This document details the full process for using the muse EEG headband with Psychopy.

Credits

This project is a collaboration between Sandra Nitchi, Lisa Zubkova and Natali Petrosyan
This project's work was overseen by Toky Raharison Ralambomihanta
This project was organized by Prof. Helene Nadeau and Prof. Sylvia Cox
This project was completed at Dawson College, Montreal, summer of 2025
For additional info, review our user guide for combining muse eeg with psychopy or contact any/all of the members of this team.

Sandra Nitchi: sandranitchi@gmail.com
Lisa Zubkova: alisa.zubkova@dawsoncollege.qc.ca
Natali Petrosyan: natalipetrosyan135@gmail.com
Toky Raharison Ralambomihanta: tokiniaina.raharisonralambomihan@mail.mcgill.ca
Prof. Helene Nadeau: HNadeau@dawsoncollege.qc.ca
Prof. Sylvia Cox: scox@dawsoncollege.qc.ca

Step 1: Install psychopy

The psychopy app can be downloaded from their website: https://www.psychopy.org/download.html
Download the most recent stable version for your device

Step 2: Pull up a project you want to use in psychopy

I will be using the default stroop task, but you can use whatever task you are working with. If you already have your task folder set up, feel free to skip to step 3

For using the stroop task, I'll create a folder on desktop and copy the default package into the new folder. After, in the psychopy builder, I have to reselect the csv of the words used in the stoop to be the new correct one. To adjust the csv location: click on the loop (trials) then select the new csv from the right folder.

Step 3: Add custom code to the psychopy builder

Open the psychopy builder. On the right side, under components, click the grey bar that says "custom", then open "code". Switch "auto→JS" to "py"

The following code will add a datetime timestamp whenever a key is pressed:

- **In "begin experiment":**
- import time
- from datetime import datetime # make sure these are in "begin experiment"
- **In "Begin routine':**

- timestamp_logged = False
- **In “Each frame”:**
- if resp.keys and not timestamp_logged:
-     # ISO format (easily readable time)
-     iso_timestamp = datetime.now().isoformat()
-     thisExp.addData('response_timestamp_iso', iso_timestamp)
-
-     # Unix time (the one we use for plotting)
-     unix_timestamp_ms = time.time()
-     thisExp.addData('Marker Timestamp', unix_timestamp_ms)
-
-     timestamp_logged = True
-

Make sure the name of your keyboard response in the builder is “resp” or this wont work. Alternatively, you can rewrite the line of code that says: if resp.keys and not timestamp_logged: and just replace “resp” with whatever the name of your keyboard responder is called.

Now your timestamps are being taken!

Alternatively, you can add code to take a marker at a certain point of the shown stimulus (image or text) instead of at the user's key response. In that case, your code should read:

- **In “Begin experiment”:**
- import time
- from datetime import datetime
- **In “Begin routine”:**
- stim_timestamp_logged = False
- **In “Each frame”:**
- if targetStim.status == STARTED and not stim_timestamp_logged:
-     # ISO timestamp (easily readable time)
-     display_timestamp_iso = datetime.now().isoformat()
-     thisExp.addData('stimulus_timestamp_datetime', display_timestamp_iso)
-
-     # Unix time (the one we use for plotting)
-     display_timestamp_unix = time.time()
-     thisExp.addData('Marker Timestamp', display_timestamp_unix)
-
-     stim_timestamp_logged = True

Again, make sure your stimulus (image or text) is named targetStim, or if not, rewrite the line of code “if targetStim.status == STARTED and not stim_timestamp_logged:” and replace “targetStim” with the name of your stimulus in the builder.

Step 4: Install muselsl

In terminal run:
pip install muselsl

That will install the package. For additional info on the package, see its github:
https://github.com/alexandrebarachant/muse-lsl
Or it's pypi (python package index):
https://pypi.org/project/muselsl/

Step 5: Start recording using muselsl

Open a terminal on your computer. If you haven't already, run:

    pip install muselsl

Now, turn on your muse and bluetooth and run:

    muselsl stream

Once connected, the terminal will say: Connected. Streaming EEG...
(optional) Once it starts streaming, in a new terminal:

    muselsl view

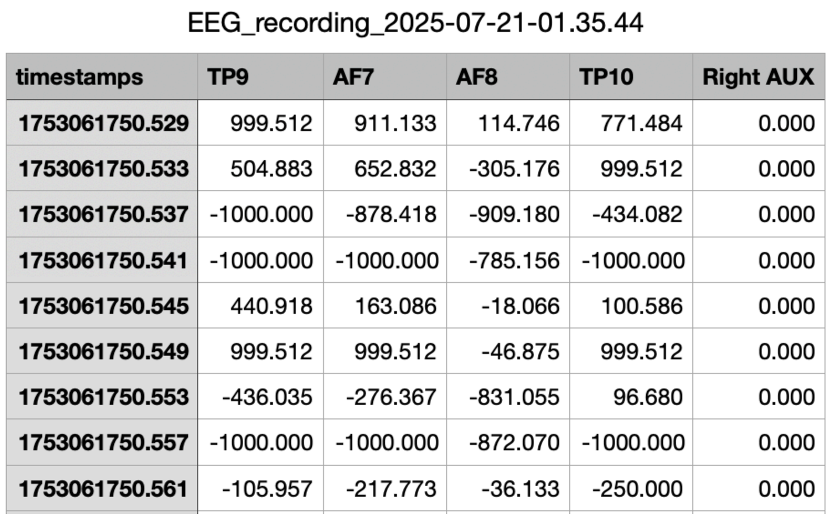
This will show the eeg reading.
Now, in yet another new terminal, run:

`muselsl record --duration` (and then the number of seconds you want to record for)
For example: muselsl record --duration 300 will record for 300 seconds (5 minutes). Make sure the time you set it to record for is longer than the time that your psychopy project will run for. It's better to record too much rather than not enough.

If it has started recording, it will show:
Start recording at time t=(some number)
Time correction: (and then some number)

Alternatively for more control over the record time: Make the duration a very large number, like 7200 seconds (2 hours) then just press control+c in terminal once you've recorded your full task duration. It will automatically cut the recording and save what's been recorded so far as a csv. This is the method I recommend for convenience.

Once the recording is done, the terminal will show the filepath to where it is. Run:
open (and then the file path) in the terminal. It will open the data file. Save it under your project folder. The data should look something like this:


EEG_recording_2025-07-21-01.35.44

timestamps	TP9	AF7	AF8	TP10	Right AUX
1753061750.529	999.512	911.133	114.746	771.484	0.000
1753061750.533	504.883	652.832	-305.176	999.512	0.000
1753061750.537	-1000.000	-878.418	-909.180	-434.082	0.000
1753061750.541	-1000.000	-1000.000	-785.156	-1000.000	0.000
1753061750.545	440.918	163.086	-18.066	100.586	0.000
1753061750.549	999.512	999.512	-46.875	999.512	0.000
1753061750.553	-436.035	-276.367	-831.055	96.680	0.000
1753061750.557	-1000.000	-1000.000	-872.070	-1000.000	0.000
1753061750.561	-105.957	-217.773	-36.133	-250.000	0.000

And will be tens of thousands of lines long or more. That is because it is taking data by milliseconds, so 1000 data point readings per second. You may need to resave it as a csv, since it sometimes automatically saves as a numbers file

Step 6: Run your psychopy project

Start recording with the muse, then run your psychopy project fully, and afterwards stop the muse recording. By the end of this, you will have 2 csv files: your psychopy one with timestamps and the eeg one. Now we can move on to the data analyzing

The csv from psychopy will have a column called “stimulus_timestamp”. That’s the timestamps we will pay attention to.

Step 7: Data analysis

Note: analysis methods 1,2 and 3 plot the data as is, analysis methods 4 and 5 do additional cleaning and analysis to get easier to use data for research and projects.

This step will be broken down into several parts depending on the kind of analysis you want to do. Your csv files will need to follow a specific format. The eeg one needs to have the columns the same as in the image from step 5- which is anyways the default formatting for muselsl recordings. The psychopy csv needs 2 columns that analysis programs will look at. The column that has the timestamps related to certain events needs to be named “Marker Timestamp” and the column with the event types (the event factors you're comparing) needs to be named "Stimulus”. For example, if you’re doing an oddball task comparing reactions to circles (common) vs squares (oddball), your column titled Stimulus would have rows that say “circle” or “square” underneath them, and the programs would compare these 2 stimuli in the plots made.

If you're doing an experiment comparing patients in different situations who are taking the same test, and you wish to compare the eeg of the different patients, I recommend pasting the different patient's csv results into one csv file, then adding a column titled "Stimulus" and label each patient depending on which initial group they were part of.

Now, moving on the analysis methods. All code required for the analysis methods can be found in this github repository:
https://github.com/Jackalope-does-coding/Eeg-analysis-methods
Open the repository on github, click the green "<> Code" button, and click "Download ZIP". This will download all files from the repository into a folder. For simplicity, I recommend also saving your psychopy project and all the csv data files into this same folder, although it's not necessary.

You will also need to have software installed on your computer to run python files. I used jupyter notebooks to run the .ipynb files, and VS code to run the .py files (see the file names to know which kind of file it is). However, you could run both types of files from jupyter notebooks or from VS code, you aren't obligated to have both.

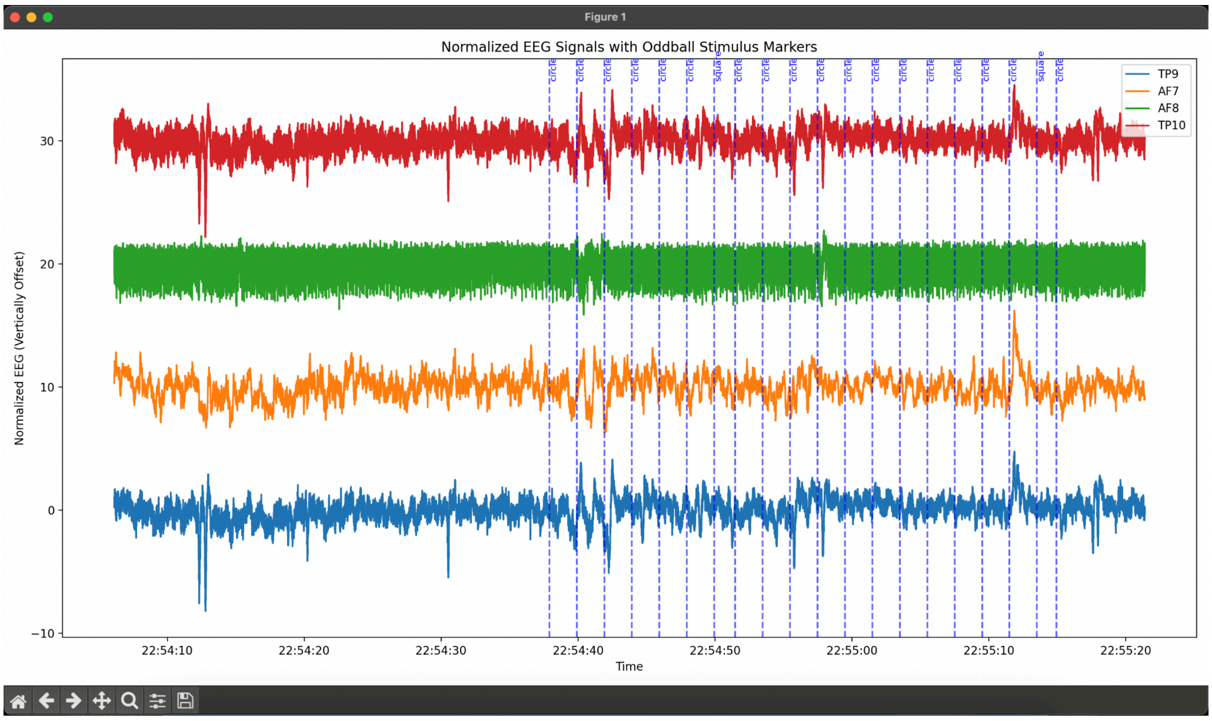
Here is the link to install VS code (visual studio code):
https://code.visualstudio.com/download
VS code has the benefit of being easy to use for beginners and having better automatic debugging tools.

And here is the link to download anaconda, which includes jupyter notebooks:
https://www.anaconda.com/download/success
Jupyter notebooks/anaconda has the benefit of being better for plotting and package installations, as well as the cells providing a simpler and more organized code breakdown, and minimizes unnecessary processing by enabling running the code in parts. I also find it to be more visually appealing, as well as being web based.

To run a code, open the file from the folder (either in jupyter notebooks or in VS code). Then, go into the code and change the file paths to match the ones from your data (comments at the start of the code and in the code indicate which lines need to be modified). Then, if it's a .ipynb file, run each cell in order. If it's a .py file, you just need to run the file. Also, keep in mind that the images showing what the result of code should look like will have different words than the one here, because you will likely be using different stimuli. All the shown images below were tested with data from a circle vs. square oddball task, which is why each image has stimulus markers called "square" and "circle".

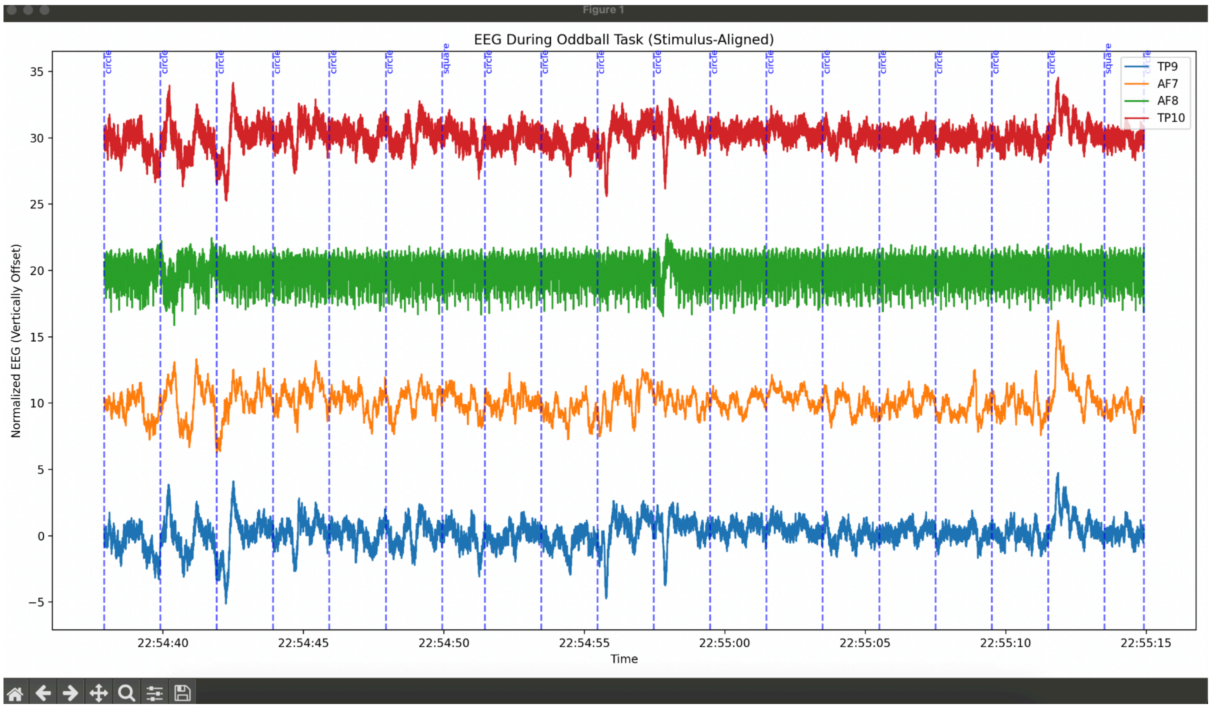
Analysis method 1 - Simple plotting

This just displays the data as is. It shows the entirety of the recorded time and its overlap with the psychopy csv data, without cleaning it. From the github folder, it's the file called "simple_plotting.py". This is a good place to start just to make sure your data is visible and overlapping correctly. A verification of sorts. Just open this python file, change the file paths to your csv folders, and you're good to run it. This is what the result should look like:



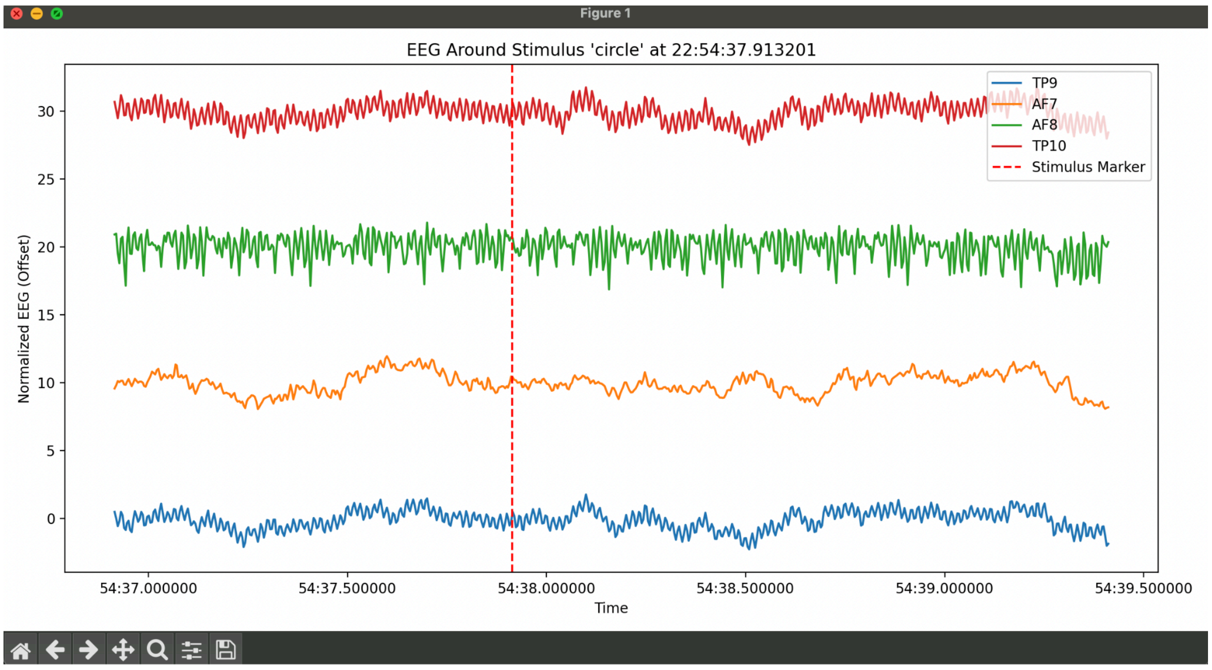
Analysis method 2 - Experiment plotting

This displays the zone where the eeg readings and the psychopy experiment overlap, which is the part that interests us. From the github folder, it's the file called "experiment_plotting.py". This is a good code to run to visualize sort of generally how your data came out looking. Just open this python file, change the file paths to your csv folders, and you're good to run it. This is what the result should look like:



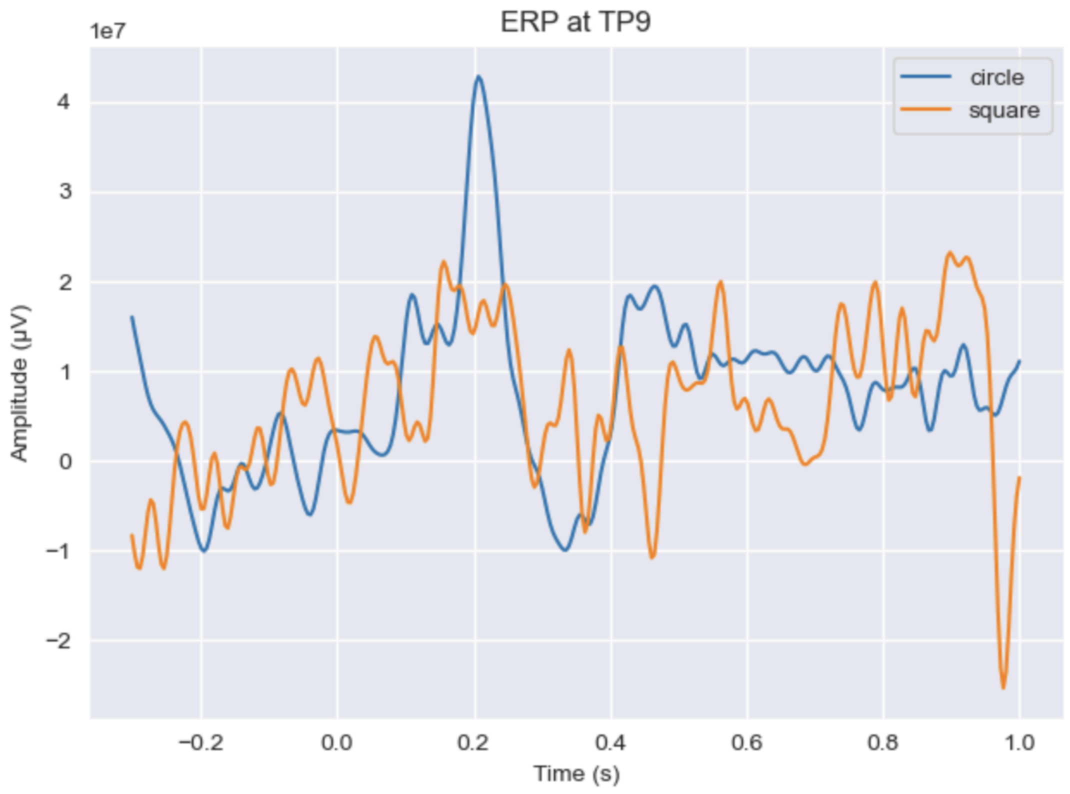
Analysis method 3 - Stimulus slices plotting

This displays one small slice of eeg reading around the point where a stimulus appears for every single stimulus shown. From the github folder, it's the file called "stimulus_slices_plotting.py". This is good for observing the expected spikes and general changes in brainwaves when participants are shown a stimulus. Just open this python file, change the file paths to your csv folders, and you're good to run it. The result should generate something that looks like the upcoming image, except it will generate one of those for every Stimulus shown. After closing each one, the next one will come up until you've gone through all of them. These EEG slices should look something like this:



Analysis method 4 - ERP analysis

Now we are moving on to the analysis methods that clean and display data with formatting that's easier to understand and use for studies and projects.
From the github folder, it’s the file called “ERP_cleaning_and_analysis.ipynb”. Notice that this filename ends with .ipynb instead of .py. This is because it was made with jupyter notebooks, so it’s running with python kernels. This means that the larger code is divided into smaller cells that can be run separately. To use, replace the filepaths to your csv files, and then run each kernel in order from top to bottom. I recommend running .ipynb files from jupyter notebooks, but it can also be run from VS code, google colab, or most other python runners. The results displayed will be four plots of cleaned eeg data, one for each electrode. Each plot will have one line displaying the average wave from this electrode for each different stimulus type. The result should be 4 plots that look similar to this:

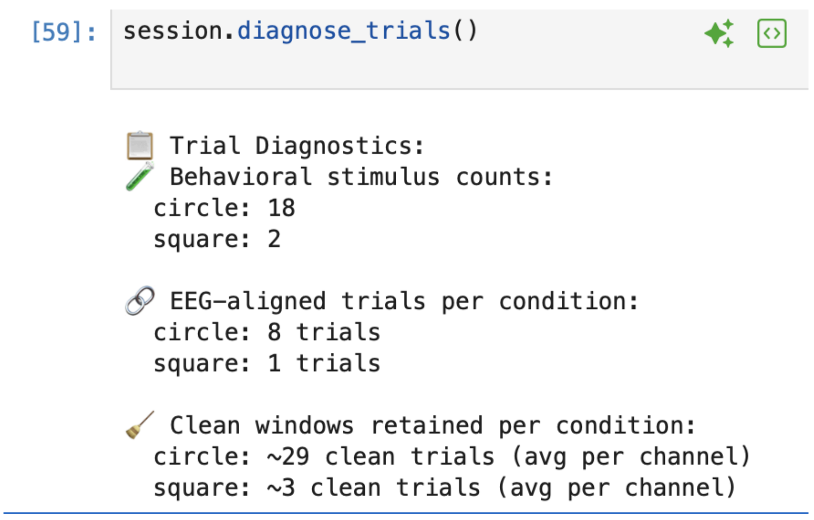


Analysis method 5 - bands analysis

From the github folder, it's the file called "general_bands_cleaning_and_analysis.ipynb". This is another .ipynb file so the kernels need to be run in order to get the end result. This file actually has 3 interesting results that it displays, which come from running the final 3 kernels. The first is a display of the cleaning diagnostics, which indicate:

1) how many stimuli were initially inserted (make sure this number is correct so that you know for sure the code is processing your file correctly!)
2) how many of them the code kept for analysis (trials that don't line up or have too much noise to clean are removed. This can happen if a testee is moving their head a lot or blinking right when the stimuli appears, as this can create large unrelated spikes of noise that cover the actual data)
3) an average for the channel of how many clean trials are likely to appear and be useable (you can ignore this part as it is rarely relevant for research, and is more of a diagnostic matter for the code)

This will look like the following:



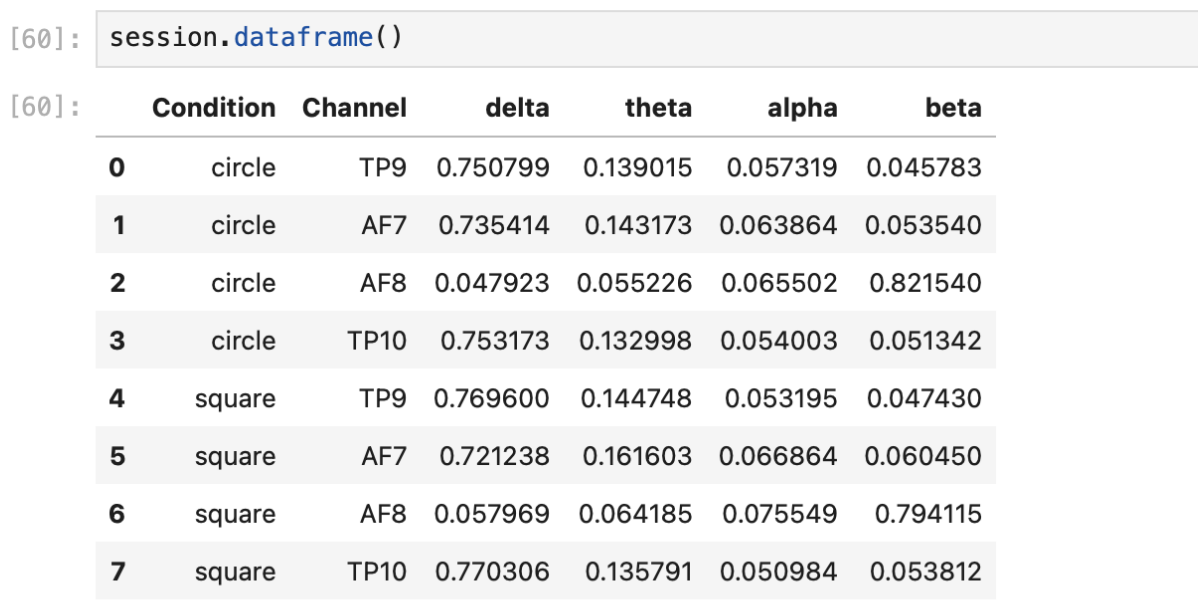
```
[59]: session.diagnose_trials()
```

```
📋 Trial Diagnostics:
🧪 Behavioral stimulus counts:
  circle: 18
  square: 2

🔗 EEG-aligned trials per condition:
  circle: 8 trials
  square: 1 trials

🧹 Clean windows retained per condition:
  circle: ~29 clean trials (avg per channel)
  square: ~3 clean trials (avg per channel)
```

The second thing it displays is the dataframe, which is a dataset composed of the average band power for each of the 4 electrodes for each one of the stimuli from the cleaned data! So basically, it takes the cleaned usable data, and does a power average so you can compare the reactions of the different parts of the brain to the different stimuli! (Though it should be kept in mind that the Muse EEG only has 4 electrodes so it doesn't provide a full complex view of the brain activity). The dataframe produced should look something like this:



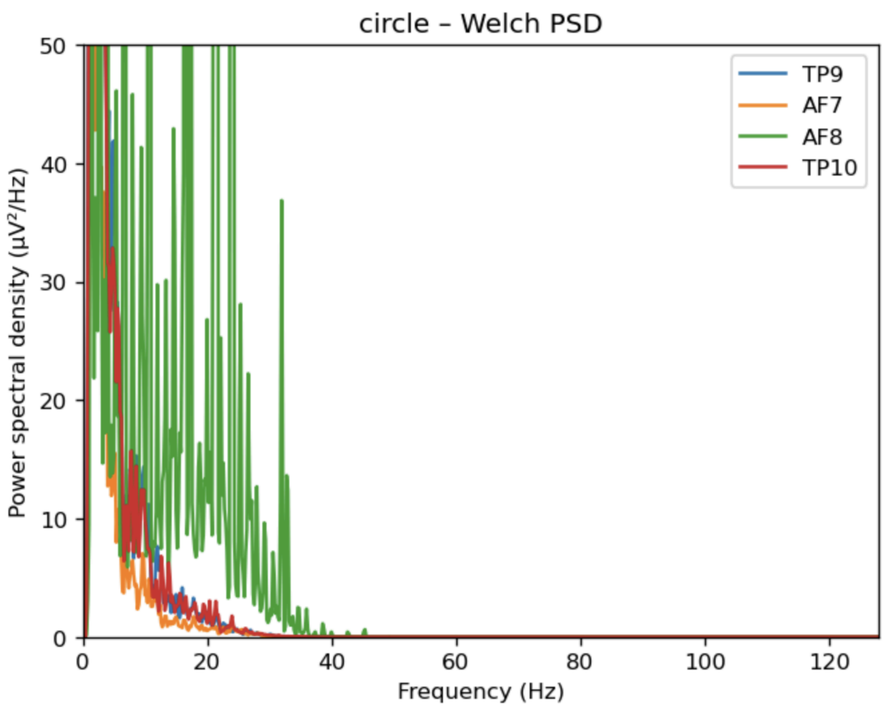
```
[60]: session.dataframe()
```

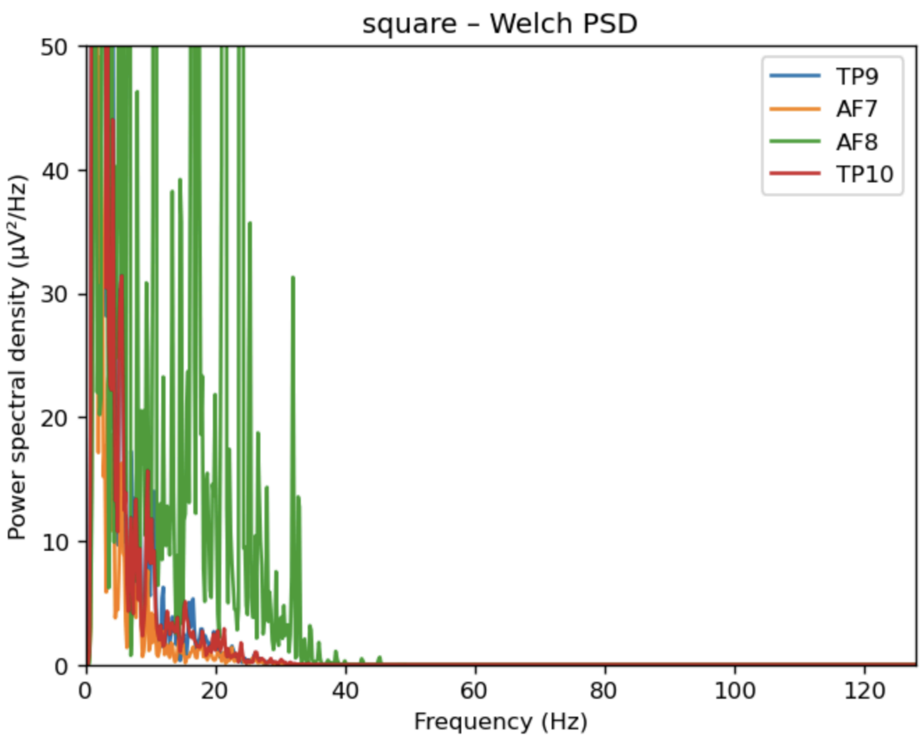
[60]:

	Condition	Channel	delta	theta	alpha	beta
0	circle	TP9	0.750799	0.139015	0.057319	0.045783
1	circle	AF7	0.735414	0.143173	0.063864	0.053540
2	circle	AF8	0.047923	0.055226	0.065502	0.821540
3	circle	TP10	0.753173	0.132998	0.054003	0.051342
4	square	TP9	0.769600	0.144748	0.053195	0.047430
5	square	AF7	0.721238	0.161603	0.066864	0.060450
6	square	AF8	0.057969	0.064185	0.075549	0.794115
7	square	TP10	0.770306	0.135791	0.050984	0.053812

Finally, there's the plotting of the power bands as a Welch periodogram, for a visual representation of the data. This will create one graph for each of the Stimulus types representing the average power bands. Here is what the data should look like:

```
[61]: for condition in session.dataframe()["Condition"].unique():
          session.plot(condition, ylim=(0, 50))
```





Best of luck on your EEG projects!