = np.abs(freq)
    ps_in_range = ps[
        (freq_abs <= freq_max) & (freq_abs >= freq_min)
    ]  # Gets power spectrum for freq in range [freq_min, freq_max]
    rms = (1 / time_series_len) * np.sqrt(np.sum(ps_in_range))
    return rms
```

After calibrating with the user by recording samples of the eyes open state and eyes closed state, we can calculate the ideal cut-off Root Mean Squared voltage for when the eyes are open and closed:

```
def gaussian_eval(relaxed, concentrated):
    """
    We approximate concentrated and relaxed brain wave data sets
each as normal Gaussian distributions.
    The cross point of the two gaussians give the best V0 (threshold
```

[24] *Power Electronics Introduction - What Is Power Electronics?*, accessed 24 November 2023, https://www.youtube.com/watch?v=hRAyfJLZnC0&list=PLmK1EnKxphinxBub5hL0ZoJXWoqjkGE19.

```
voltage which separates relazed and concentrated data) which
minimizes over all error.
    The overlap area divided by 2 give the probability of wrong
classification
    The ratio of overlap area left of V0 to right of V0 gives the
percentage of wrong estimation being we guessed concentrated but is
actually relaxed.

    :param data[0]: relaxed data time average
    :param data[1]: concentrated time average

    RETURN:
        V0: threshold voltage which separates relazed and
concentrated data
        c_overlap: probability of wrongly classified as relaxed
given person is concentrated
        r_overlap: probability of wrongly classified as concentrated
given person is relaxed
    """
    # calculate the meanie and std to construct the gaussian normal
distributions
    r_mean = np.mean(relaxed)
    c_mean = np.mean(concentrated)
    r_std = np.std(relaxed)
    c_std = np.std(concentrated)

    # Solve for cross point of the two normal distributions
    a = 1/(2*r_std**2) - 1/(2*c_std**2)
    b = c_mean/(c_std**2) - r_mean/(r_std**2)
    c = r_mean**2 /(2*r_std**2) - c_mean**2 / (2*c_std**2) -
np.log(c_std/r_std)
    results = np.roots([a,b,c])

    # Select the cross point in the middle of the two mean values.
    intersection = []
    for i in range(len(results)):
        if results[i] < r_mean and results[i] > c_mean:
            intersection.append(results[i])
    V0 = intersection[0]

    # Calculate the overlap area using erf function
    r_z = (V0 - r_mean)/(r_std * math.sqrt(2))
    r_overlap = (1-abs(erf(r_z)))/2
    c_z = (V0 - c_mean)/(c_std* math.sqrt(2))
    c_overlap = (1-abs(erf(c_z)))/2

    return V0, c_overlap, r_overlap

def get_cutoff(relaxed_rms, concentrated_rms):
    cutoff, _, __ = gaussian_eval(relaxed_rms, concentrated_rms)
    return cutoff
```

4.5 Flappy bird

Flappy Bird is coded in Python using pygame. Instead of being able to jump by pressing the space button, one can jump by blinking. As the game updates nearly 100 times a second, it is not possible to perform the signal processing as done above, because every sample is only 100th of a second long. Therefore, it is not possible to isolate the alpha waves from the signal. However, for this instance, the signal processing was not needed because blinking adds so much noise to the signal – not only do the brain waves change, but also a lot of muscles activate – the peaks are so significant that they can be detected in the raw signal. There is a downside to this, if one is in a building that receives a lot of radio signal, it is not possible to detect blinks. The brain signal gets drowned out by the stronger radio signal. For the live demonstration, see the EEG flappy bird video. On the right one can see the raw signal from the DIY EEG machine in Audacity, on the left, the user is blinking, and in the centre, the bird jumps whenever the user blinks. As the code is not very relevant for our research question, refer to the GitHub page for the game code.[25]

4.6 Data analysis on EEG data (external dataset)

The following datasets that we used consist of multiple EEG channels, which is why a more complicated analyses are needed. We built multiple models to predict whether someone's eyes are open or closed: a K Nearest Neighbors (KNN) model, a Support Vector Machine (SVM), and a neural network built using TensorFlow and Keras. These were built in Jupiter Notebooks. For a great explanation of all the concepts, we highly recommend Sentdex's machine learning YouTube course.[26] To train the Support Vector Machine, we recommend following 'A Practical Guide to Support Vector Classification'.[27] We used the following dataset: Eye State Classification EEG Dataset.[28]

KNN (K-Nearest Neighbors) model
Firstly, the data from Kaggle.com has to be loaded in and processed. If the Z-score, the number of standard deviations away from the mean, of a specific value is higher than 5, it will be filtered out.

```
import polars as pl
import numpy as np
from sklearn import neighbors, preprocessing
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score, mean_squared_error, roc_auc_sc
ore
import math
from sklearn.ensemble import BaggingRegressor
from imblearn import under_sampling
from sklearnex import patch_sklearn

patch_sklearn() # this only works for intel devices
```

```
--------------------------------------------------------------------------------

Intel(R) Extension for Scikit-learn* enabled (https://github.com/intel/sci
kit-learn-intelex)
```

[25] Olivier Berg, 'IkBenOlie5/EEG', Jupyter Notebook, 14 December 2023, https://github.com/IkBenOlie5/EEG.
[26] *Practical Machine Learning Tutorial with Python Intro p.1*, 2016, https://www.youtube.com/watch?v=OGxgnH8y2NM.
[27] Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin, 'A Practical Guide to Support Vector Classification', n.d.
[28] 'Eye State Classification EEG Dataset', accessed 24 November 2023, https://www.kaggle.com/datasets/robikscube/eye-state-classification-eeg-dataset.

```
df = pl.read_csv("./data/EEG_Eye_State_Classification.csv")
df.head()
```

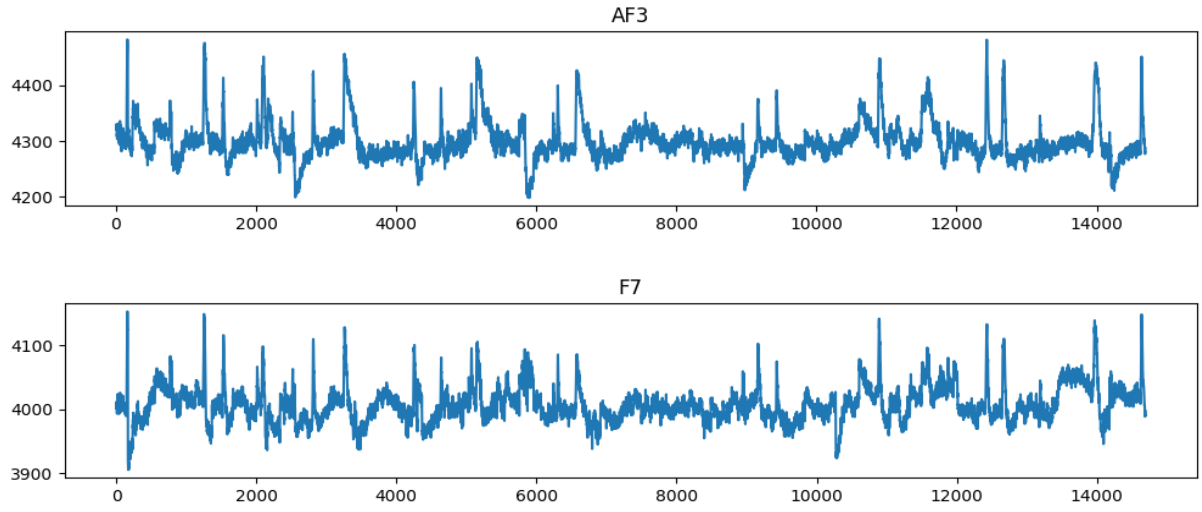
shape: (5, 15)

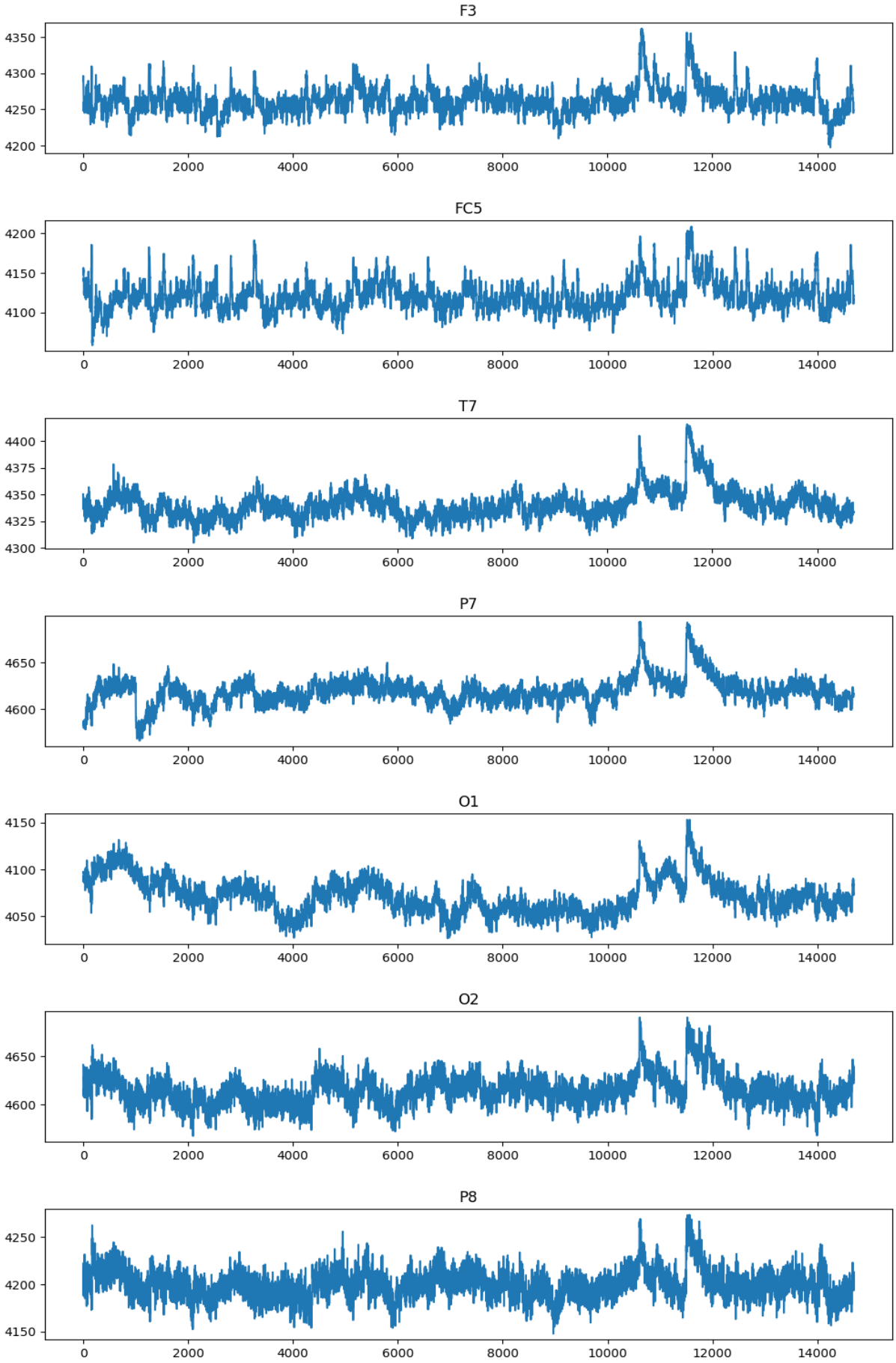
AF3	F7	F3	FC5	…	F4	F8	AF4	eyeDetection
f64	f64	f64	f64		f64	f64	f64	i64
4329.23	4009.23	4289.23	4148.21	…	4280.51	4635.9	4393.85	0
4324.62	4004.62	4293.85	4148.72	…	4279.49	4632.82	4384.1	0
4327.69	4006.67	4295.38	4156.41	…	4282.05	4628.72	4389.23	0
4328.72	4011.79	4296.41	4155.9	…	4287.69	4632.31	4396.41	0
4326.15	4011.79	4292.31	4151.28	…	4288.21	4632.82	4398.46	0

```
# removing outliers
for col in df.columns:
    zscores = (np.abs(stats.zscore(df.get_column(col))) < 5)
    df = df.filter(zscores)
```

The data then looks like this ('eyeDetection' is 0 when eyes are open and 1 when eyes are closed):

```
for col in df.columns:
    plt.figure(figsize=(12, 2))
    plt.title(col)
    y = df.get_column(col)
    plt.plot(range(len(y)), y)
```




F3
4350
4300
4250
4200
0
2000
4000
6000
8000
10000
12000
14000
FC5
4200
4150
4100
T7
4400
4375
4350
4325
4300
P7
4650
4600
O1
4150
4100
4050
O2
4650
4600
P8
4250
4200
4150

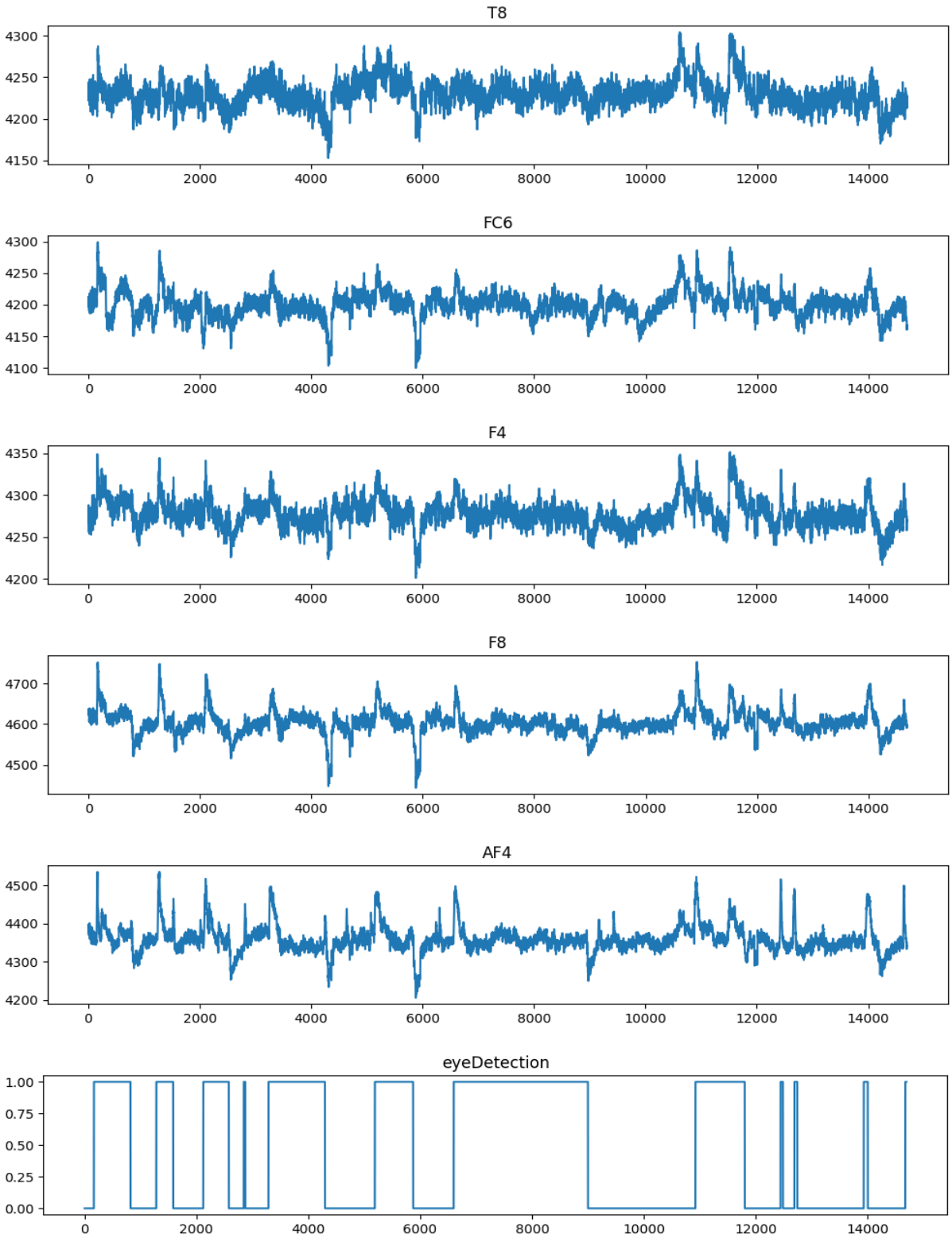


Figure 4: A visualization of all the columns from the dataset.

The dataset has to be split into two groups, a train and test group, consisting of 80% and 20% of the entire dataset respectively:

```
x_train, x_test, y_train, y_test = train_test_split(
    df.drop("eyeDetection").to_numpy(),
    df.select("eyeDetection").to_numpy(),
```

```
    test_size=0.2,
    random_state=42,
)
```

To make sure that the model is as accurate as possible, we use a process called hyperparameter tuning, where we try a bunch of options, search a grid of parameters, and select the parameters which work the best:

```
params = {"n_neighbors": range(1, 50), "weights": ["uniform", "distance"]}


best_params = (
    GridSearchCV(neighbors.KNeighborsClassifier(), params)
    .fit(x_train, y_train.ravel())
    .best_params_
)
best_params

------------------------------------------------------------------------

{'n_neighbors': 1, 'weights': 'uniform'}
```

The optimal parameters for the KNN model are n_neighbors=1 and weights='uniform'. The fact that the KNN model only looks at a single neighbor does not necessarily have to mean anything, but it is slightly suspicious. This will be explained later. The last step is training and testing the model. We use a bagged KNN model, this means that it trains multiple, in this case 10, of the same model, and then averages the predictions:

```
model = BaggingRegressor(neighbors.KNeighborsClassifier(**best_params), n_
estimators=10)
model.fit(x_train, y_train.ravel())
y_pred =  model.predict(x_test)

------------------------------------------------------------------------

rmse_train = math.sqrt(mean_squared_error(model.predict(x_train), y_train)
)
acc = accuracy_score(y_test, np.int32(y_pred))
rmse_test = math.sqrt(mean_squared_error(y_test, y_pred))

rmse_train, rmse_test, acc

------------------------------------------------------------------------

(0.09913480119052337, 0.15285446012893575, 0.9457729875378915)
```

An accuracy of 95% is achieved. Although on the surface this might seem great, it is remarkably high.

SVM model

The only difference in processing the data for the SVM model is that the data has to be scaled. We, as recommended in 'A Practical Guide to Support Vector Classification', used the MinMaxScaler. This linearly scales the data to the range [-1, 1].[29]

```
scaler = sklearn.preprocessing.MinMaxScaler((-1, 1))
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

There is also a difference in the hyperparameter tuning. As recommended by 'A Practical Guide to Support Vector Classification', first we loosely try to find the best parameters by searching a coarse grid. Then we make the differences between the searched parameters smaller and search a finer grid to find the optimal parameter.[30]

```
params_loose = {
    "C": [math.pow(2, exp) for exp in range(-3, 5)],
    "gamma": [math.pow(2, exp) for exp in range(-3, 5)],
}
best_params_loose = model = (
    GridSearchCV(svm.SVC(), params_loose).fit(x_train, y_train).best_param
s_
)
best_params_loose

--------------------------------------------------------------------------

{'C': 4.0, 'gamma': 16.0}

--------------------------------------------------------------------------

params_fine = {
    "C": [math.pow(2, exp / 4) for exp in range(5, 11)],
    "gamma": [math.pow(2, exp / 4) for exp in range(9, 19)],
}
best_params_fine = model = (
    GridSearchCV(svm.SVC(), params_fine, verbose=2).fit(x_train, y_train).
best_params_
)
best_params_fine

--------------------------------------------------------------------------

{'C': 2.378414230005442, 'gamma': 22.627416997969522}
```

Then we train and test the model:

```
model = BaggingRegressor(svm.SVC(**best_params_fine), n_estimators=10)
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
```

[29] Hsu, Chang, and Lin, 'A Practical Guide to Support Vector Classification'.
[30] Hsu, Chang, and Lin.

```
---------------------------------------------------------------------------

rmse_test = math.sqrt(mean_squared_error(y_test, y_pred))
rmse_train = math.sqrt(mean_squared_error(model.predict(x_train), y_train)
)
acc = accuracy_score(y_test, np.int32(y_pred))

rmse_test, rmse_train, acc

---------------------------------------------------------------------------
(0.15508542973118122, 0.013015031279794358, 0.9447831978319783)
```

Using a SVM also results in a remarkably high accuracy of 94%.

Shuffling the data

There is a reason for the remarkably high accuracy. Since the data is a time series, it should not be shuffled.[31] If the data is shuffled, all the data points are so close to each other that the model only has to "connect the dots". The model does not have to "generalise", which is what we want it to do. However, the train_test_split() function shuffles the data by default. If we add shuffle=False to the function, it will not shuffle the data:

```
x_train, x_test, y_train, y_test = train_test_split(
    df.drop("eyeDetection").to_numpy(),
    df.select("eyeDetection").to_numpy(),
    test_size=0.2,
    shuffle=False
)
```

After hyperparameter tuning, the best parameters for the KNN model are:

```
{'n_neighbors': 15, 'weights': 'uniform'}
```

However, if we train and test the model, it is no longer as accurate as before:

```
rmse_train, rmse_test, acc

(0.1595743116219084, 0.6400861974624038, 0.6968356583872065)
```

It is only able to achieve an accuracy of 70%. Not bad, but not very accurate either.

The same is true for the SVM model:

```
best_params_loose
{'C': 2.0, 'gamma': 4.0}

best_params_fine
{'C': 2.0, 'gamma': 4.756828460010884}
```

[31] 'Sklearn.Model_selection.TimeSeriesSplit', scikit-learn, accessed 24 February 2024, https://scikit-learn/stable/modules/generated/sklearn.model_selection.TimeSeriesSplit.html.

```
rmse_test, rmse_train, acc
```

```
(0.7269492998503029, 0.13987374674762937, 0.6480352303523035)
```

An accuracy of only 65%. It is only 15% better than randomly guessing – that would achieve an accuracy of 50%.

This can mean two different things. Either the KNN and SVM models are not right for the EEG data, or the data simply does not have features that really allow for the detection of the eye state.

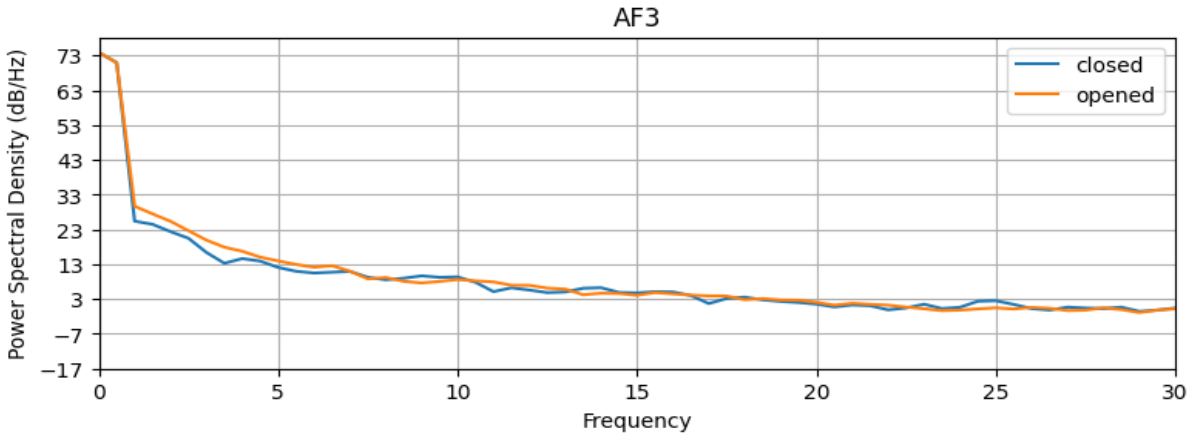
Alpha spindling

When the eyes are closed, we expect alpha spindling to occur in the EEG data. Alpha spindles are brief bursts of high-frequency alpha activity, typically lasting from 0.5 to 2 seconds.[32] To detect whether alpha spindles are present in the EEG data, one can look at the power spectra of the channels. If the alpha spindles are present, we would expect the alpha band (8-12 Hz) to be much higher when the eyes are closed compared to when the eyes are open. However, in all the channels, this is not the case:

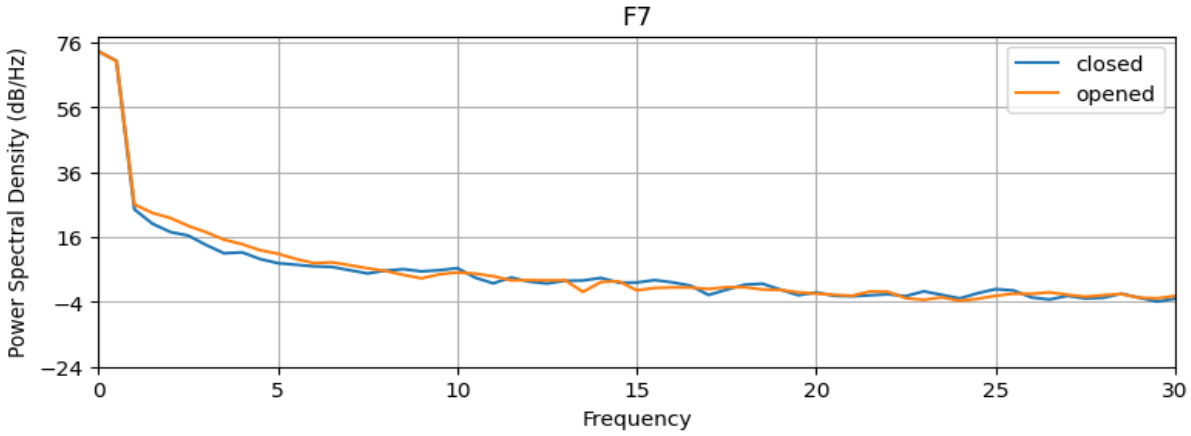
```
closed = df.filter(pl.col("eyeDetection") == 1).drop(["eyeDetection"])
opened = df.filter(pl.col("eyeDetection") == 0).drop(["eyeDetection"])
```

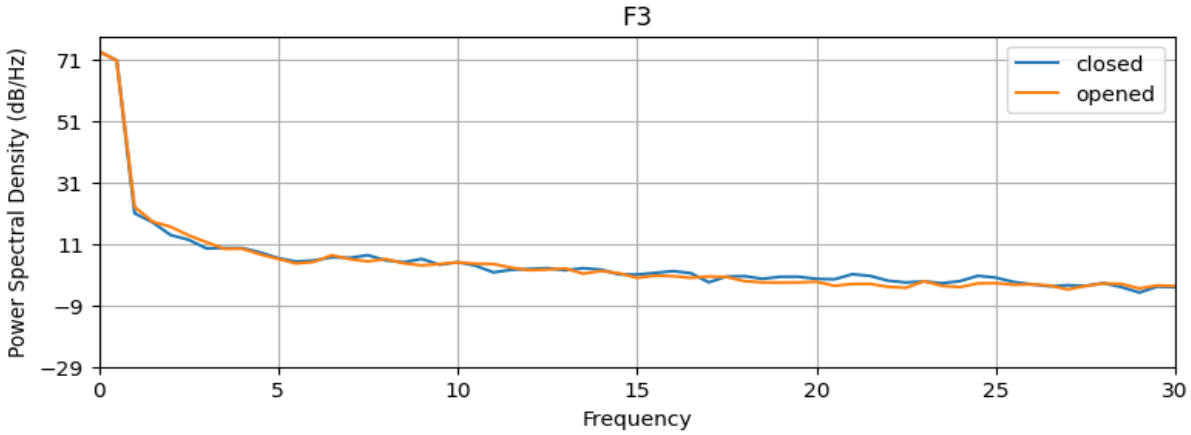
```
for col in closed.columns:
    plt.figure(figsize=(9,3))
    plt.title(col)
    plt.psd(closed.get_column(col), Fs=128)
    plt.psd(opened.get_column(col), Fs=128)
    plt.xlim([0, 30])
    plt.legend(["closed", "opened"])
```

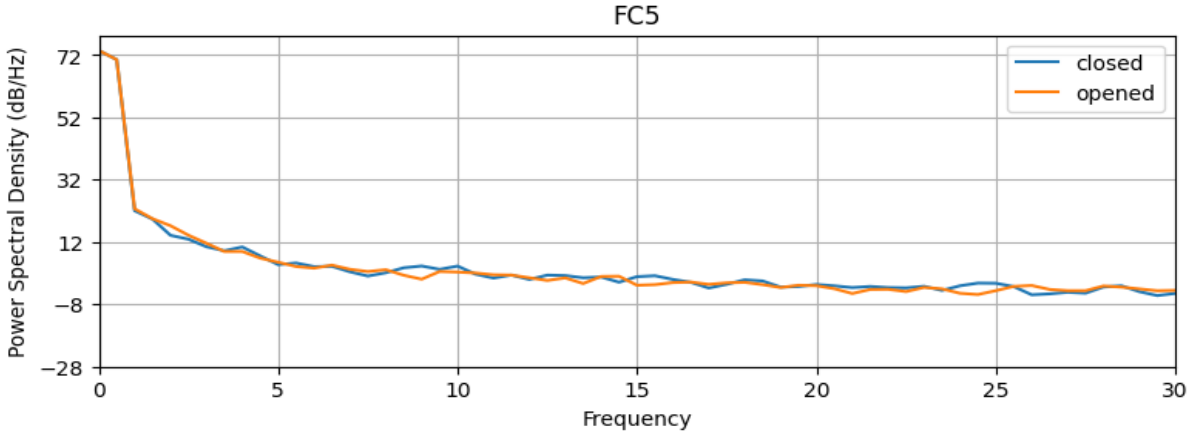
```
--------------------------------------------------------------------------------
```

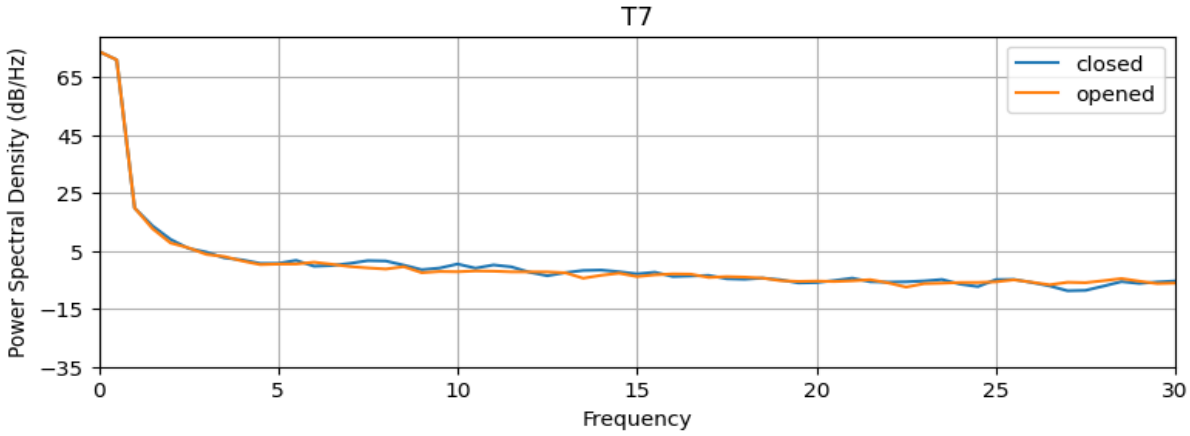


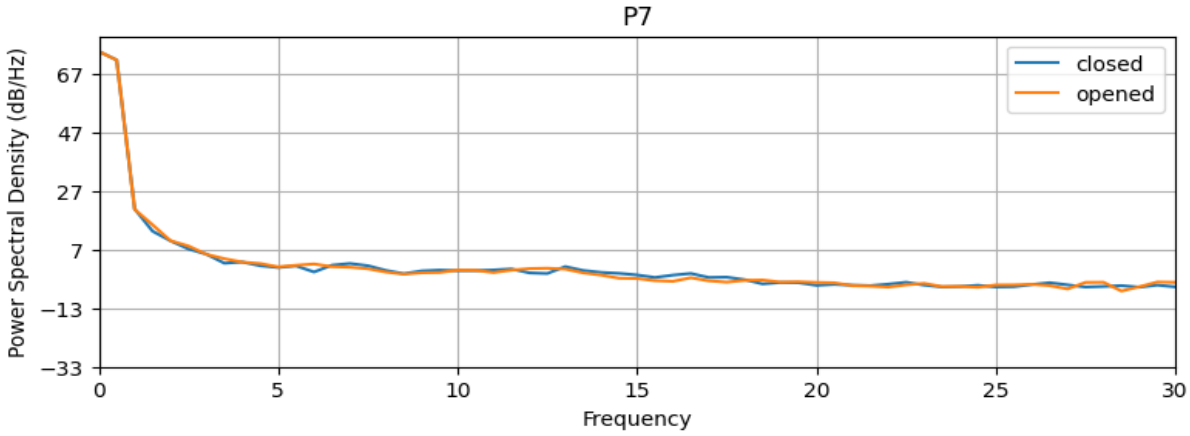
[32] Vernon Lawhern, Scott Kerick, and Kay A Robbins, 'Detecting Alpha Spindle Events in EEG Time Series Using Adaptive Autoregressive Models', *BMC Neuroscience* 14 (18 September 2013): 101, https://doi.org/10.1186/1471-2202-14-101.

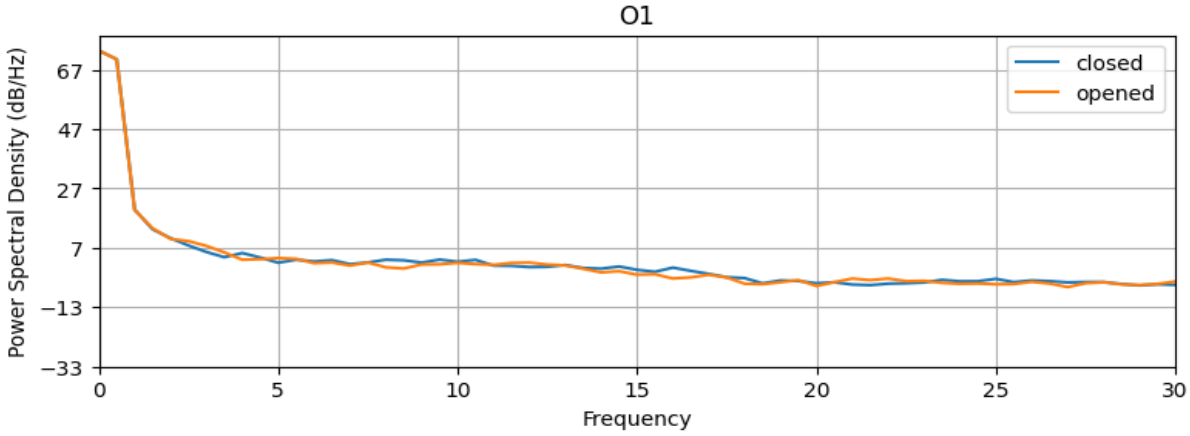

F7
Power Spectral Density (dB/Hz)
76
56
36
16
−4
−24
0
5
10
15
20
25
30
Frequency
closed
opened

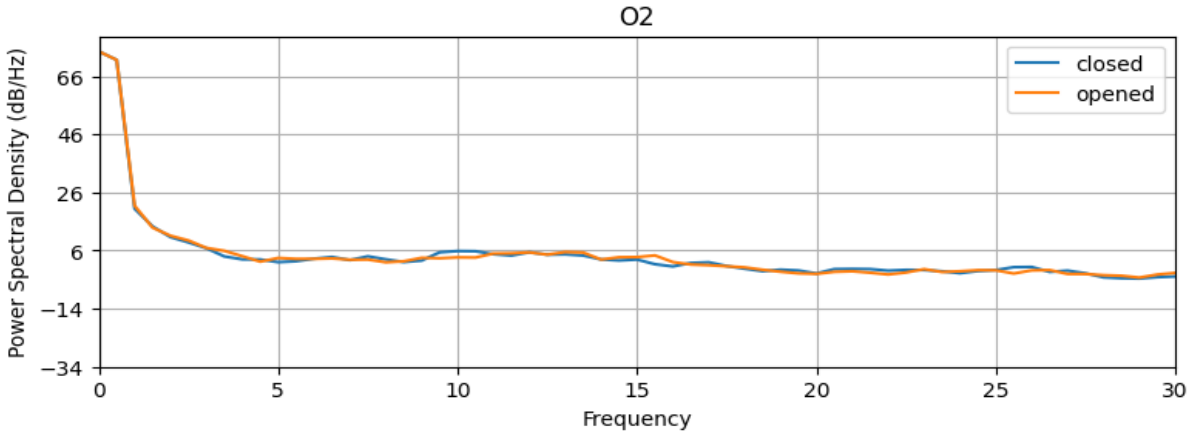

F3
Power Spectral Density (dB/Hz)
71
51
31
11
−9
−29
0
5
10
15
20
25
30
Frequency
closed
opened

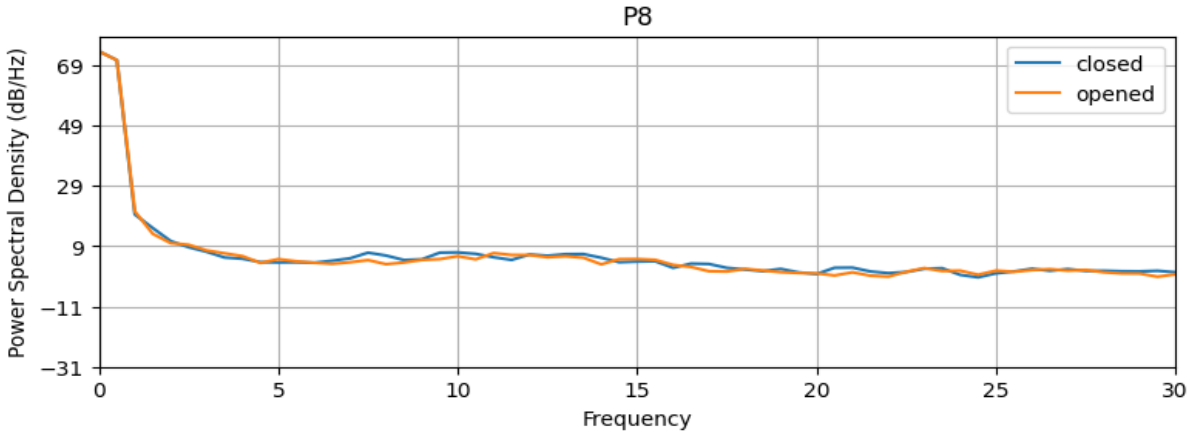

FC5
Power Spectral Density (dB/Hz)
72
52
32
12
−8
−28
0
5
10
15
20
25
30
Frequency
closed
opened

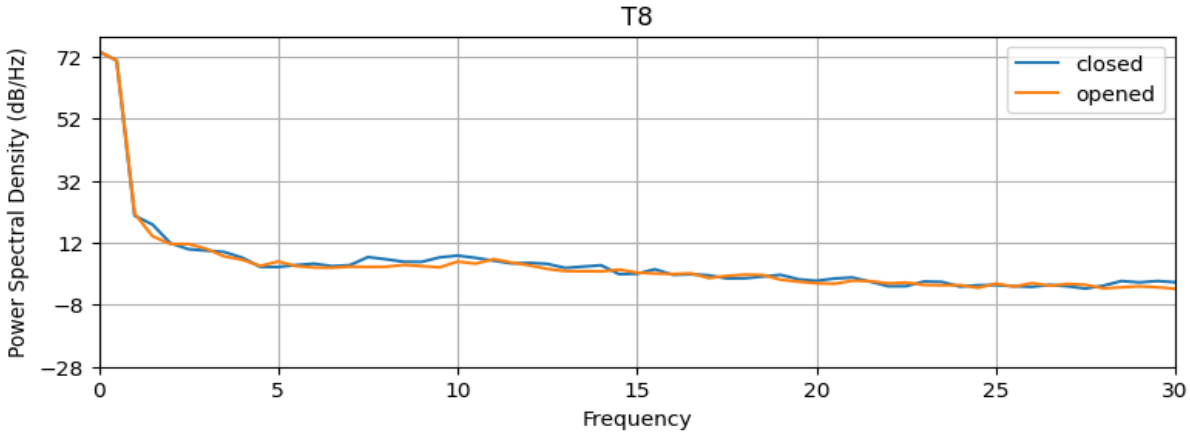

T7
Power Spectral Density (dB/Hz)
65
45
25
5
−15
−35
0
5
10
15
20
25
30
Frequency
closed
opened

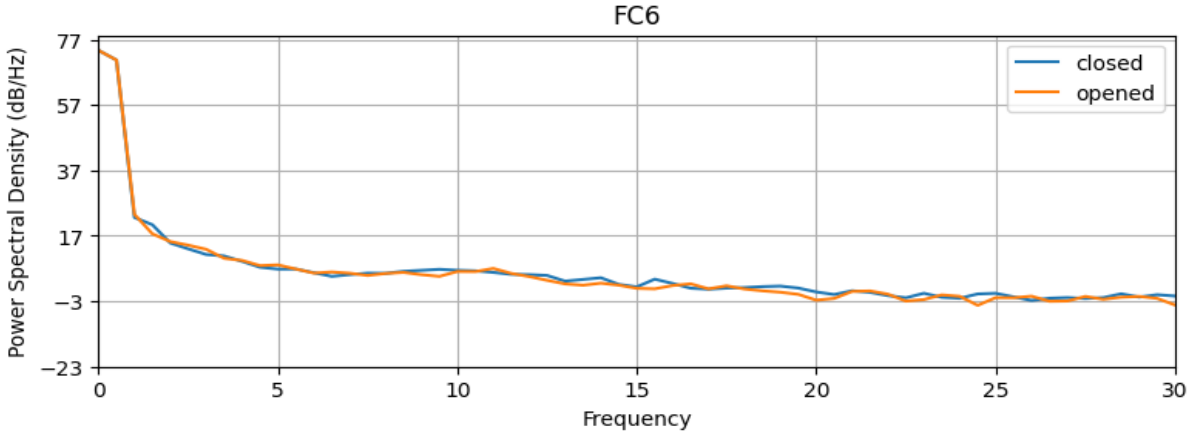

P7
Power Spectral Density (dB/Hz)
67
47
27
7
−13
−33
0
5
10
15
20
25
30
Frequency
closed
opened

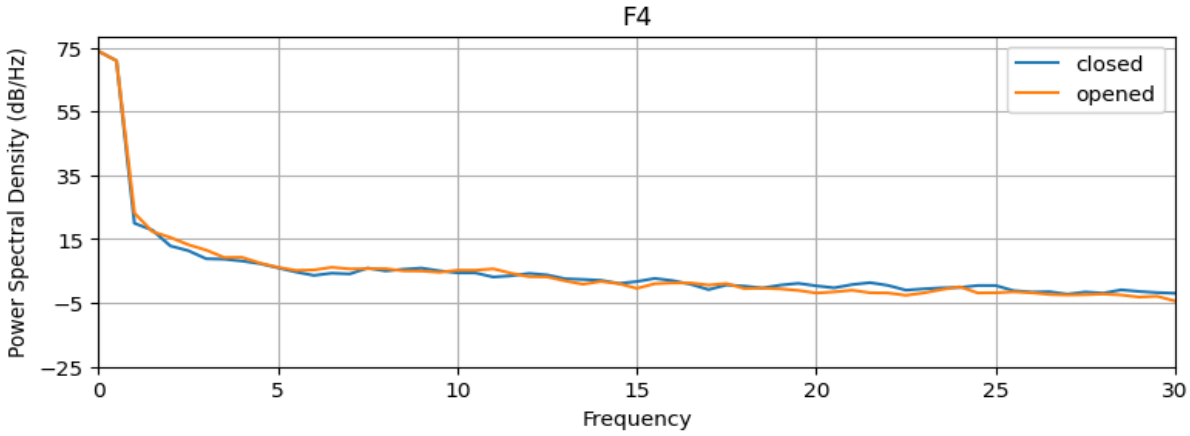

O1
Power Spectral Density (dB/Hz)
67
47
27
7
−13
−33
0
5
10
15
20
25
30
Frequency
closed
opened


O2
Power Spectral Density (dB/Hz)
66
46
26
6
−14
−34
0
5
10
15
20
25
30
Frequency
closed
opened


P8
Power Spectral Density (dB/Hz)
69
49
29
9
−11
−31
0
5
10
15
20
25
30
Frequency
closed
opened


T8
Power Spectral Density (dB/Hz)
72
52
32
12
−8
−28
0
5
10
15
20
25
30
Frequency
closed
opened


FC6
Power Spectral Density (dB/Hz)
77
57
37
17
−3
−23
0
5
10
15
20
25
30
Frequency
closed
opened


F4
Power Spectral Density (dB/Hz)
75
55
35
15
−5
−25
0
5
10
15
20
25
30
Frequency
closed
opened

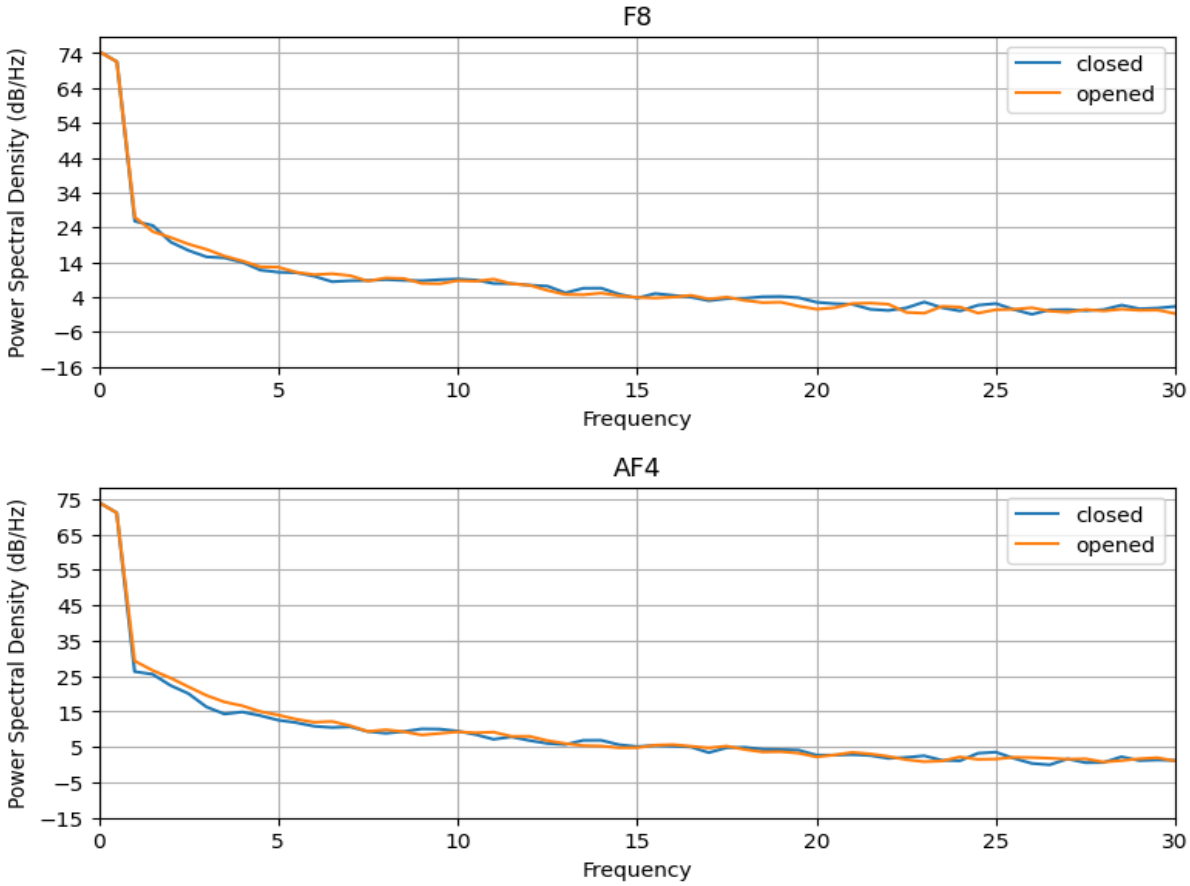


Figure 5: A power spectrum of all the channels.

This means that there is probably no possible way to accurately detect closed eyes. The only possibility would be by detecting blinks. Unfortunately, the data does not contain enough blinks to train a model to do this.
Although the initial objective of predicting whether someone's eyes are opened or closed in this dataset is not achieved, a potentially more significant accomplishment is achieved. We have successfully determined whether it is even possible to predict the state of someone's eyes (open or closed) on a certain dataset.

Another dataset

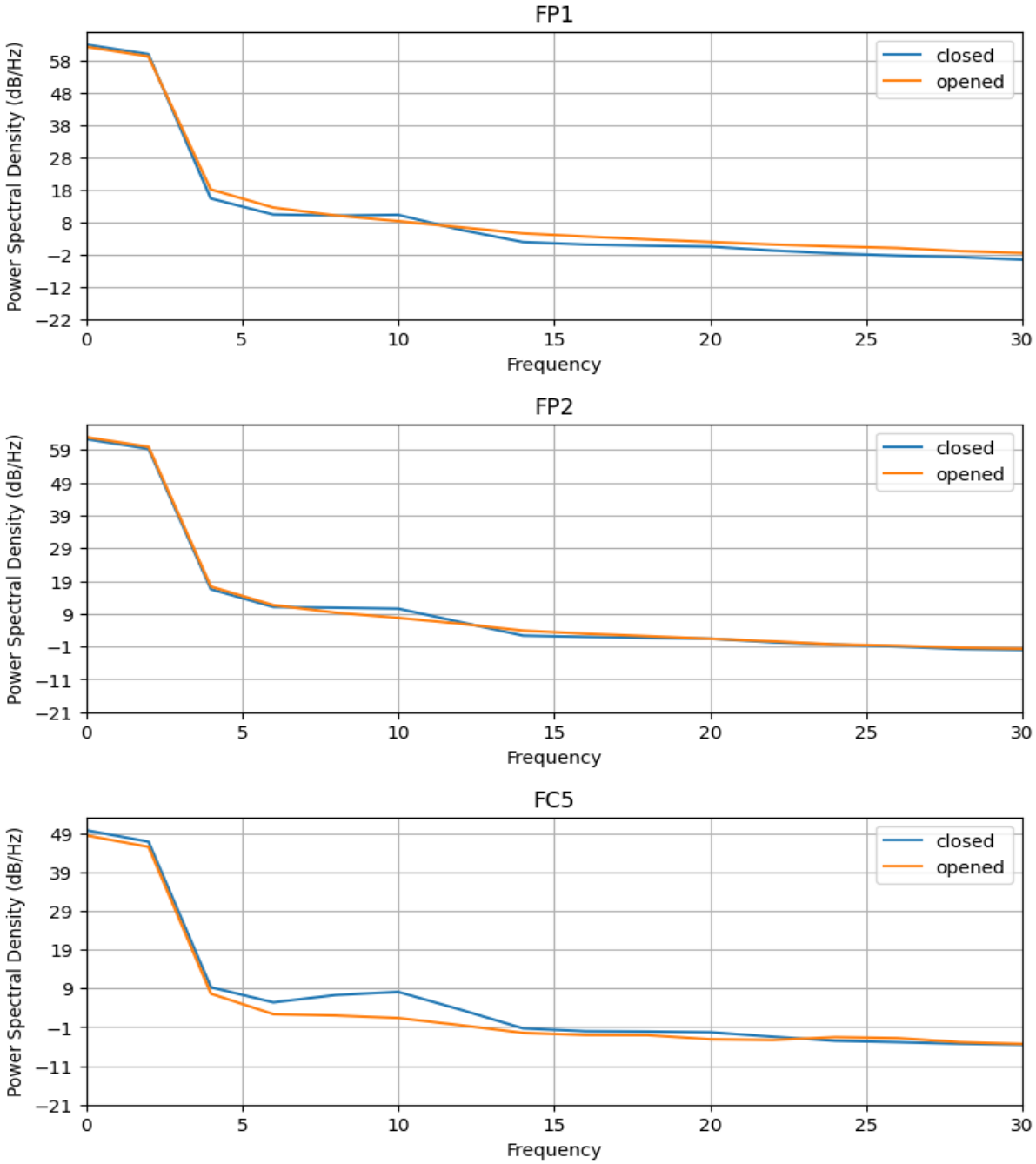
Because the Kaggle.com dataset is not very usable for eye state classification, we used another dataset, found on a list of publicly available datasets: EEG Alpha Waves dataset.[33] With this dataset, the models did work. This is because there clearly was alpha spindling:

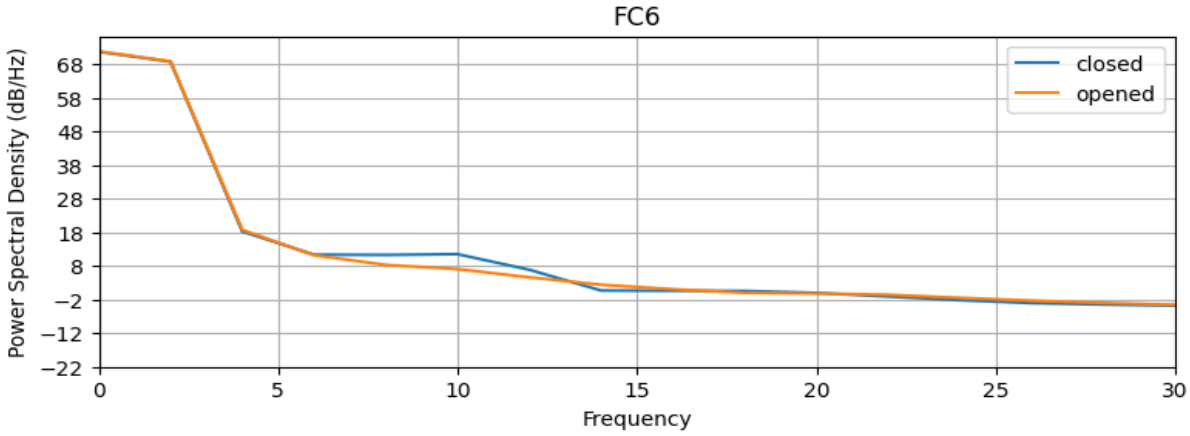
```
closed = df.filter(pl.col("eyestate") == 1).drop(["eyestate", "time"])
opened = df.filter(pl.col("eyestate") == 0).drop(["eyestate", "time"])
```

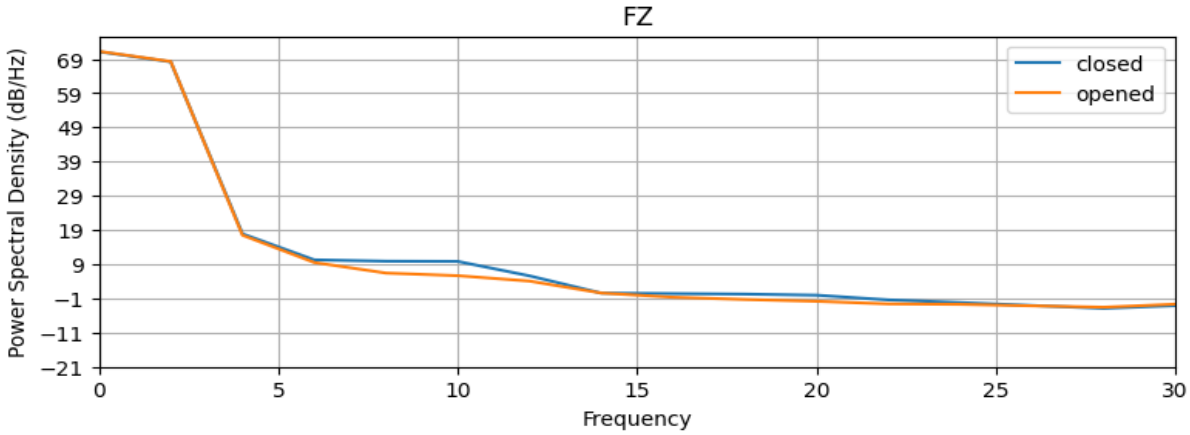
```
for col in closed.columns:
    plt.figure(figsize=(9,3))
    plt.title(col)
    plt.psd(closed.get_column(col), Fs=512)
```

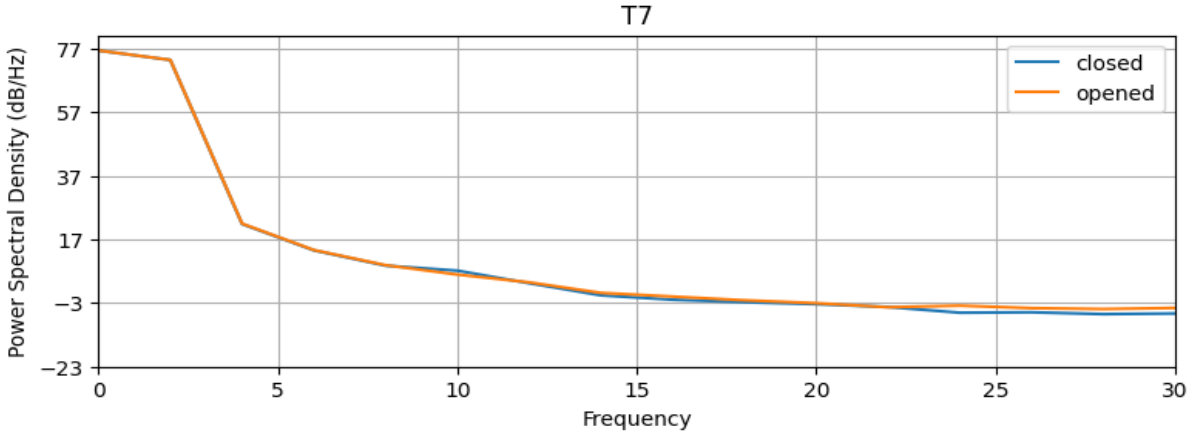
[33] Grégoire Cattan, Pedro L. C. Rodrigues, and Marco Congedo, 'EEG Alpha Waves Dataset' (Zenodo, 17 December 2018), https://doi.org/10.5281/zenodo.2348892; Mohit Agarwal, 'Meagmohit/EEG-Datasets', 24 February 2024, https://github.com/meagmohit/EEG-Datasets.

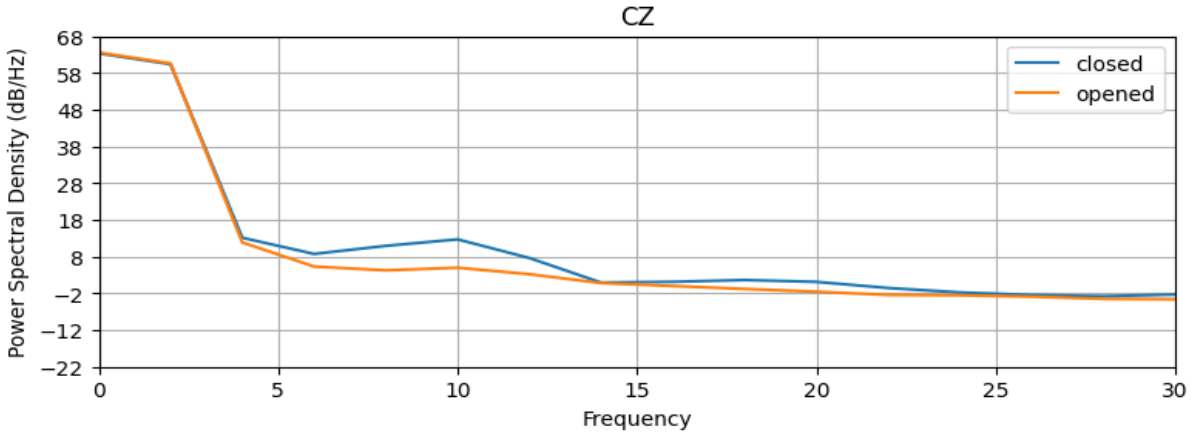
```
    plt.psd(opened.get_column(col), Fs=512)
    plt.xlim([0, 30])
    plt.legend(["closed", "opened"])
```

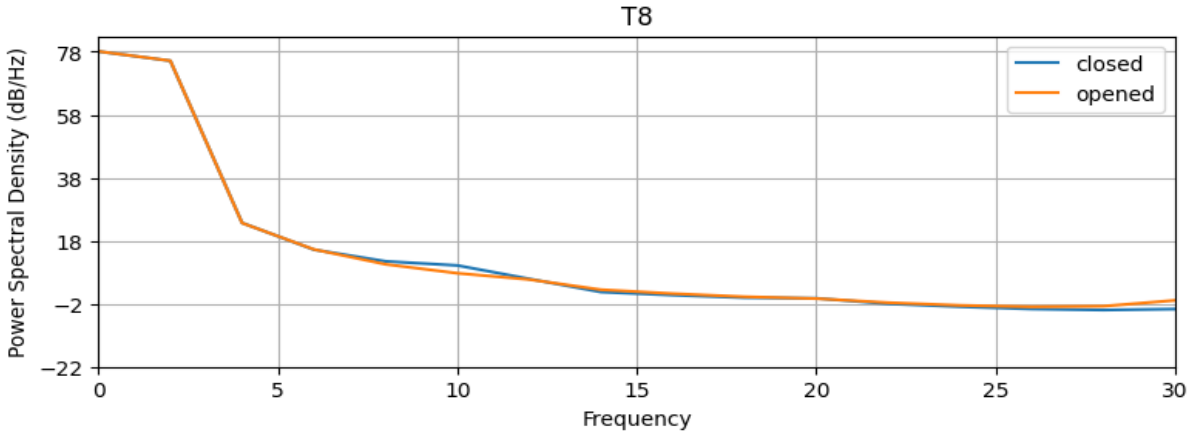


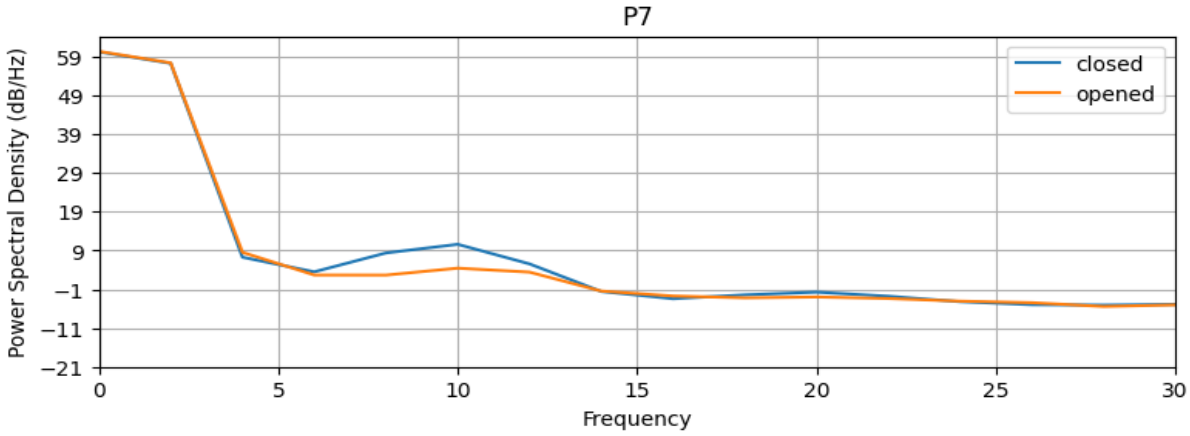

FC6
Power Spectral Density (dB/Hz)
68
58
48
38
28
18
8
−2
−12
−22
0
5
10
15
20
25
30
Frequency
closed
opened

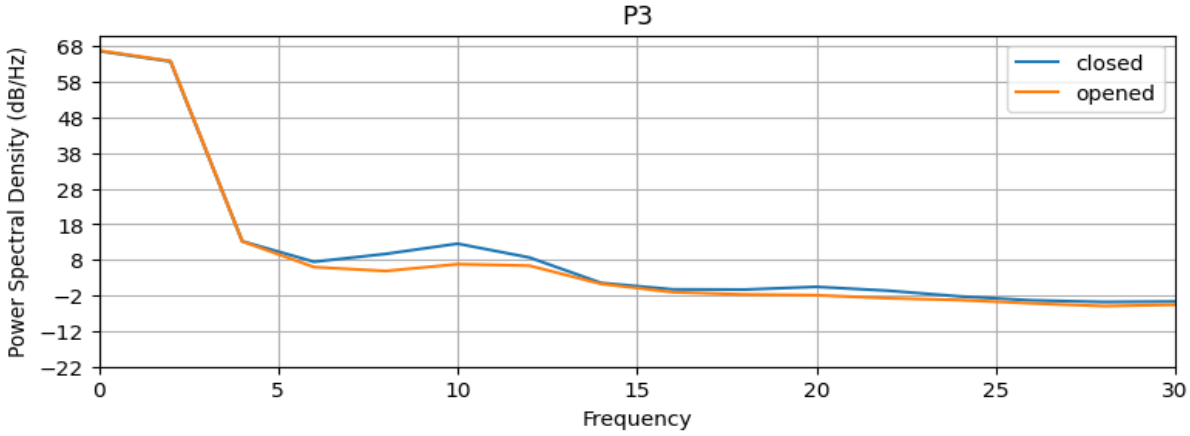

FZ
Power Spectral Density (dB/Hz)
69
59
49
39
29
19
9
−1
−11
−21
0
5
10
15
20
25
30
Frequency
closed
opened

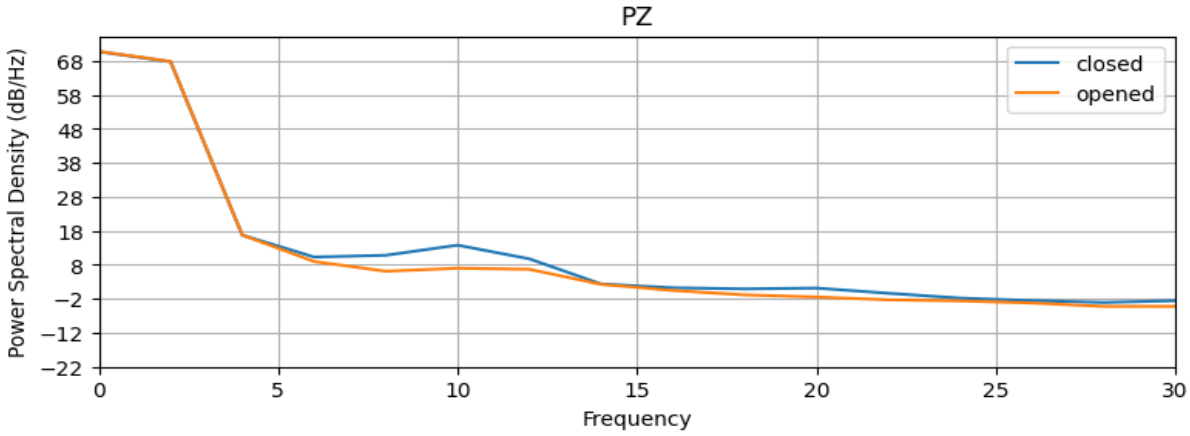

T7
Power Spectral Density (dB/Hz)
77
57
37
17
−3
−23
0
5
10
15
20
25
30
Frequency
closed
opened

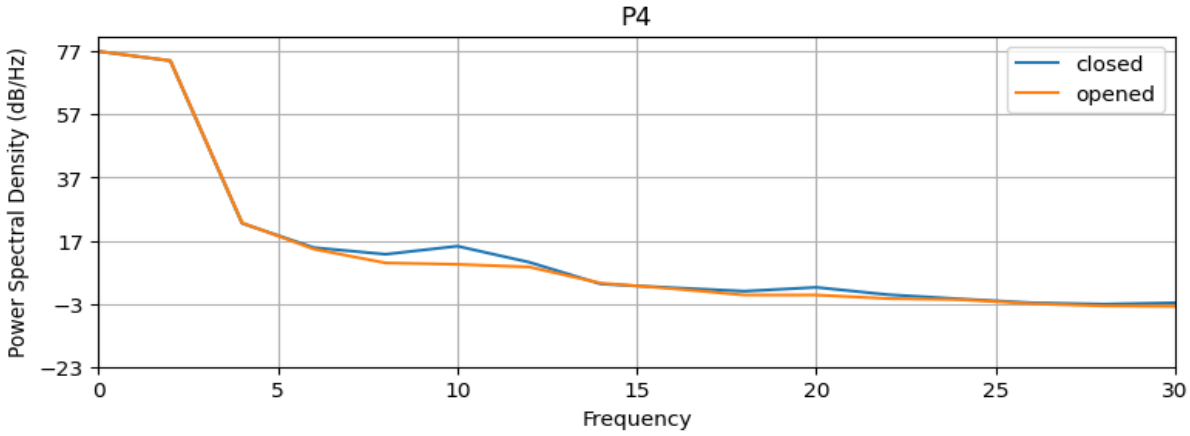

CZ
Power Spectral Density (dB/Hz)
68
58
48
38
28
18
8
−2
−12
−22
0
5
10
15
20
25
30
Frequency
closed
opened

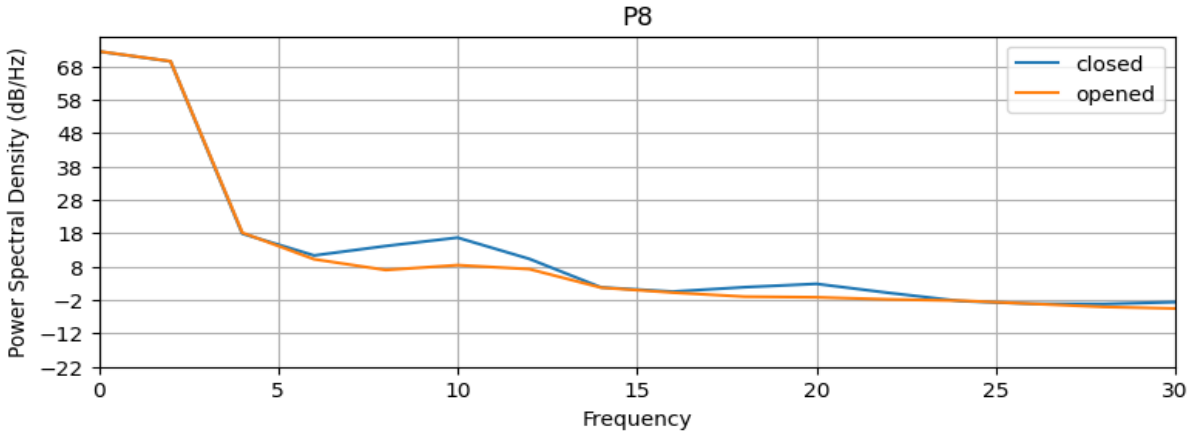

T8
Power Spectral Density (dB/Hz)
78
58
38
18
−2
−22
0
5
10
15
20
25
30
Frequency
closed
opened

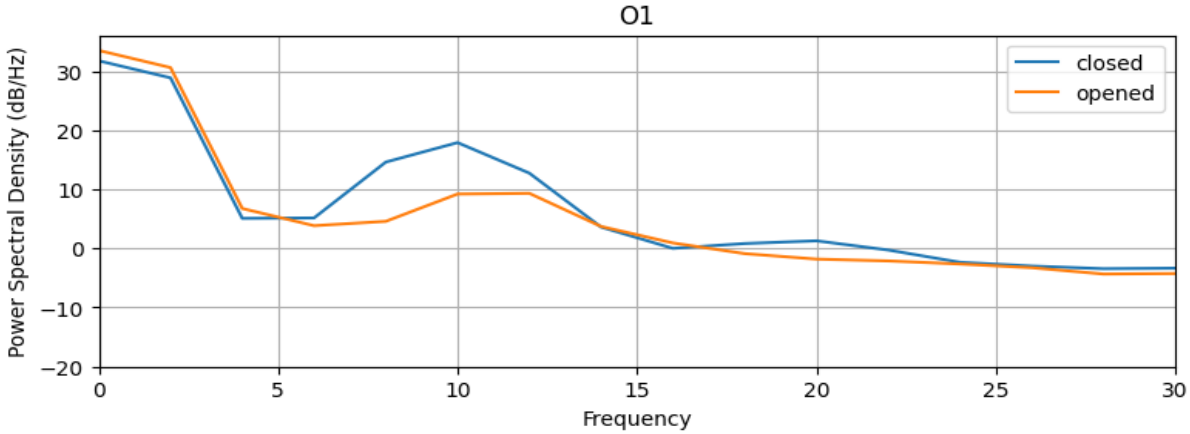

P7
Power Spectral Density (dB/Hz)
59
49
39
29
19
9
−1
−11
−21
0
5
10
15
20
25
30
Frequency
closed
opened

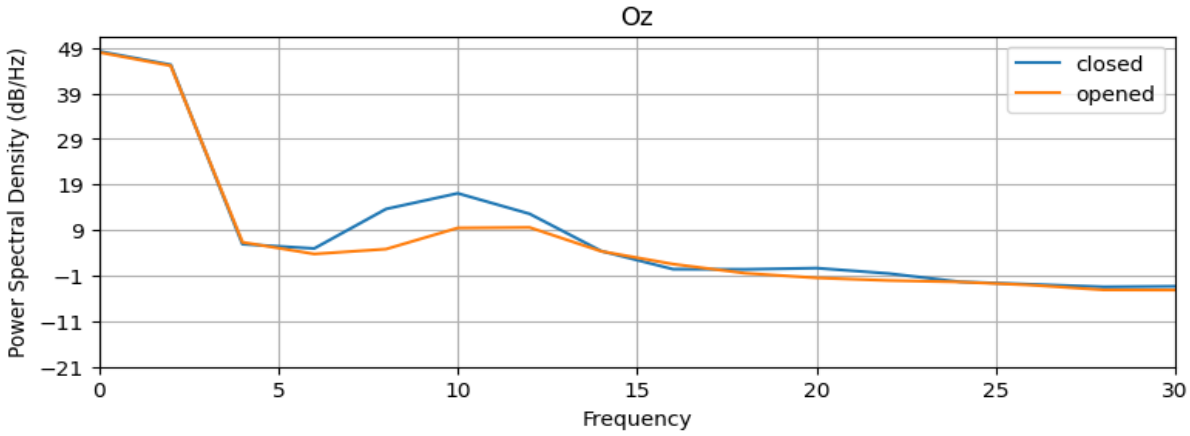

P3
Power Spectral Density (dB/Hz)
68
58
48
38
28
18
8
−2
−12
−22
0
5
10
15
20
25
30
Frequency
closed
opened


PZ
Power Spectral Density (dB/Hz)
68
58
48
38
28
18
8
−2
−12
−22
0
5
10
15
20
25
30
Frequency
closed
opened


P4
Power Spectral Density (dB/Hz)
77
57
37
17
−3
−23
0
5
10
15
20
25
30
Frequency
closed
opened


P8
Power Spectral Density (dB/Hz)
68
58
48
38
28
18
8
−2
−12
−22
0
5
10
15
20
25
30
Frequency
closed
opened


O1
Power Spectral Density (dB/Hz)
30
20
10
0
−10
−20
0
5
10
15
20
25
30
Frequency
closed
opened


Oz
Power Spectral Density (dB/Hz)
49
39
29
19
9
−1
−11
−21
0
5
10
15
20
25
30
Frequency
closed
opened

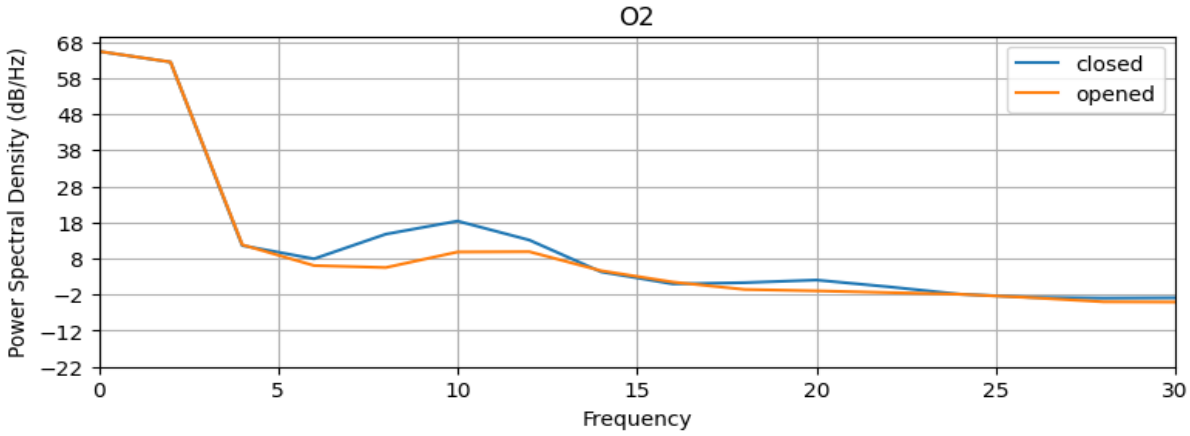


Figure 6: The power spectrum of all the channels from the second dataset.

KNN

Imports, loading in, and processing the data – the same process as above:

```
import scipy
import polars as pl
from scipy import stats
from sklearn import neighbors
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import BaggingRegressor
import math
from sklearn.metrics import accuracy_score, mean_squared_error, roc_auc_sc
ore

from sklearnex import patch_sklearn

patch_sklearn() # this only works for intel devices
```

```
--------------------------------------------------------------------------------

Intel(R) Extension for Scikit-learn* enabled (https://github.com/intel/sci
kit-learn-intelex)

--------------------------------------------------------------------------------
```

```
def eye_state(open_, closed, last):
    if open_:
        return 0
    elif closed:
        return 1
    else:
        return last
```

```
--------------------------------------------------------------------------------
```

```
data = scipy.io.loadmat("subject_00.mat")
df = pl.from_numpy(data["SIGNAL"], schema=["time", "FP1", "FP2", "FC5", "F
C6", "FZ", "T7", "CZ", "T8", "P7", "P3", "PZ", "P4", "P8", "O1", "Oz",  "O
2", "close", "open"])
```

```
last = 0
eye = []
for row in df.rows(named=True):
    last = eye_state(row["open"], row["close"], last)
    eye.append(last)

df = df.with_columns(pl.Series("eyestate", eye))
df = df.drop(["close", "open"])
df.head()
```

---

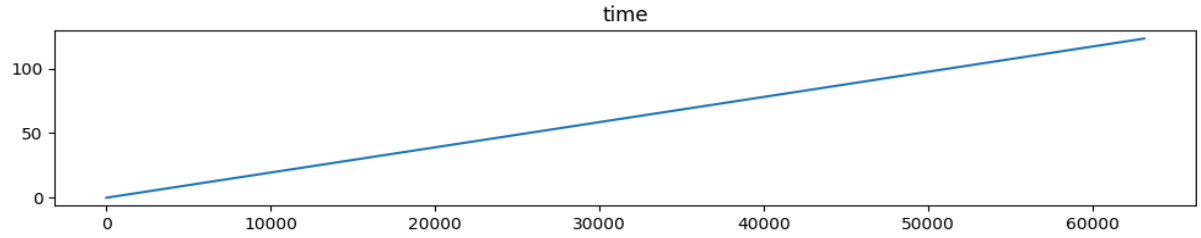
shape: (5, 18)

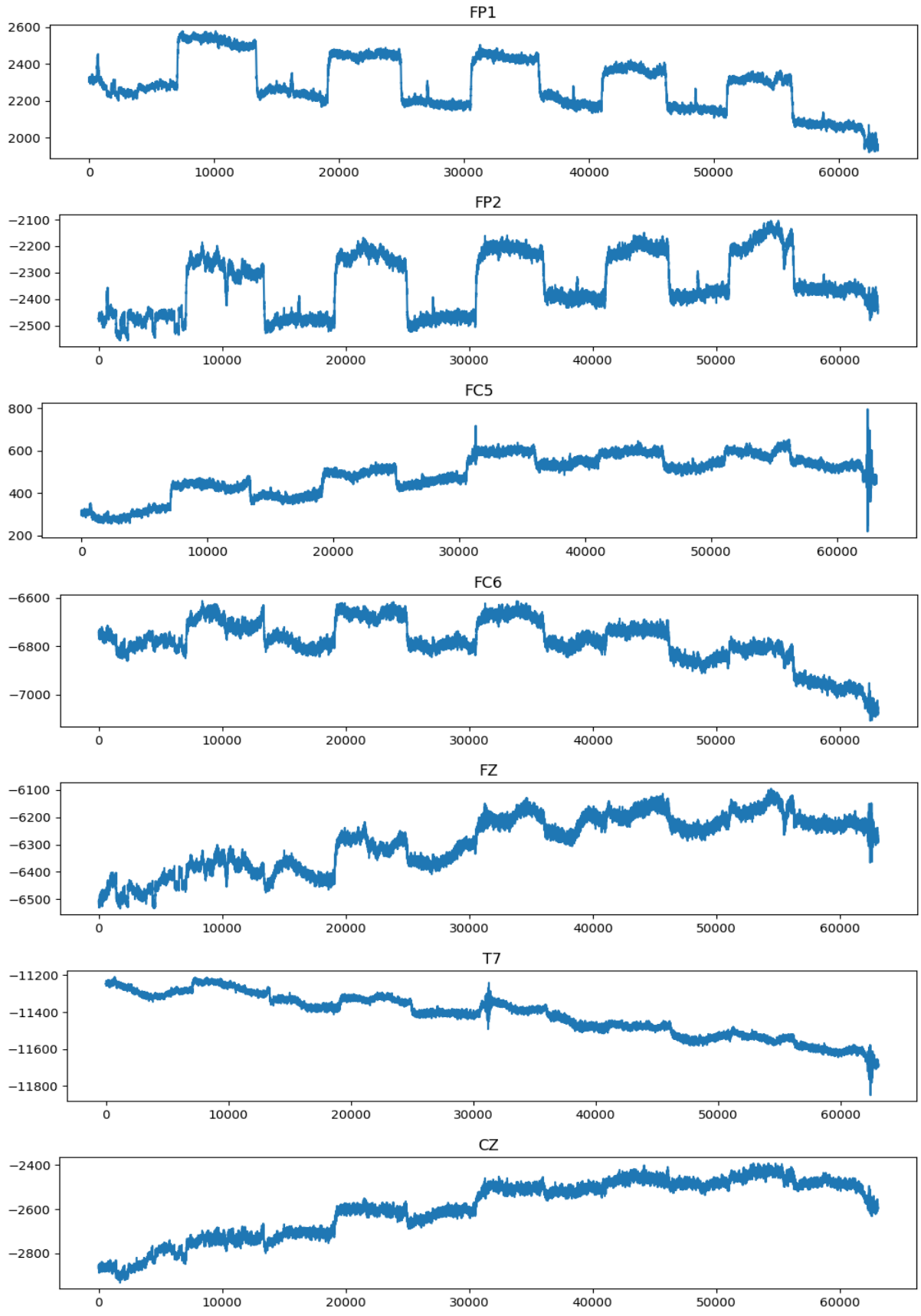
time	FP1	FP2	FC5	…	O1	Oz	O2	eyestate
f64	f64	f64	f64		f64	f64	f64	i64
0.0	2314.7109 38	-2473.864 258	304.72659 3	…	170.43650 8	397.11026	-3278.888 184	0
0.001953	2312.0773 93	-2477.192 871	295.27121	…	175.32383 7	400.19650 3	-3276.674 561	0
0.003906	2308.1391 6	-2477.75	292.86230 5	…	193.75805 7	410.23248 3	-3263.133 545	0
0.005859	2303.1752 93	-2477.378 174	296.50705	…	197.83984 4	410.05499 3	-3260.899 902	0
0.0078125	2307.3315 43	-2471.294 434	301.83157 3	…	200.90509	415.01074 2	-3255.357 422	0

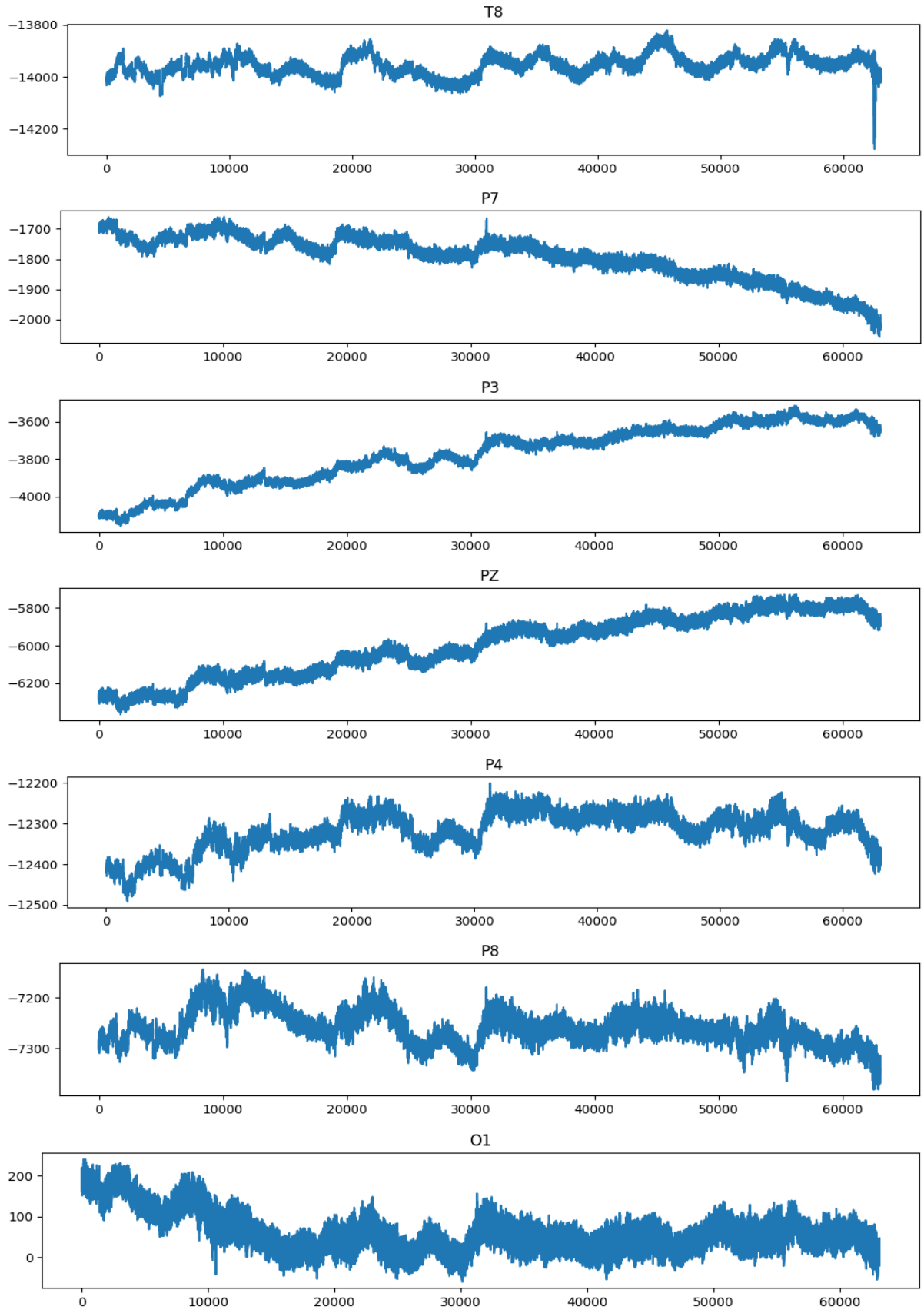
```
# removing outliers
for col in df.columns:
    zscores =(np.abs(stats.zscore(df.get_column(col))) < 5)
    df = df.filter(zscores)
```

Visualizing the data ('eyestate' is 0 when eyes are open and 1 when eyes are closed):

```
for col in df.columns:
    plt.figure(figsize=(12,2))
    plt.title(col)
    y = df.get_column(col)
    plt.plot(range(len(y)), y)
```




FP1
2600
2400
2200
2000
0
10000
20000
30000
40000
50000
60000
FP2
−2100
−2200
−2300
−2400
−2500
FC5
800
600
400
200
FC6
−6600
−6800
−7000
FZ
−6100
−6200
−6300
−6400
−6500
T7
−11200
−11400
−11600
−11800
CZ
−2400
−2600
−2800


T8
−13800
−14000
−14200
0
10000
20000
30000
40000
50000
60000
P7
−1700
−1800
−1900
−2000
P3
−3600
−3800
−4000
PZ
−5800
−6000
−6200
P4
−12200
−12300
−12400
−12500
P8
−7200
−7300
O1
200
100
0

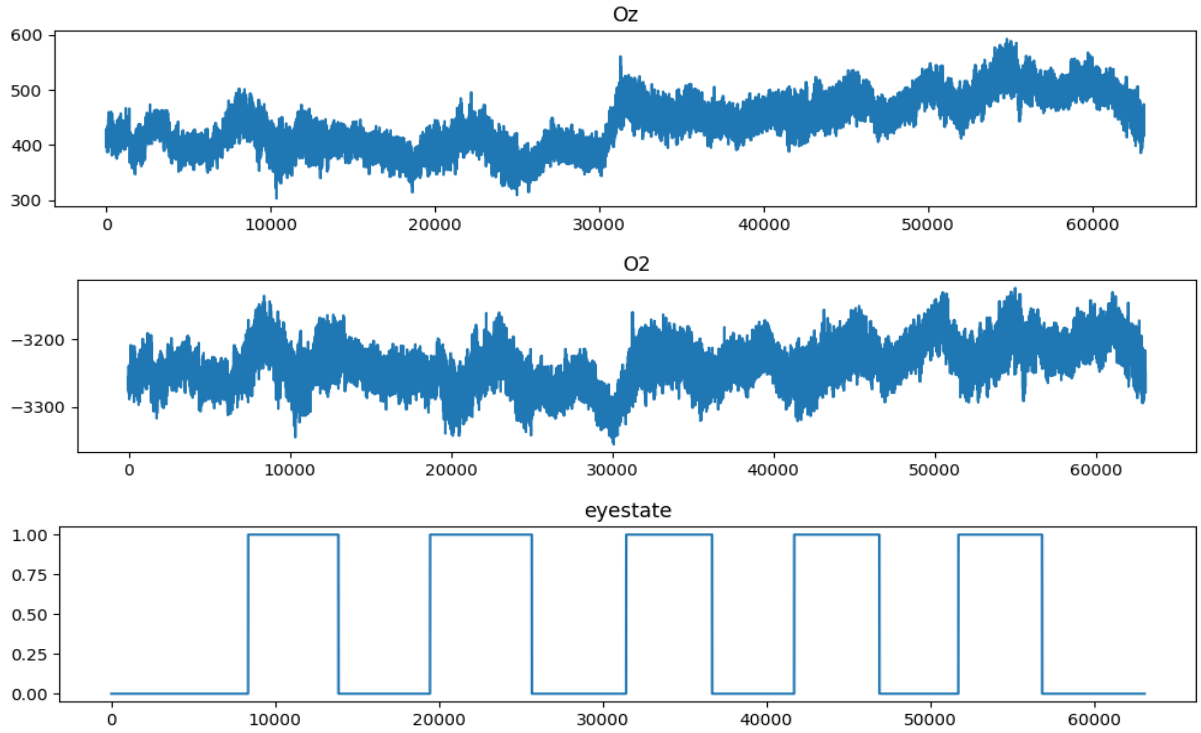


Figure 7: The visualized columns of the dataset.

Splitting the data, hyperparameter tuning, training the model, and testing the model:

```
x_train, x_test, y_train, y_test = train_test_split(
    df.drop(["eyestate", "time"]).to_numpy(),
    df.select("eyestate").to_numpy(),
    test_size=0.2,
    shuffle=False
)

params = {"n_neighbors": range(1, 50), "weights": ["uniform", "distance"]}


best_params = (
    GridSearchCV(neighbors.KNeighborsClassifier(), params)
    .fit(x_train, y_train.ravel())
    .best_params_
)
best_params

--------------------------------------------------------------------------

{'n_neighbors': 1, 'weights': 'uniform'}

--------------------------------------------------------------------------

model = BaggingRegressor(neighbors.KNeighborsClassifier(**best_params), n_
estimators=10)
model.fit(x_train, y_train.ravel())
y_pred =  model.predict(x_test)
```

```
rmse_train = math.sqrt(mean_squared_error(model.predict(x_train), y_train)
)
acc = accuracy_score(y_test, np.int32(y_pred))
rmse_test = math.sqrt(mean_squared_error(y_test, y_pred))

rmse_train, rmse_test, acc

--------------------------------------------------------------------------

(0.032605676846662816, 0.3707279172841329, 0.8492195547104033)
```

The KNN model is able to achieve an accuracy of 85%.

SVM
The parameters were:

```
best_params_loose
{'C': 0.125, 'gamma': 0.125}

best_params_fine
{'C': 0.21022410381342863, 'gamma': 0.0625}
```

And it was able to achieve an accuracy of 88%:

```
rmse_train, rmse_test, acc

(0.3363228590591355, 0.2950160118360779, 0.8817843277077886)
```

Neural network
Lastly, we trained a neural network using TensorFlow and Keras. It is actually a common approach to first train simpler models such as Random Forest, K Nearest Neighbors, and Support Vector Machines to find out where the upper bound of accuracy may lie. Then optimising that accuracy by training a neural network. For an explanation of all the concepts, we recommend Sentdex' YouTube tutorial series: Deep Learning basics with Python, TensorFlow and Keras.[34]

The model consists of an input layer, three hidden layers (the first model.add() adds both the input and the first hidden layer), and an output layer. The hidden layers use the Rectified Linear Unit (ReLU) activation function, and the output layer uses a sigmoid activation function.[35] The number of nodes (12, 24, and 16) were selected after testing a lot of different options. We used the binary cross entropy loss function and the Adam optimizer.[36]

[34] *Deep Learning with Python, TensorFlow, and Keras Tutorial*, 2018, https://www.youtube.com/watch?v=wQ8BIBpya2k.
[35] Jason Brownlee, 'A Gentle Introduction to the Rectified Linear Unit (ReLU)', *MachineLearningMastery.Com* (blog), 8 January 2019, https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/.
[36] Daniel Godoy, 'Understanding Binary Cross-Entropy / Log Loss: A Visual Explanation', Medium, 10 July 2022, https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a; Jason Brownlee, 'Gentle Introduction to the Adam Optimization Algorithm for Deep Learning', *MachineLearningMastery.Com* (blog), 2 July 2017, https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/.

```
import scipy
import polars as pl
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

-------------------------------------------------------------------------

model = Sequential()
model.add(Dense(12, input_shape=(16,), activation='relu'))
model.add(Dense(24, activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
```

And training and testing the model, we trained 20 epochs:

```
model.fit(x_train, y_train, epochs=20, batch_size=None)

-------------------------------------------------------------------------

Epoch 1/20
1578/1578 [==============================] - 20s 10ms/step - loss: 10.0875
- accuracy: 0.6843
Epoch 2/20
1578/1578 [==============================] - 13s 8ms/step - loss: 3.6608 -
accuracy: 0.7650
Epoch 3/20
1578/1578 [==============================] - 12s 7ms/step - loss: 3.6189 -
accuracy: 0.7754
Epoch 4/20
1578/1578 [==============================] - 9s 6ms/step - loss: 2.9421 -
accuracy: 0.7945
Epoch 5/20
1578/1578 [==============================] - 11s 7ms/step - loss: 2.4967 -
accuracy: 0.7918
Epoch 6/20
1578/1578 [==============================] - 12s 8ms/step - loss: 2.0254 -
accuracy: 0.7946
Epoch 7/20
1578/1578 [==============================] - 12s 7ms/step - loss: 1.3851 -
accuracy: 0.7858
Epoch 8/20
1578/1578 [==============================] - 13s 8ms/step - loss: 1.0410 -
accuracy: 0.7943
Epoch 9/20
1578/1578 [==============================] - 11s 7ms/step - loss: 0.6881 -
accuracy: 0.8216
Epoch 10/20
1578/1578 [==============================] - 13s 8ms/step - loss: 0.8035 -
accuracy: 0.8143
Epoch 11/20
1578/1578 [==============================] - 9s 6ms/step - loss: 0.5723 -
```

```
accuracy: 0.8237
Epoch 12/20
1578/1578 [==============================] - 7s 4ms/step - loss: 0.4594 -
accuracy: 0.8403
Epoch 13/20
1578/1578 [==============================] - 7s 4ms/step - loss: 0.4171 -
accuracy: 0.8539
Epoch 14/20
1578/1578 [==============================] - 7s 5ms/step - loss: 0.3847 -
accuracy: 0.8676
Epoch 15/20
1578/1578 [==============================] - 6s 4ms/step - loss: 0.3680 -
accuracy: 0.8750
Epoch 16/20
1578/1578 [==============================] - 13s 8ms/step - loss: 0.3642 -
accuracy: 0.8772
Epoch 17/20
1578/1578 [==============================] - 8s 5ms/step - loss: 0.3649 -
accuracy: 0.8779
Epoch 18/20
1578/1578 [==============================] - 7s 5ms/step - loss: 0.3872 -
accuracy: 0.8546
Epoch 19/20
1578/1578 [==============================] - 8s 5ms/step - loss: 0.3653 -
accuracy: 0.8763
Epoch 20/20
1578/1578 [==============================] - 7s 5ms/step - loss: 0.3572 -
accuracy: 0.8805

<keras.src.callbacks.History at 0x149dfe050>

------------------------------------------------------------------------

model.evaluate(x_test, y_test)

------------------------------------------------------------------------

99/99 [==============================] - 1s 8ms/step - loss: 0.3023 - accu
racy: 0.9032

[0.3022618293762207, 0.9031772613525391]
```

The neural network achieves the highest accuracy, 90%.

5. Discussion

5.1 More electrodes

The EEG machine that we made was a basic EEG machine. It had just a single output, the difference between the O2 and Fp2 electrode. Therefore, the most straight-forward way to improve our project is by using more electrodes and analysing multiple signals, getting a lot more data from other regions of the brain.

ICA
Using more electrodes would allow for the application of a technique called Independent Component Analysis (ICA).[37] Although our circuit and software are already capable of filtering out noise from for instance the skin or the power lines, it cannot filter out signals unrelated to the brain waves of interest. ICA allows not only for the filtering of signals caused by muscles firing, blinks, and heart noise, but also allows for the separation of mixtures of brain activity. ICA is a technique used in signal processing to separate independent sources that have been combined linearly across multiple sensors.

Thought detection
With more electrodes it would even be possible to detects certain thoughts: Playing Video Games With Mind Control - YouTube.[38] A neural network could be trained to detect for instance the visualization of pushing an object forward or of an object sinking in water.

5.2 Radio signal

The raw data collected through the EEG machine can be listened to and is contaminated with a radio signal. The radio signal was clear enough to understand – we were even able to make out a Nivea commercial. We were not able to identify the radio station that sent the radio signal. We think that the bare wire, which did not have isolation, worked as a small antenna. This was only the case whenever the circuit was turned on.

5.3 Beta waves

Lastly, this circuit should also be able to capture beta waves (12-30 Hz), which work the opposite of alpha waves. They are increased when the subject concentrates and decreased when the subject is relaxed. We would like to experiment with these in the future.

[37] *ICA Applied to EEG: What Is ICA?*, accessed 24 November 2023, https://www.youtube.com/watch?v=kWAjhXr7pT4&list=PLXc9qfVbMMN2uDadxZ_OEsHjzcRtlLNxc&index=2.
[38] *Playing Video Games With Mind Control*, 2023, https://www.youtube.com/watch?v=DBYY3D1gkQ0.

6. Bibliography

3Blue1Brown (Director). (2018, January 26). *But what is the Fourier Transform? A visual introduction.* https://www.youtube.com/watch?v=spUNpyF58BY

10–20 system (EEG). (2023). In *Wikipedia*. https://en.wikipedia.org/w/index.php?title=10%E2%80%9320_system_(EEG)&oldid=1144970353

Analog-to-digital converter. (2023). In *Wikipedia*. https://en.wikipedia.org/w/index.php?title=Analog-to-digital_converter&oldid=1179394589

Berg, O. (2023). *IkBenOlie5/EEG* [Jupyter Notebook]. https://github.com/IkBenOlie5/EEG (Original work published 2023)

cah6. (n.d.). *DIY EEG (and ECG) Circuit*. Instructables. Retrieved 26 November 2023, from https://www.instructables.com/DIY-EEG-and-ECG-Circuit/

Changpuak, A. C. F., aka. (n.d.). *Online Engineering Calculator: Active Twin - T - Notch Filter Calculator*. Www.Changpuak.Ch. Retrieved 26 November 2023, from https://www.changpuak.ch/electronics/Active_Notch_Filter.php

chipstein—Building the Amplifier. (n.d.). Retrieved 26 November 2023, from https://sites.google.com/site/chipstein/homebrew-do-it-yourself-eeg-ekg-and-emg/building-the-amplifier

Dahl, Ø. N. (2020, September 18). *What is Ground in Electronic Circuits?* Build Electronic Circuits. https://www.build-electronic-circuits.com/what-is-ground/

EEG (Electroencephalography): The Complete Pocket Guide - iMotions. (2019, August 27). https://imotions.com/blog/learning/best-practice/eeg/

Einthoven's triangle. (2023). In *Wikipedia*. https://en.wikipedia.org/w/index.php?title=Einthoven%27s_triangle&oldid=1186304522

electronzapdotcom (Director). (2020, August 10). *Electronics single versus dual or split power supply explained*. https://www.youtube.com/watch?v=jiWKQEOSgIs

Eye State Classification EEG Dataset. (n.d.). Retrieved 24 November 2023, from https://www.kaggle.com/datasets/robikscube/eye-state-classification-eeg-dataset

Frequency ratio method to detect alpha spindles. (n.d.). Retrieved 24 February 2024, from https://kaggle.com/code/pochenliu/frequency-ratio-method-to-detect-alpha-spindles

How to tune the K-Nearest Neighbors classifier with Scikit-Learn in Python. (2020, January 28). DataSklr. https://www.datasklr.com/select-classification-methods/k-nearest-neighbors

Hsu, C.-W., Chang, C.-C., & Lin, C.-J. (n.d.). *A Practical Guide to Support Vector Classification*.

ICA applied to EEG: What is ICA? (n.d.). Retrieved 24 November 2023, from https://www.youtube.com/watch?v=kWAjhXr7pT4&list=PLXc9qfVbMMN2uDadxZ_OEsHjzcRtlLNxc&index=2

Lawhern, V., Kerick, S., & Robbins, K. A. (2013). Detecting alpha spindle events in EEG time series using adaptive autoregressive models. *BMC Neuroscience*, *14*, 101. https://doi.org/10.1186/1471-2202-14-101

Leske, S., & Dalal, S. S. (2019). Reducing power line noise in EEG and MEG data via spectrum interpolation. *NeuroImage*, *189*, 763–776. https://doi.org/10.1016/j.neuroimage.2019.01.026

Lopez, R. (2023). *EEG* [Python]. https://github.com/ryanlopezzzz/EEG (Original work published 2021)

Netspanning. (2024). In *Wikipedia*. https://nl.wikipedia.org/w/index.php?title=Netspanning&oldid=67012357

No alpha spindling when eyes are closed. (n.d.). Retrieved 24 February 2024, from https://kaggle.com/code/pochenliu/no-alpha-spindling-when-eyes-are-closed

Notch Filter Calculator. (n.d.). Retrieved 26 November 2023, from https://www.learningaboutelectronics.com/Articles/Notch-filter-calculator.php#answer1

Physics Videos by Eugene Khutoryansky (Director). (2015, September 4). *Resistors—Ohm's Law is not a real law*. https://www.youtube.com/watch?v=G3H5lKoWPpY

Physics Videos by Eugene Khutoryansky (Director). (2016a, February 2). *Op Amp Circuits: Analog Computers from operational amplifiers*. https://www.youtube.com/watch?v=_o4ScgRZtNI

Physics Videos by Eugene Khutoryansky (Director). (2016b, April 26). *Capacitors and Capacitance: Capacitor physics and circuit operation*. https://www.youtube.com/watch?v=f_MZNsEqyQw

Power Electronics Introduction—What is Power Electronics? (n.d.). Retrieved 24 November 2023, from https://www.youtube.com/watch?v=hRAyfJLZnC0&list=PLmK1EnKxphinxBub5hL0ZoJXWoqjkGE19

Python, R. (n.d.). *Fourier Transforms With scipy.fft: Python Signal Processing – Real Python*. Retrieved 24 November 2023, from https://realpython.com/python-scipy-fft/

Reese, L. (2020, May 5). *What is signal attenuation?* Analog IC Tips. https://www.analogictips.com/what-is-signal-attenuation/

sentdex (Director). (2016, April 11). *Practical Machine Learning Tutorial with Python Intro p.1*. https://www.youtube.com/watch?v=OGxgnH8y2NM

Sklearn.model_selection.TimeSeriesSplit. (n.d.). Scikit-Learn. Retrieved 24 February 2024, from https://scikit-learn/stable/modules/generated/sklearn.model_selection.TimeSeriesSplit.html

Storr, W. (2013a, August 14). Active High Pass Filter—Op-amp High Pass Filter. *Basic Electronics Tutorials*. https://www.electronics-tutorials.ws/filter/filter_6.html

Storr, W. (2013b, August 14). Active Low Pass Filter—Op-amp Low Pass Filter. *Basic Electronics Tutorials*. https://www.electronics-tutorials.ws/filter/filter_5.html

Storr, W. (2013c, August 14). Capacitive Reactance—The Reactance of Capacitors. *Basic Electronics Tutorials*. https://www.electronics-tutorials.ws/filter/filter_1.html

Storr, W. (2013d, August 14). High Pass Filter—Passive RC Filter Tutorial. *Basic Electronics Tutorials*. https://www.electronics-tutorials.ws/filter/filter_3.html

Storr, W. (2013e, August 14). Low Pass Filter—Passive RC Filter Tutorial. *Basic Electronics Tutorials*. https://www.electronics-tutorials.ws/filter/filter_2.html

Storr, W. (2015, October 20). Band Stop Filter and Notch Filter Design Tutorial. *Basic Electronics Tutorials*. https://www.electronics-tutorials.ws/filter/band-stop-filter.html

Storr, W. (2021, October 7). Sallen and Key Filter Design for Second Order RC Filters. *Basic Electronics Tutorials*. https://www.electronics-tutorials.ws/filter/sallen-key-filter.html

Texas Instruments (Director). (2020, January 7). *Introduction to instrumentation amplifiers*. https://www.youtube.com/watch?v=NvyDw8ZpLd0

Twin-T Filter. (n.d.). Retrieved 26 November 2023, from https://www.falstad.com/circuit/e-twint.html

Vansteensel, M. J., Pels, E. G. M., Bleichner, M. G., Branco, M. P., Denison, T., Freudenburg, Z. V., Gosselaar, P., Leinders, S., Ottens, T. H., Van Den Boom, M. A., Van Rijen, P. C., Aarnoutse, E. J., & Ramsey, N. F. (2016). Fully Implanted Brain–Computer Interface in a Locked-In Patient with ALS. *New England Journal of Medicine*, *375*(21), 2060–2066. https://doi.org/10.1056/NEJMoa1608085

What Are the Different Types of Brain Waves? (n.d.). Retrieved 25 February 2024, from https://info.tmsi.com/blog/types-of-brain-waves