
eeg-notebooks

Release *fe535do*

John Griffiths, Dano Morrison, et al.

Jul 25, 2018

Background

1	Getting Started	1
2	muse-lsl + eeg-notebooks windows installation + setup instructions	2
3	Available Notebooks	8
4	N170	10
5	P300	18
6	SSVEP	26
7	Using an extra electrode with Muse	36
8	Technical Information about the MUSE	38

EEG notebooks is a collection of classic EEG experiments, implemented in Python and Jupyter notebooks. The experimental protocols and analyses are quite generic, but are primarily tailored for low-budget / consumer EEG hardware such as the MUSE. The goal is to make cognitive neuroscience and neurotechnology more *accessible*, *affordable*, and *scalable*.

The code is hosted on github: <https://github.com/NeuroTechX/eeg-notebooks/>

1 Getting Started

1.1 Installation

You will need a Muse 2016 and Python installed on your computer. Psychopy, the stimulus presentation library that underlies most of the experiments, officially only supports Python 2. However, some users, especially those on Linux, have been able to work entirely in Python 3 without any issues.

```
git clone https://github.com/neurotechx/eeg-notebooks
```

Install all requirements.

```
pip install requirements.txt
```

See [here](#)¹ for more detailed setup instructions for windows operating systems.

1.2 Running Experiments

Open the experiment you are interested in running in notebooks folder. Notebooks can be opened either with the Jupyter Notebook browser environment (run `jupyter notebook`) or in the [nteract](#)² desktop application.

All experiments should be able to performed entirely within the notebook environment. On Windows 10, you will want to skip the bluetooth connection step and start an EEG data stream through the [BlueMuse](#)³ GUI.

*Note: if errors are encountered during viewing of the eeg data, try starting the viewer directly from the command line (`muselsl view`). Version 2 of the viewer may work better on Windows computers (`muselsl view -v 2`)

The basic steps of each experiment are as follows:

1. Open an LSL stream of EEG data.
2. Ensure that EEG signal quality is excellent and that there is very little noise. The standard deviation of the signal (displayed next to the raw traces) should ideally be below 10 for all channels of interest.
3. Define subject and session ID, as well as trial duration. *Note: sessions are analyzed independently. Each session can contain multiple trials or 'run-throughs' of the experiments.*
4. Simultaneously run stimulus presentation and recording processes to create a data file with both EEG and event marker data.
5. Repeat step 4 to collect as many trials as needed (4-6 trials of two minutes each are recommended in order to see the clearest results)
6. Load experimental data into an MNE Raw object.
7. Apply a band-pass filter to remove noise
8. Epoch the data, removing epochs where amplitude of the signal exceeded a given threshold (removes eye blinks)
9. Generate averaged waveforms from all channels for each type of stimulus presented

Notebooks in the `old_notebooks` folder only contain the data analysis steps (6-9). They can be used by using the `run_experiments.py` script (e.g `python run_eeg_experiment.py Auditory_P300 15 1`)

2 muse-lsl + eeg-notebooks windows installation + setup instructions

2.1 1. Install miniconda

Miniconda is a 'mini' version of the anaconda python distribution.

Download the Windows miniconda installer from <https://conda.io/miniconda.html> (python 2.7 64-bit version)


¹ http://eeg-notebooks.readthedocs.io/en/latest/setup_instructions_windows.html

² <https://nteract.io/desktop>

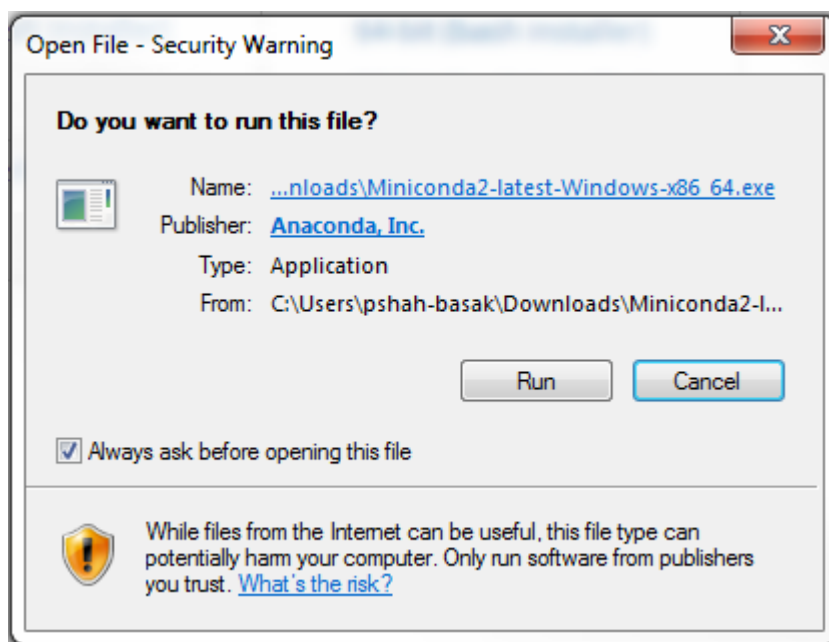
³ <https://github.com/kowalej/BlueMuse>

Tip: you can check your windows operating system type in the Control Panel → System and Security → System

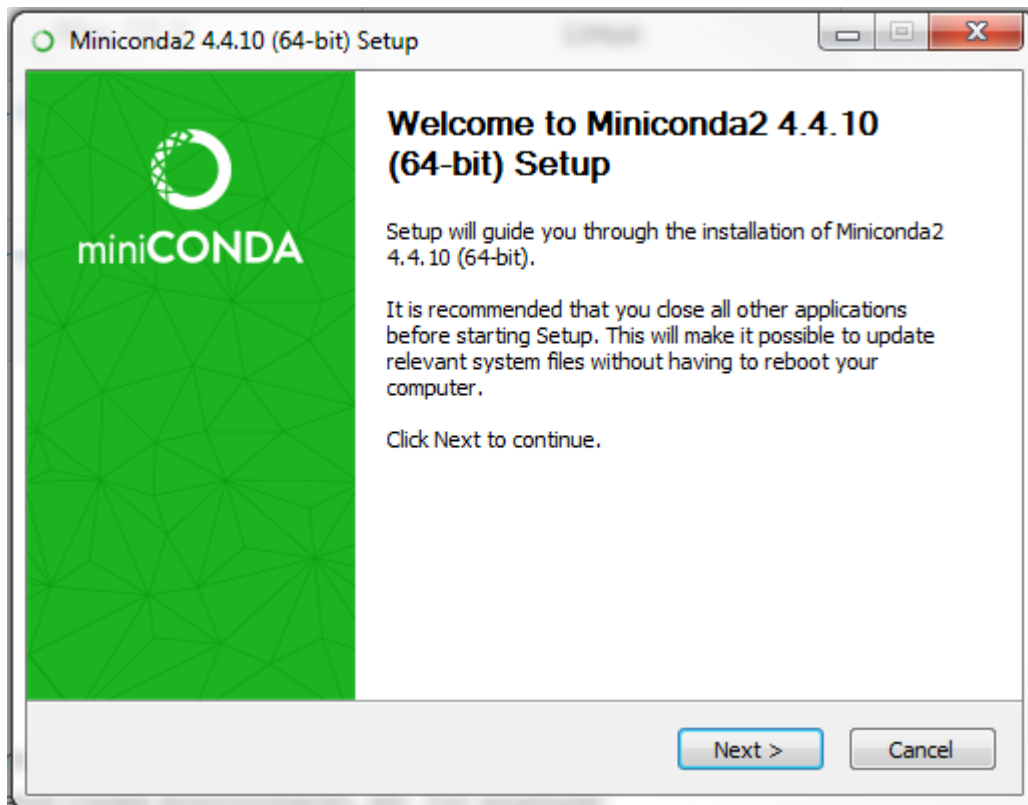
Miniconda

	 Windows
Python 3.6	64-bit (exe installer) 32-bit (exe installer)
Python 2.7	64-bit (exe installer) 32-bit (exe installer)

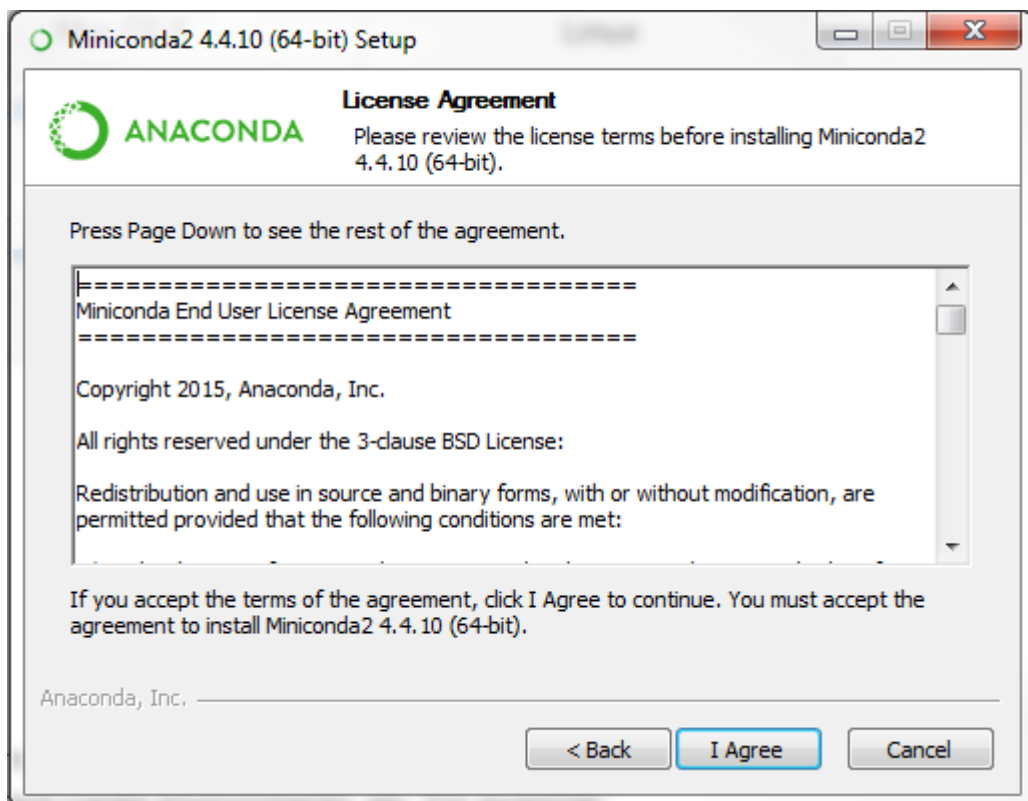
Run the installer and follow the steps below



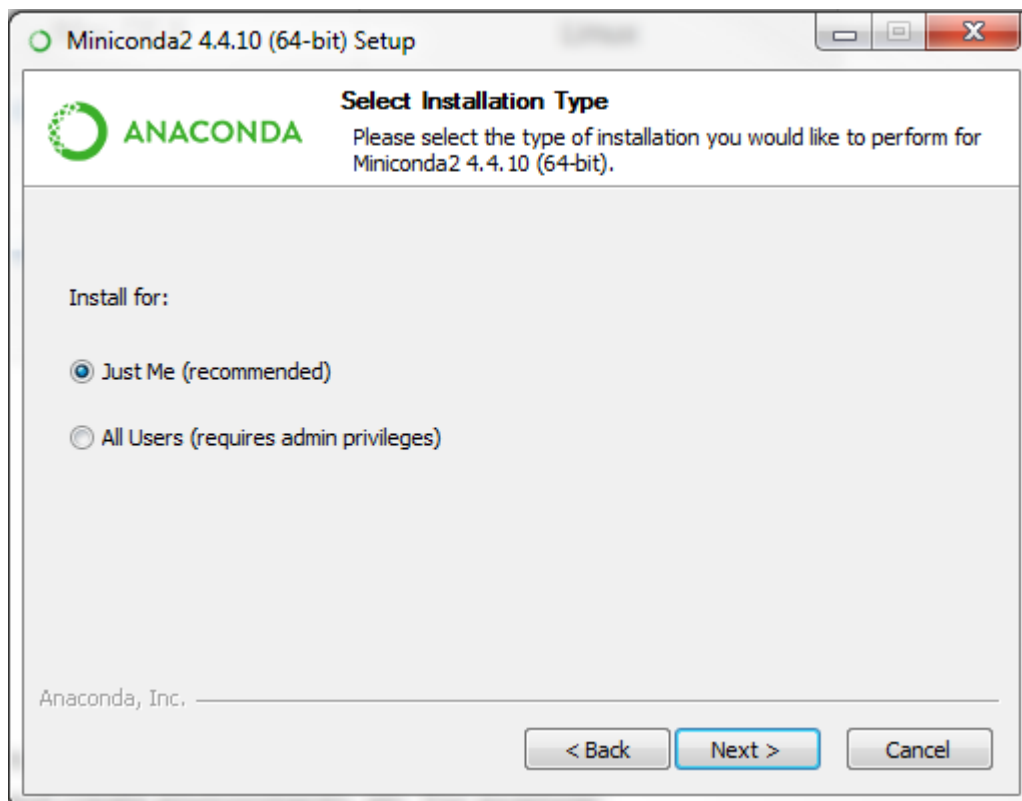
Click Next



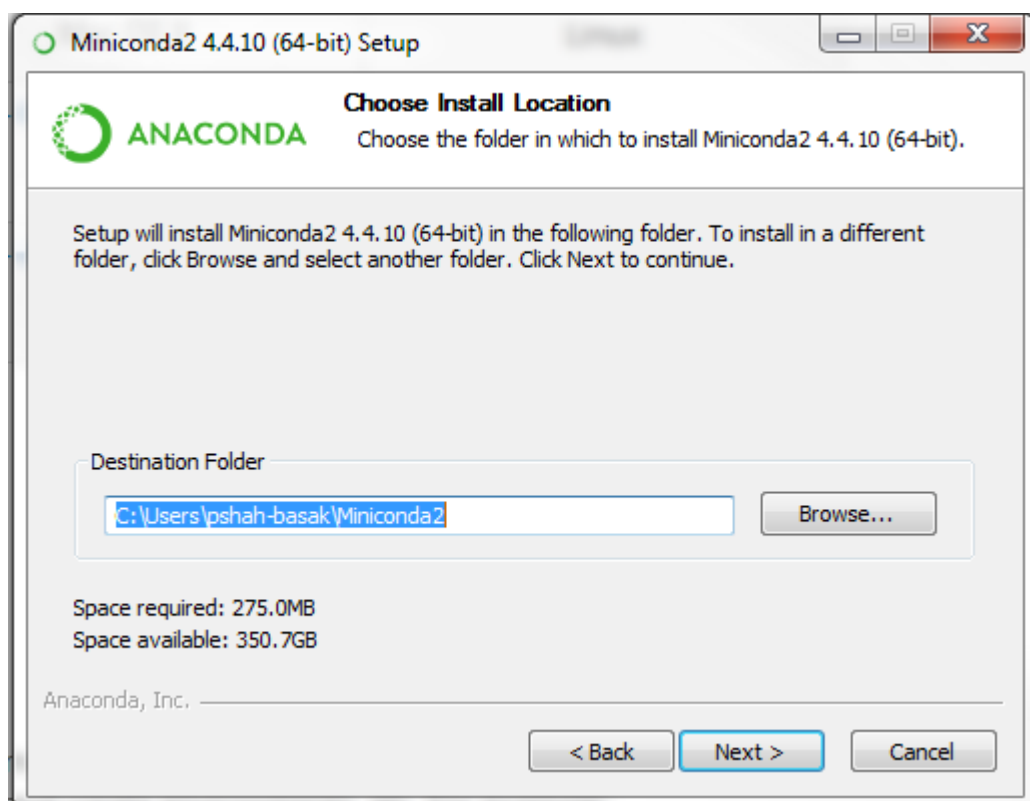
Click 'I Agree'



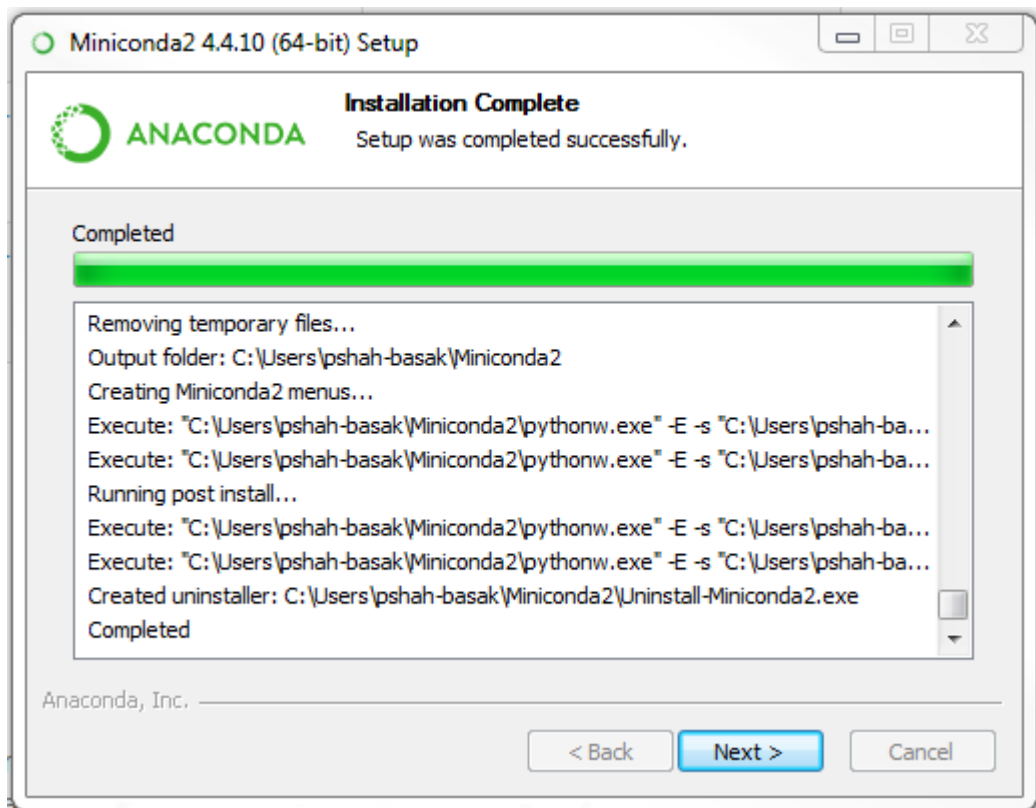
Select 'Just Me' and click Next



Browse to the location where you want to install, or click Next to keep the default location (Tip: Make sure you have enough space available on your hard-drive for this installation)



Once installation is complete, click Next



Click Finish



After the installation is complete, click on the Windows button and search for 'Anaconda Prompt' Tip: pin it to the Windows taskbar for easy access in the future

Click on Anaconda Prompt, which will bring up an anaconda terminal

Create a conda environment for your neurobrite work:

```
conda create -n "neurobrite" python=2
```

2.2 2. Download git for windows

Git is a version control system that will allow you to download and track changes to eeg notebooks. On Windows, it also gives you Git Bash, which is a useful linux-style terminal.

You can download git from: <https://git-scm.com/download/win>

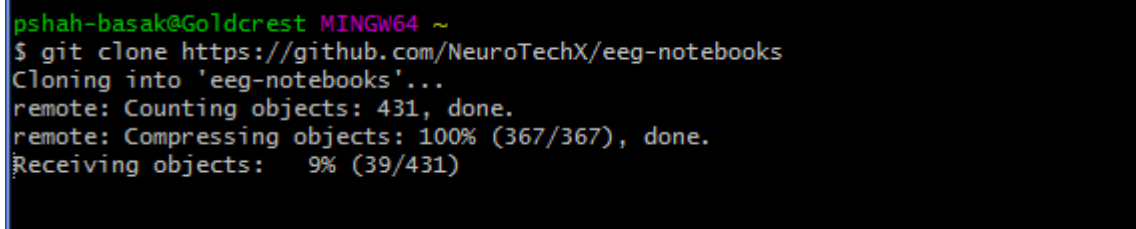
Run the installation with default settings

2.3 3. Get eeg-notebooks

You have two options, pick one from the following:

1. With git bash:

```
git clone --recursive https://github.com/NeuroTechX/eeg-notebooks
```



```
pshah-basak@Goldcrest MINGW64 ~  
$ git clone https://github.com/NeuroTechX/eeg-notebooks  
Cloning into 'eeg-notebooks'...  
remote: Counting objects: 431, done.  
remote: Compressing objects: 100% (367/367), done.  
Receiving objects: 9% (39/431)
```

de

1. Without git bash:

Navigate to the github repository (or <https://github.com/NeuroTechX/eeg-notebooks>) directly in a web browser and download eeg-notebooks.

Now, you are ready to use the jupyter notebook.

2.4 4. Install python dependencies

Go back to your open Anaconda Prompt (or open a new one) and navigate to the location where you installed eeg-notebooks. You can use the cd command to change directories (i.e. `cd eeg-notebooks`)

Activate the neurobrite conda environment. Note: you will need to activate this environment every time you start a new terminal when you want to do work within the neurobrite environment we are about to setup

```
conda activate neurobrite
```

Now, install the dependencies identified in the requirements.txt file. Note: this may take a long time, up to 15 minutes

```
pip install -r requirements.txt
```

2.5 5. (Optional) Install BlueMuse

BlueMuse⁴ is a Windows 10 program that allows communication between a Muse headband and a computer's native bluetooth drivers using the LSL communication protocol. It can be used as an alternative to an external BLE¹ dongle.

⁴ <https://github.com/kowalej/BlueMuse>

To install, go to the [BlueMUSE github website](#)⁵ and follow the installation instructions.

2.6 6. Start a jupyter notebook session

Finally, start a jupyter notebook session from your Anaconda Prompt in the eeg-notebooks directory

```
jupyter notebook
```

A browser should automatically open. If it doesn't, visit `localhost:8888`.

This should bring up the eeg-notebooks folder structure. You will find the list of available experiments in the notebooks folder.

3 Available Notebooks

3.1 Visual P300 with Oddball paradigm

The visual P300 is a spike that occurs 300ms after perceiving a visual stimulus that has implications on decision making. This was validated in Muse by Alexandre Barachant with the Oddball paradigm, in which low-probability target items (oddballs) are interspersed with high probability non-target items. With AB's paradigm, the experiment takes about 10 minutes to run (5 x 2 minute trials). Although the Muse's sensors aren't in the ideal position for detecting the P300, AB was able to attain "good" accuracy in identifying P300 spikes.

3.2 N170

The N170 is an ERP specifically related to the perception of faces. This was validated in Muse by Hubert with a 12 minute experiment (6 x 2 minute trials). Stimuli consists of 12 pictures of houses and 12 pictures of faces. Accuracy of N170 detection is rather good.

3.3 SSVEP

The steady state visual evoked potential is a frequency response produced visual stimulation at specific frequencies. It was validated by Hubert in a 12 minute experiment (6 x 2 minute trials). Stimulation frequencies of 30hz and 20hz were used and an extra electrode at POz was added. Found clear peaks in the PSD at the stimulation frequencies. The peaks were most significant at the extra electrode, which is closest to the primary visual regions, but was detectable at all electrodes and found to have remarkably high accuracy when using a filter bank approach to isolate specific frequencies.

3.4 Old Notebooks

Go/No-Go

An experiment designed to investigate the event-related potentials that can be detected during a Go-No-Go Task, which measures executive, inhibitory control and sustained attention. The subject is rapidly presented with a sequence of circles and squares and is asked to indicate, by pressing the spacebar, whether a shape is a circle.

⁵ <https://github.com/kowalej/BlueMuse>

SSAEP

The steady state auditory evoked potential is a frequency response produced when hearing modulating tones of certain frequencies. It was validated in Muse by Hubert, who used 45hz and 40hz amplitude modulation applied to 900 and 770h carrier frequencies. A PSD of the produced EEG signal showed clear spikes, correspondingly, at 45 and 40hz in the temporal electrodes. The N100 and P200 complex was also noticed at the beginning of stimulus onset.

C1 and P1

C1 and P1 are two ERPs related to the perception of a visual stimulus. The C1 is the first component, appearing in the 65-90ms range after stimulus onset while the P1 appears later, around 100ms.

C1 and P1 were validated in Muse by Hubert with a left/right visual field experiment. Comparing ERPs to left or right-field presentation of visual stimuli revealed a contralateral pattern of C1 and P1 in the both the temporal and anterior electrodes. However, their timing seems a little delayed.

Auditory P300

Same as the visual P300, but dependent on auditory stimulus. Auditory P300s are normally less distinguishable than visual P300s, but they may be more suited to the Muse since its electrodes are closer to auditory centers (superior temporal cortex).

3.5 Unvalidated Experiments and other phenomena

N100 - P200

The combination of a negative evoked potential around 100ms after any unpredictable stimulus and a positive potential 200ms after. These were noticed in Hubert's SSAEP experiment, but not independently classified or tested.

On-task Beta

Noticed in Hubert's visual grating test, but difficult to extract.

Alpha reset

A noticeable increase in alpha activity after stimulus presentation ends. Noticed in Hubert's visual grating test.

The following section was generated from doc/n170.nblink

```
In [3]: from muselsl import stream, list_muses, view, record
        from multiprocessing import Process
        from mne import Epochs, find_events
        from time import time, strftime, gmtime
        import os
        from stimulus_presentation import n170
        from utils import utils
        from collections import OrderedDict
        import warnings
        warnings.filterwarnings('ignore')
```

4 N170

The N170 is a large negative event-related potential (ERP) component that occurs after the detection of faces, but not objects, scrambled faces, or other body parts such as hands. The N170 occurs around 170ms after face perception and is most easily detected at lateral posterior electrodes such as T5 and T6⁶. Frontal or profile views of human (and animal²⁷) faces elicit the strongest N170 and the strength of the N170 does not seem to be influenced by how familiar a face is. Thus, although there is no consensus on the specific source of the N170, researchers believe it is related to activity in the fusiform face area, an area of the brain that shows a similar response pattern and is involved in encoding the holistic representation of a face (i.e eyes, nose mouth all arranged in the appropriate way).

In this notebook, we will attempt to detect the N170 with the Muse headband using faces and houses as our stimuli. The Muse's temporal electrodes (TP9 and TP10) are well positioned to detect the N170 and we expect we'll be able to see an N170 emerge from just a few dozen trials. We will then run several different classification algorithms on our data in order to evaluate the performance of a potential brain-computer interface using the N170.

4.1 Step 1: Connect to an EEG Device

Note: if using Windows 10 and BlueMuse, skip this section and connect using the BlueMuse GUI

Make sure your Muse 2016 is turned on and then run the following code. It should detect and connect to the device and begin 'Streaming...'

If the device is not found or the connection times out, try running this code again

```
In [4]: # Search for available Muse devices
        muses = list_muses()
```

```
Searching for Muses, this may take up to 10 seconds...
Found device Muse-6BF0, MAC Address 00:55:DA:B0:6B:F0
```

```
In [5]: # Start a background process that will stream data from the first available Muse
        stream_process = Process(target=stream, args=(muses[0]['address'],))
        stream_process.start()
```

```
Connecting to Muse : 00:55:DA:B0:6B:F0...
Connected.
Streaming...
```

4.2 Step 2: Apply the EEG Device and Wait for Signal Quality to Stabilize

Once your Muse is connected and streaming data, put it on and run the following code to view the raw EEG data stream.

The numbers on the side of the graph indicate the variance of the signal. Wait until this decreases below 10 for all electrodes before proceeding.

```
In [6]: %matplotlib
        # On Windows, you may need to run the command %matplotlib tk
        view()
```

```
Using matplotlib backend: Qt5Agg
Looking for an EEG stream...
Start acquiring data.
```

```
toggle filter : d
toggle full screen : f
```

⁶ <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.601.6917&rep=rep1&type=pdf>

⁷ <https://www.ncbi.nlm.nih.gov/pubmed/14995895>

```

zoom out : /
zoom in : *
increase time scale : -
decrease time scale : +

```

4.3 Step 3: Run the Experiment

Modify the variables in the following code chunk to define how long you want to run the experiment and the name of the subject and session you are collecting data from.

```

In [20]: # Define these parameters
         duration = 120 # in seconds. 120 is recommended
         subject = 1 # unique id for each participant
         session = 1 # represents a data collection session. Multiple trials can be performed for each ses

```

Seat the subject in front of the computer and run the following cell to run a single trial of the experiment.

In order to maximise the possibility of success, participants should take the experiment in a quiet environment and do their best to minimize movement that might contaminate the signal. With their jaw and face relaxed, subjects should focus on the stimuli, mentally noting whether they see a “face” or a “house”.

Data will be recorded into CSV files in the eeg-notebooks/data directory

```

In [8]: recording_path = os.path.join(os.path.expanduser("~"), "eeg-notebooks", "data", "visual", "N170",
                                       strftime("%Y-%m-%d-%H.%M.%S", gmtime())) + ".csv")

print('Recording data to: ', recording_path)

stimulus = Process(target=n170.present, args=(duration,))
recording = Process(target=record, args=(duration, recording_path))

stimulus.start()
recording.start()

```

```

Recording data to: /home/dano/eeg-notebooks/data/visual/N170/subject_dano/session2/recording_2018-07-24-15
Looking for an EEG stream...
Started acquiring data.
Looking for a Markers stream...
Start recording at time t=1532446368.998
Time correction: -3.252900205552578e-05
52.6888          WARNING          User requested fullscreen with size [1600  900], but screen is actually [1
Time correction: -1.2708998838206753e-05
Done - wrote file: /home/dano/eeg-notebooks/data/visual/N170/subject_dano/session2/recording_2018-07-24-15

```

Repeat Data Collection 3-6 times

Visualizing ERPs requires averaging the EEG response over many different rounds of stimulus presentation. Depending on experimental conditions, this may require as little as one two minute trial or as many as 6. We recommend repeating the above experiment 3-6 times before proceeding.

Make sure to take breaks, though! Inattention, fatigue, and distraction will decrease the quality of event-related potentials such as the N170

4.4 Step 4: Prepare the Data for Analysis

Once a suitable data set has been collected, it is now time to analyze the data and see if we can identify the N170

Load data into MNE objects

MNE⁸ is a very powerful Python library for analyzing EEG data. It provides helpful functions for performing key tasks such as filtering EEG data, rejecting artifacts, and grouping EEG data into chunks (epochs).

The first step to using MNE is to read the data we've collected into an MNE Raw object

```
In [21]: raw = utils.load_data('visual/N170', sfreq=256.,
                                subject_nb=subject, session_nb=session)

Creating RawArray with float64 data, n_channels=5, n_times=30720
  Range : 0 ... 30719 =      0.000 ...   119.996 secs
Ready.
Creating RawArray with float64 data, n_channels=5, n_times=30732
  Range : 0 ... 30731 =      0.000 ...   120.043 secs
Ready.
Creating RawArray with float64 data, n_channels=5, n_times=30732
  Range : 0 ... 30731 =      0.000 ...   120.043 secs
Ready.
Creating RawArray with float64 data, n_channels=5, n_times=30732
  Range : 0 ... 30731 =      0.000 ...   120.043 secs
Ready.
Creating RawArray with float64 data, n_channels=5, n_times=30732
  Range : 0 ... 30731 =      0.000 ...   120.043 secs
Ready.
Creating RawArray with float64 data, n_channels=5, n_times=30732
  Range : 0 ... 30731 =      0.000 ...   120.043 secs
Ready.
```

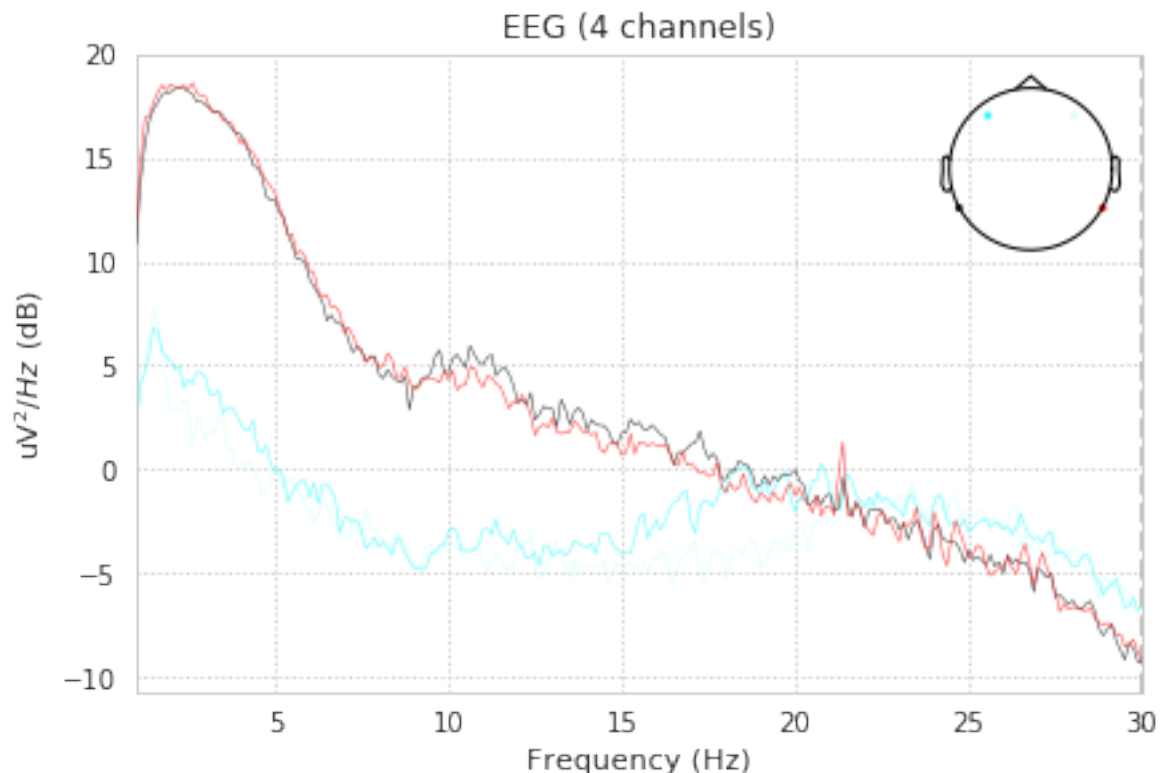
Visualizing the Power Spectrum

Plotting the power spectral density (PSD) of our dataset will give us a glimpse at the different frequencies that are present. We won't be able to see the N170 in the PSD, but it will give us an impression of how noisy our data was. A very noisy or flat PSD may represent poor signal quality at certain electrodes

```
In [22]: %matplotlib inline
         raw.plot_psd();
```

Effective window size : 8.000 (s)

⁸ <https://martinos.org/mne/stable/index.html>



This PSD of frequencies between 1 and 30 hz looks good. The difference between the temporal channels (red and black) and the frontal channels (blue and green) is clearly evident. The huge peak from 1 to 3hz is largely due to the presence of eye blinks, which produce large amplitude, low-frequency events in the EEG.

Epoching

Next, we will chunk (epoch) the data into segments representing the data 100ms before to 800ms after each stimulus. No baseline correction is needed (signal is bandpass filtered) and we will reject every epoch where the amplitude of the signal exceeded 75 uV, which should most eye blinks.

```
In [24]: # Create an array containing the timestamps and type of each stimulus (i.e. face or house)
events = find_events(raw)
event_id = {'House': 1, 'Face': 2}

# Create an MNE Epochs object representing all the epochs around stimulus presentation
epochs = Epochs(raw, events=events, event_id=event_id,
                tmin=-0.1, tmax=0.8, baseline=None,
                reject={'eeg': 75e-6}, preload=True,
                verbose=False, picks=[0,1,2,3])
print('sample drop %: ', (1 - len(epochs.events)/len(events)) * 100)
epochs
```

```
1174 events found
Event IDs: [1 2]
sample drop %: 4.088586030664398
```

```
Out[24]: <Epochs | 1126 events (all good), -0.101562 - 0.800781 sec, baseline off, ~8.0 MB, data loaded
          'Face': 562
          'House': 564>
```

Sample drop % is an important metric representing how noisy our data set was. If this is greater than 20%, consider ensuring that signal variances is very low in the raw EEG viewer and collecting more data

4.5 Step 5: Analyze the Data

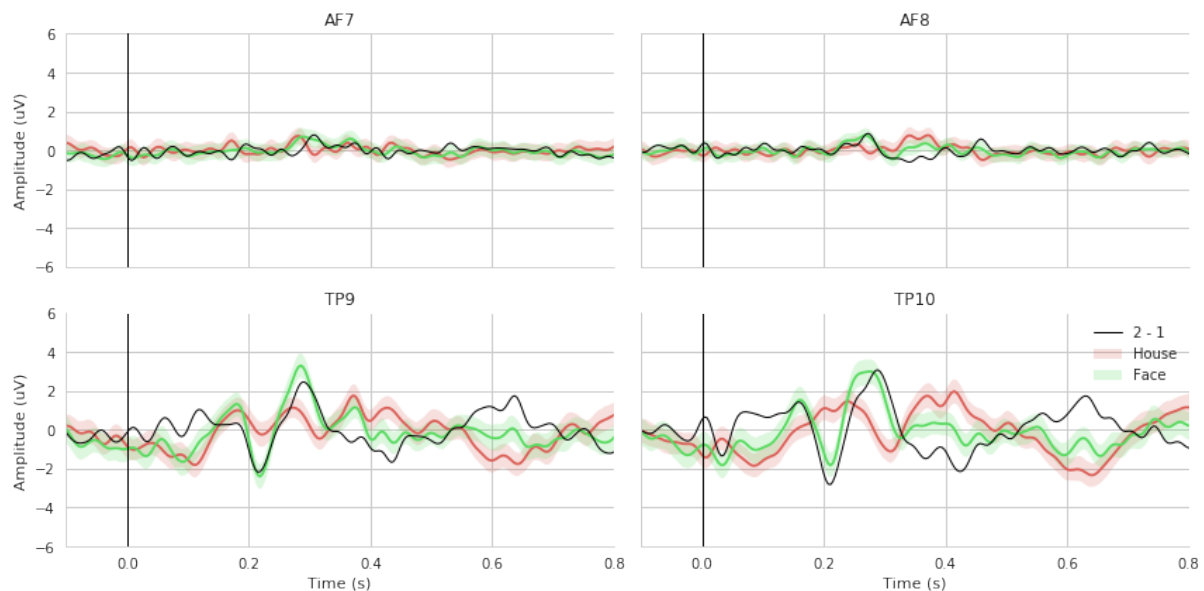
Finally, we can now analyze our results by averaging the epochs that occurred during the different stimuli and looking for differences in the waveform

Epoch average

With our `plot_conditions` utility function, we can plot the average ERP for all electrodes for both conditions:

```
In [25]: %matplotlib inline
         conditions = OrderedDict()
         conditions['House'] = [1]
         conditions['Face'] = [2]

         fig, ax = utils.plot_conditions(epochs, conditions=conditions,
                                         ci=97.5, n_boot=1000, title='',
                                         diff_waveform=(1, 2))
```



Here we have a very nice deflection in the temporal channels around 200ms for face stimuli. This is likely the N170, although appearing slightly later due to delay in receiving the data over bluetooth.

There's not much to see in the frontal channels (AF7 and AF8), but that's to be expected based on the fact that the N170 is mostly a lateral posterior brain phenomenon

Decoding the N170

Next, we will use 4 different machine learning pipelines to classify the N170 based on the data we collected. The

- **Vect + LR** : Vectorization of the trial + Logistic Regression. This can be considered the standard decoding pipeline for MEG / EEG.
- **Vect + RegLDA** : Vectorization of the trial + Regularized LDA. This one is very commonly used in P300 BCIs. It can outperform the previous one but become unusable if the number of dimension is too high.
- **ERPCov + TS**: ErpCovariance + Tangent space mapping. One of the most reliable Riemannian geometry-based pipeline.

- **ERPCov + MDM:** ErpCovariance + MDM. A very simple, yet effective (for low channel count), Riemannian geometry classifier.

Evaluation is done through cross-validation, with area-under-the-curve (AUC) as metric (AUC is probably the best metric for binary and unbalanced classification problem)

Note: because we're doing machine learning here, the following cell may take a while to complete

```
In [26]: import pandas as pd
         from sklearn.pipeline import make_pipeline

         from mne.decoding import Vectorizer

         from sklearn.linear_model import LogisticRegression
         from sklearn.preprocessing import StandardScaler
         from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

         from sklearn.model_selection import cross_val_score, StratifiedShuffleSplit

         from pyriemann.estimation import ERPCovariances, XdawnCovariances
         from pyriemann.tangentspace import TangentSpace
         from pyriemann.classification import MDM

         from collections import OrderedDict

         clfs = OrderedDict()

         clfs['Vect + LR'] = make_pipeline(Vectorizer(), StandardScaler(), LogisticRegression())
         clfs['Vect + RegLDA'] = make_pipeline(Vectorizer(), LDA(shrinkage='auto', solver='eigen'))
         clfs['ERPCov + TS'] = make_pipeline(ERPCovariances(estimator='oas'), TangentSpace(), LogisticRegression())
         clfs['ERPCov + MDM'] = make_pipeline(ERPCovariances(estimator='oas'), MDM())
         clfs['XdawnCov + TS'] = make_pipeline(XdawnCovariances(estimator='oas'), TangentSpace(), LogisticRegression())
         clfs['XdawnCov + MDM'] = make_pipeline(XdawnCovariances(estimator='oas'), MDM())

         # format data
         epochs.pick_types(eeg=True)
         X = epochs.get_data() * 1e6
         times = epochs.times
         y = epochs.events[:, -1]

         # define cross validation
         cv = StratifiedShuffleSplit(n_splits=20, test_size=0.25,
                                     random_state=42)

         # run cross validation for each pipeline
         auc = []
         methods = []
         for m in clfs:
             print(m)
             try:
                 res = cross_val_score(clfs[m], X, y==2, scoring='roc_auc',
                                       cv=cv, n_jobs=-1)
                 auc.extend(res)
                 methods.extend([m]*len(res))
             except:
                 pass

Vect + LR
Vect + RegLDA
ERPCov + TS
ERPCov + MDM
```

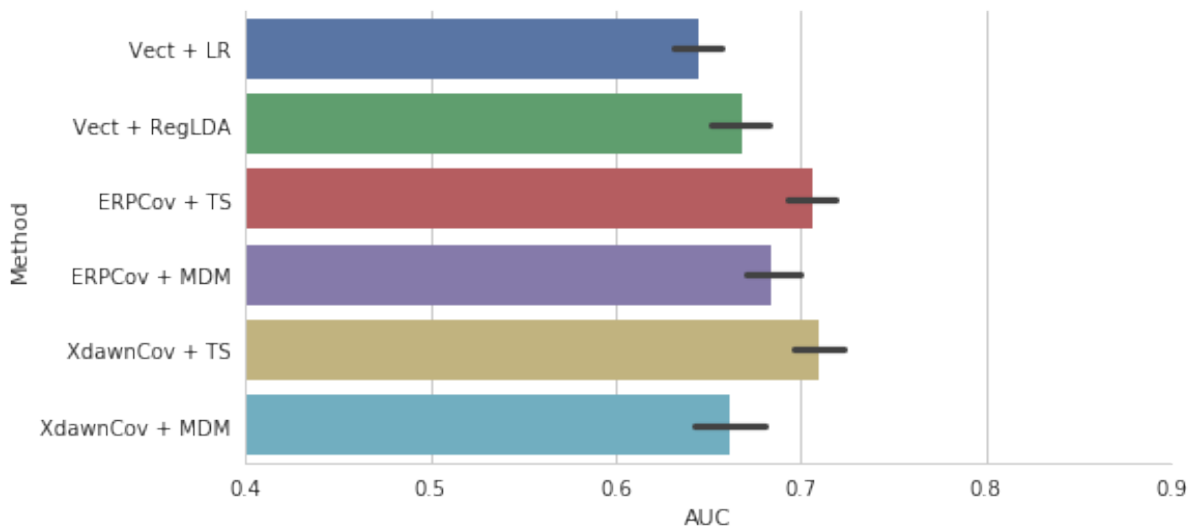

XdawnCov + TS
XdawnCov + MDM

In [27]: *## Plot Decoding Results*

```
import seaborn as sns
from matplotlib import pyplot as plt

results = pd.DataFrame(data=auc, columns=['AUC'])
results['Method'] = methods

fig = plt.figure(figsize=[8,4])
sns.barplot(data=results, x='AUC', y='Method')
plt.xlim(0.4, 0.9)
sns.despine()
```



The best classifiers for this data set appear to be the ERPCov and XdawnCov with tangent space projection pipelines. AUC is around .7, which is good, but on the low end for being able to run a brain-computer interface.

4.6 Step 6: Share your Data!

How did your experiment go? If you're excited by your results we'd love to see your data!

Follow the instructions on our [Contributions](#)⁹ page to make a pull request with your data and we'll review it to be added to the EEG notebooks project.

End of doc/n170.nblink

⁹ <https://github.com/NeuroTechX/eeg-notebooks/blob/master/CONTRIBUTING.md>

The following session was generated from a Jupyter Notebook.

```

In [1]: from mutils import list_muses, visual_p300_notebook
from multiprocessing import Process
from mne import Epochs, find_events
from time import time, strftime, gmtime
import os
from stimulus_presentation import visual_p300_stripes
from utils import utils
from collections import OrderedDict
import warnings
warnings.filterwarnings('ignore')

```

5 P300

The P300 is a positive event-related potential (ERP) that occurs around 300ms after perceiving a novel or unexpected stimulus. It is most commonly elicited through ‘oddball’ experimental paradigms, where a certain subtype of stimulus is presented rarely amidst a background of another more common type of stimulus. Interestingly, the P300 is able to be elicited by multiple sensory modalities (e.g. visual, odditory, somatosensory). Thus, it is believed that the P300 may be a signature of higher level cognitive processing such as conscious attention.

In this notebook, we will attempt to elicit a P300 with a visual oddball stimulation paradigm using the Muse headband

5.1 Step 1: Connect to an EEG Device

Note: if using Windows 10 and BlueMuse, skip this section and connect using the BlueMuse GUI

Make sure your device is turned on and run the following code. It should detect and connect to the device and begin ‘Streaming...’

If the device is not found or the connection times out, try running this code again

```

In [2]: # Search for available Muse devices
muses = list_muses()

```

Searching for Muses, this may take up to 10 seconds...
 Found device Muse-6BF0, MAC Address 00:55:DA:B0:6B:F0

```

In [3]: # Start a background process that will stream data from the first available Muse
stream_process = Process(target=stream, args=(muses[0]['address'],))
stream_process.start()

```

Connecting to Muse : 00:55:DA:B0:6B:F0...
 Connected.
 Streaming...

5.2 Step 2: Apply the EEG Device and Wait for Signal Quality to Stabilize

Once your Muse is connected and streaming data, put it on and run the following code to view the raw EEG data stream.

The numbers on the side of the graph indicate the variance of the signal. Wait until this decreases below 10 for all electrodes before proceeding.

```
In [4]: # On Windows, you may need to replace this with the command %matplotlib tk
        %matplotlib
```

```
view()
```

```
Using matplotlib backend: Qt5Agg
```

```
Looking for an EEG stream...
```

```
Start acquiring data.
```

```
toggle filter : d
toggle full screen : f
zoom out : /
zoom in : *
increase time scale : -
decrease time scale : +
```

5.3 Step 3: Run the Experiment

Modify the variables in the following code chunk to define how long you want to run the experiment and the name of the subject and session you are collecting data from.

```
In [6]: # Define these parameters
        duration = 120 # in seconds. 120 is recommended
        subject = 1 # unique id for each participant
        session = 1 # represents a data collection session. Multiple trials can be performed for each session
```

Seat the subject in front of the computer and run the following cell to run a single trial of the experiment.

In order to maximise the possibility of success, participants should take the experiment in a quiet environment and do their best to minimize movement that might contaminate the signal. With their jaw and face relaxed, subjects should focus on the stimuli, mentally noting when they perceive the oddball stimulus

Data will be recorded into CSV files in the eeg-notebooks/data directory

```
In [5]: recording_path = os.path.join(os.path.expanduser("~"), "eeg-notebooks", "data", "visual", "P300",
                                     strftime("%Y-%m-%d-%H.%M.%S", gmtime())) + ".csv")
```

```
stimulus = Process(target=visual_p300_stripes.present, args=(duration,))
recording = Process(target=record, args=(duration, recording_path))
```

```
stimulus.start()
recording.start()
```

```
Looking for an EEG stream...
```

```
Started acquiring data.
```

```
Looking for a Markers stream...
```

```
Start recording at time t=1532446206.828
```

```
Time correction: -5.673002306139097e-06
```

```
24.9566
```

```
WARNING
```

```
Use of rgb arguments to stimuli are deprecated. Please use color and color
```

Repeat Data Collection 3-6 times

Visualizing ERPs requires averaging the EEG response over many different rounds of stimulus presentation. Depending on experimental conditions, this may require as little as one two minute trial or as many as 6. We recommend repeating the above experiment 3-6 times before proceeding.

Make sure to take breaks, though! Inattention, fatigue, and distraction will decrease the quality of event-related potentials.

5.4 Step 4: Prepare the Data for Analysis

Once a suitable data set has been collected, it is now time to analyze the data and see if we can identify the N170

Load data into MNE objects

MNE¹⁰ is a very powerful Python library for analyzing EEG data. It provides helpful functions for performing key tasks such as filtering EEG data, rejecting artifacts, and grouping EEG data into chunks (epochs).

The first step to using MNE is to read the data we've collected into an MNE Raw object

```
In [4]: raw = utils.load_data('visual/P300', sfreq=256.,
                             subject_nb=subject, session_nb=session)
```

```
Creating RawArray with float64 data, n_channels=5, n_times=30732
Range : 0 ... 30731 =      0.000 ...   120.043 secs
Ready.
```

```
Creating RawArray with float64 data, n_channels=5, n_times=30732
Range : 0 ... 30731 =      0.000 ...   120.043 secs
Ready.
```

```
Creating RawArray with float64 data, n_channels=5, n_times=30732
Range : 0 ... 30731 =      0.000 ...   120.043 secs
Ready.
```

```
Creating RawArray with float64 data, n_channels=5, n_times=30732
Range : 0 ... 30731 =      0.000 ...   120.043 secs
Ready.
```

```
Creating RawArray with float64 data, n_channels=5, n_times=30732
Range : 0 ... 30731 =      0.000 ...   120.043 secs
Ready.
```

```
Creating RawArray with float64 data, n_channels=5, n_times=30732
Range : 0 ... 30731 =      0.000 ...   120.043 secs
Ready.
```

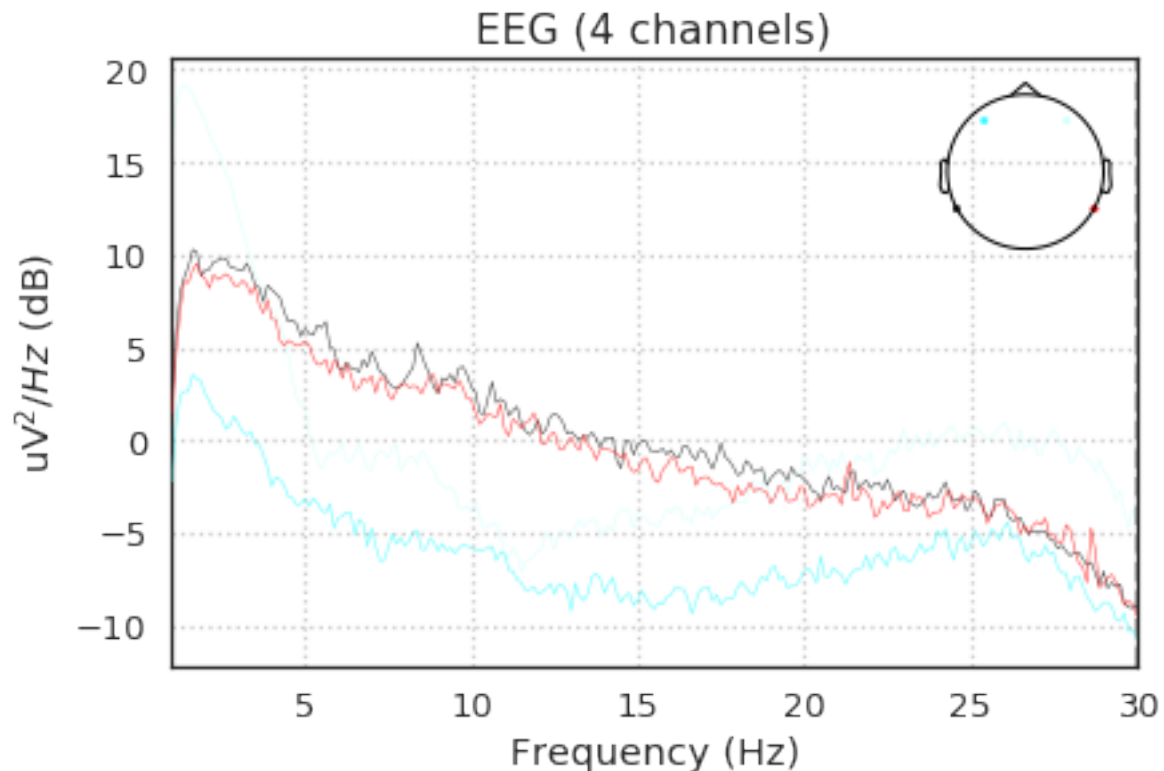
Visualizing the Power Spectrum

Plotting the power spectral density (PSD) of our dataset will give us a glimpse at the different frequencies that are present. We won't be able to see the P300 in the PSD, but it will give us an impression of how noisy our data was. A very noisy or flat PSD may represent poor signal quality at certain electrodes

```
In [5]: %matplotlib inline
        raw.plot_psd();
```

```
Effective window size : 8.000 (s)
```

¹⁰ <https://martinos.org/mne/stable/index.html>



This PSD looks great. The AF8 electrode (Front right; light green) seems to have some noise in the signal, but the TP9 and TP10 electrodes (red and black) look great

Epoching

Next, we will chunk (epoch) the data into segments representing the data 100ms before to 800ms after each stimulus. No baseline correction is needed (signal is bandpass filtered) and we will reject every epoch where the amplitude of the signal exceeded 75 uV, which should most eye blinks.

```
In [7]: events = find_events(raw)
        event_id = {'Non-Target': 1, 'Target': 2}

        epochs = Epochs(raw, events=events, event_id=event_id,
                        tmin=-0.1, tmax=0.8, baseline=None,
                        reject={'eeg': 100e-6}, preload=True,
                        verbose=False, picks=[0,1,2,3])
        print('sample drop %: ', (1 - len(epochs.events)/len(events)) * 100)
        epochs
```

1161 events found

Event IDs: [1 2]

sample drop %: 1.5503875968992276

```
Out[7]: <Epochs | 1143 events (all good), -0.101562 - 0.800781 sec, baseline off, ~8.1 MB, data loaded,
        'Non-Target': 959
        'Target': 184>
```

Sample drop % is an important metric representing how noisy our data set was. If this is greater than 20%, consider ensuring that signal variances is very low in the raw EEG viewer and collecting more data

5.5 Step 5: Analyze the Data

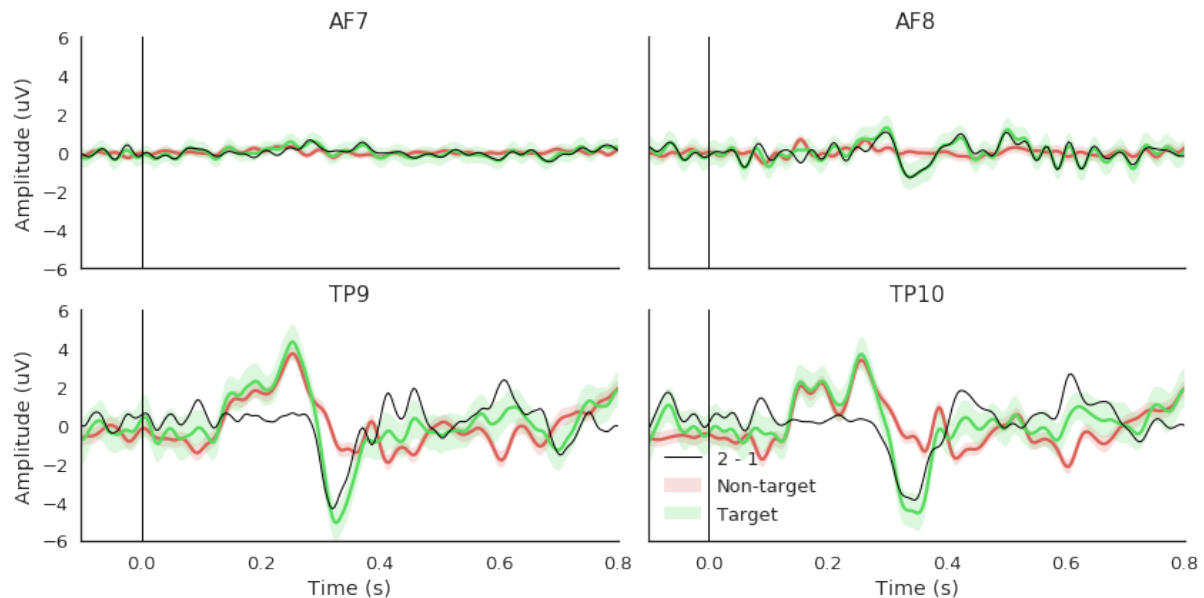
Finally, we can now analyze our results by averaging the epochs that occurred during the different stimuli and looking for differences in the waveform

Epoch average

With our `plot_conditions` utility function, we can plot the average ERP for all electrodes for both conditions:

```
In [8]: %matplotlib inline
        conditions = OrderedDict()
        conditions['Non-target'] = [1]
        conditions['Target'] = [2]

        fig, ax = utils.plot_conditions(epochs, conditions=conditions,
                                       ci=97.5, n_boot=1000, title='',
                                       diff_waveform=(1, 2))
```



Here we can see a beautiful negative deflection in the EEG around 350ms after presentation of Target stimuli. The fact that it's occurring ~50ms later than expected is probably due to delay introduced by transmitting the data over bluetooth.

But wait, isn't this supposed to be a positive ERP? Well, yes, but the Muse's reference electrode is in a different location than traditional EEG systems (at the very front of the forehead instead of the top of the head or near the ear). Because of the location of the source of the P300 signal in the brain, this means that the direction of the P300 potential is inverted

Decoding the N170

Next, we will use 4 different machine learning pipelines to classify the P300 based on the data we collected.

- **Vect + LR** : Vectorization of the trial + Logistic Regression. This can be considered the standard decoding pipeline for MEG / EEG.

- **Vect + RegLDA** : Vectorization of the trial + Regularized LDA. This one is very commonly used in P300 BCIs. It can outperform the previous one but become unusable if the number of dimension is too high.
- **ERPCov + TS**: ErpCovariance + Tangent space mapping. One of the most reliable Riemannian geometry-based pipeline.
- **ERPCov + MDM**: ErpCovariance + MDM. A very simple, yet effective (for low channel count), Riemannian geometry classifier.

Evaluation is done through cross-validation, with area-under-the-curve (AUC) as metric (AUC is probably the best metric for binary and unbalanced classification problem)

Note: because we're doing machine learning here, the following cell may take a while to complete

In [11]: `from sklearn.pipeline import make_pipeline`

```

from mne.decoding import Vectorizer

from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

from sklearn.model_selection import cross_val_score, StratifiedShuffleSplit

from pyriemann.estimation import ERPCovariances
from pyriemann.tangentspace import TangentSpace
from pyriemann.classification import MDM
from pyriemann.spatialfilters import Xdawn

from collections import OrderedDict

clfs = OrderedDict()

clfs['Vect + LR'] = make_pipeline(Vectorizer(), StandardScaler(), LogisticRegression())
clfs['Vect + RegLDA'] = make_pipeline(Vectorizer(), LDA(shrinkage='auto', solver='eigen'))
clfs['Xdawn + RegLDA'] = make_pipeline(Xdawn(2, classes=[1]), Vectorizer(), LDA(shrinkage='auto',
clfs['ERPCov + TS'] = make_pipeline(ERPCovariances(), TangentSpace(), LogisticRegression())
clfs['ERPCov + MDM'] = make_pipeline(ERPCovariances(), MDM())

# format data
epochs.pick_types(eeg=True)
X = epochs.get_data() * 1e6
times = epochs.times
y = epochs.events[:, -1]

# define cross validation
cv = StratifiedShuffleSplit(n_splits=10, test_size=0.25, random_state=42)

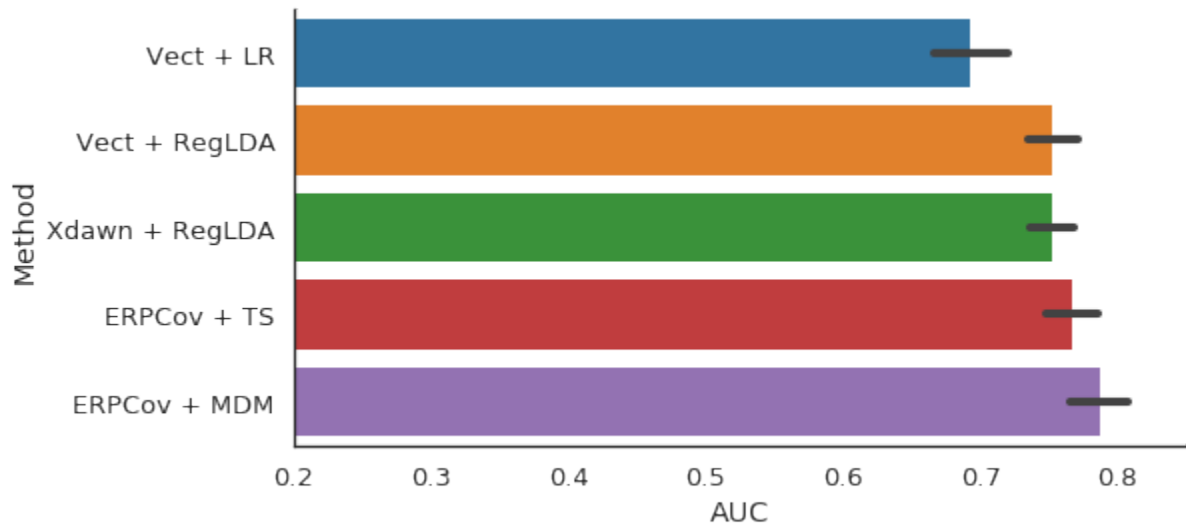
# run cross validation for each pipeline
auc = []
methods = []
for m in clfs:
    res = cross_val_score(clfs[m], X, y==2, scoring='roc_auc', cv=cv, n_jobs=-1)
    auc.extend(res)
    methods.extend([m]*len(res))

results = pd.DataFrame(data=auc, columns=['AUC'])
results['Method'] = methods

```



```
plt.figure(figsize=[8,4])
sns.barplot(data=results, x='AUC', y='Method')
plt.xlim(0.2, 0.85)
sns.despine()
```



The best classifier for this dataset was ERP Covariance + MDM, which achieved an accuracy of around 0.77. This could be considered good enough for a BCI application, though one shouldn't expect outstanding results

5.6 Step 6: Share your Data!

How did your experiment go? If you're excited by your results we'd love to see your data!

Follow the instructions on our [Contributions¹¹](https://github.com/NeuroTechX/eeg-notebooks/blob/master/CONTRIBUTING.md) page to make a pull request with your data and we'll review it to be added to the EEG notebooks project.

End of doc/visual_p300.nblink

¹¹ <https://github.com/NeuroTechX/eeg-notebooks/blob/master/CONTRIBUTING.md>

The following section was generated from doc/ssvep.nblink

```
In [28]: from muselsl import stream, list_muses, view, record
        from multiprocessing import Process
        from mne import Epochs, find_events
        from time import time, strftime, gmtime
        import os
        from stimulus_presentation import ssvep
        from utils import utils
        from collections import OrderedDict
        import warnings
        warnings.filterwarnings('ignore')
```

6 SSVEP

The steady-state visual evoked potential (SSVEP) is a repetitive evoked potential that is naturally produced when viewing stimuli flashing between a range of 6-75hz. Electrical activity at the same frequency as the visual stimulation can be detected in the occipital areas of the brain, likely due to the perceptual recreation of the stimulus in the primary visual cortex.

The SSVEP is often used in BCI applications due to its ease of detection and the amount of information that a user can communicate due to the high potential frequency resolution of the SSVEP.

In this notebook, we will use the Muse EEG headband with an extra occipital electrode to detect the SSVEP and evaluate it's use in SSVEP-based BCIs.

Extra Electrode

Although the SSVEP is detectable at the default temporal electrodes, it can be seen much more clearly directly over the occipital cortex.

The Muse 2016 supports the addition of an extra electrode which can be connected through the devices microUSB charging port.

- [Instructions on how to build an extra electrode for Muse¹²](#)
- [Working with the extra electrode¹³](#)

For this experiment, the extra electrode should be placed at POz, right at the back of the skull. It can be secured in place with a bandana or a hat

6.1 Step 1: Connect to an EEG Device

Note: if using Windows 10 and BlueMuse, skip this section and connect using the BlueMuse GUI

Make sure your device is turned on and run the following code. It should detect and connect to the device and begin 'Streaming...'

If the device is not found or the connection times out, try running this code again

```
In [29]: # Search for available Muse devices
        muses = list_muses()
```

Searching for Muses, this may take up to 10 seconds...
Found device Muse-6BF0, MAC Address 00:55:DA:B0:6B:F0

¹² <http://forum.choosemuse.com/t/step-by-step-tutorial-for-making-muse-auxilliary-channel-electrode/3172?u=tttz>

¹³ https://eeg-notebooks.readthedocs.io/en/latest/using_an_extra_electrode_muse.html

```
In [30]: # Start a background process that will stream data from the first available Muse
stream_process = Process(target=stream, args=(muses[0]['address'],))
stream_process.start()
```

```
Connecting to Muse : 00:55:DA:B0:6B:F0...
Connected.
Streaming...
missing sample 0 : 65535
```

6.2 Step 2: Apply the EEG Device and Wait for Signal Quality to Stabilize

Once your Muse is connected and streaming data, put it on and run the following code to view the raw EEG data stream.

The numbers on the side of the graph indicate the variance of the signal. Wait until this decreases below 10 for all electrodes before proceeding.

Note: the extra electrode's data is included in the stream as the 'Right Aux' Channel

```
In [31]: %matplotlib
view()
```

```
Using matplotlib backend: Qt5Agg
Looking for an EEG stream...
Start acquiring data.
```

```
toggle filter : d
toggle full screen : f
zoom out : /
zoom in : *
increase time scale : -
decrease time scale : +
```

6.3 Step 3: Run the Experiment

Modify the variables in the following code chunk to define how long you want to run the experiment and the name of the subject and session you are collecting data from.

```
In [45]: # Define these parameters
duration = 120 # in seconds. 120 is recommended
subject = 1 # unique id for each participant
session = 1 # represents a data collection session. Multiple trials can be performed for each ses
```

Seat the subject in front of the computer and run the following cell to run a single trial of the experiment.

In order to maximise the possibility of success, participants should take the experiment in a quiet environment and do their best to minimize movement that might contaminate the signal. With their jaw and face relaxed, subjects should look directly at the flashing stimuli.

Data will be recorded into CSV files in the eeg-notebooks/data directory

```
In [46]: recording_path = os.path.join(os.path.expanduser("~"), "eeg-notebooks", "data", "visual", "SSVEP",
strftime("%Y-%m-%d-%H.%M.%S", gmtime())) + ".csv")

stimulus = Process(target=ssvep.present, args=(duration,))
recording = Process(target=record, args=(duration, recording_path))
```

```

stimulus.start()
recording.start()

Looking for an EEG stream...
Started acquiring data.
Looking for a Markers stream...
Flickering frequencies (Hz): [30.0, 20.0]

Start recording at time t=1532444633.838
Time correction: -2.2330499632516876e-05
5720.7450      WARNING      User requested fullscreen with size [1600  900], but screen is actually [1

```

Repeat Data Collection 3-6 times

The SSVEP is a very prominent signal, but visualizing it can sometimes take many rounds of stimulus presentation. Depending on experimental conditions, this may require as little as one two minute trial or as many as 6. We recommend repeating the above experiment 3-6 times before proceeding.

Make sure to take breaks, though! Inattention, fatigue, and distraction will decrease the quality of potentials such as the SSVEP

6.4 Step 4: Prepare the Data for Analysis

Once a suitable data set has been collected, it is now time to analyze the data and see if we can identify the N170

Load data into MNE objects

MNE¹⁴ is a very powerful Python library for analyzing EEG data. It provides helpful functions for performing key tasks such as filtering EEG data, rejecting artifacts, and grouping EEG data into chunks (epochs).

The first step to using MNE is to read the data we've collected into an MNE Raw object

```

In [66]: raw = utils.load_data('visual/SSVEP', sfreq=256.,
                                subject_nb=subject, session_nb=session,
                                ch_ind=[0, 1, 2, 3, 4],
                                replace_ch_names={'Right AUX': 'POz'})

```

```

Creating RawArray with float64 data, n_channels=6, n_times=30720
Range : 0 ... 30719 =      0.000 ...   119.996 secs

```

Ready.

```

Creating RawArray with float64 data, n_channels=6, n_times=30732
Range : 0 ... 30731 =      0.000 ...   120.043 secs

```

Ready.

```

Creating RawArray with float64 data, n_channels=6, n_times=30732
Range : 0 ... 30731 =      0.000 ...   120.043 secs

```

Ready.

```

Creating RawArray with float64 data, n_channels=6, n_times=30732
Range : 0 ... 30731 =      0.000 ...   120.043 secs

```

Ready.

```

Creating RawArray with float64 data, n_channels=6, n_times=30732
Range : 0 ... 30731 =      0.000 ...   120.043 secs

```

Ready.

```

Creating RawArray with float64 data, n_channels=6, n_times=30720

```

¹⁴ <https://martinos.org/mne/stable/index.html>

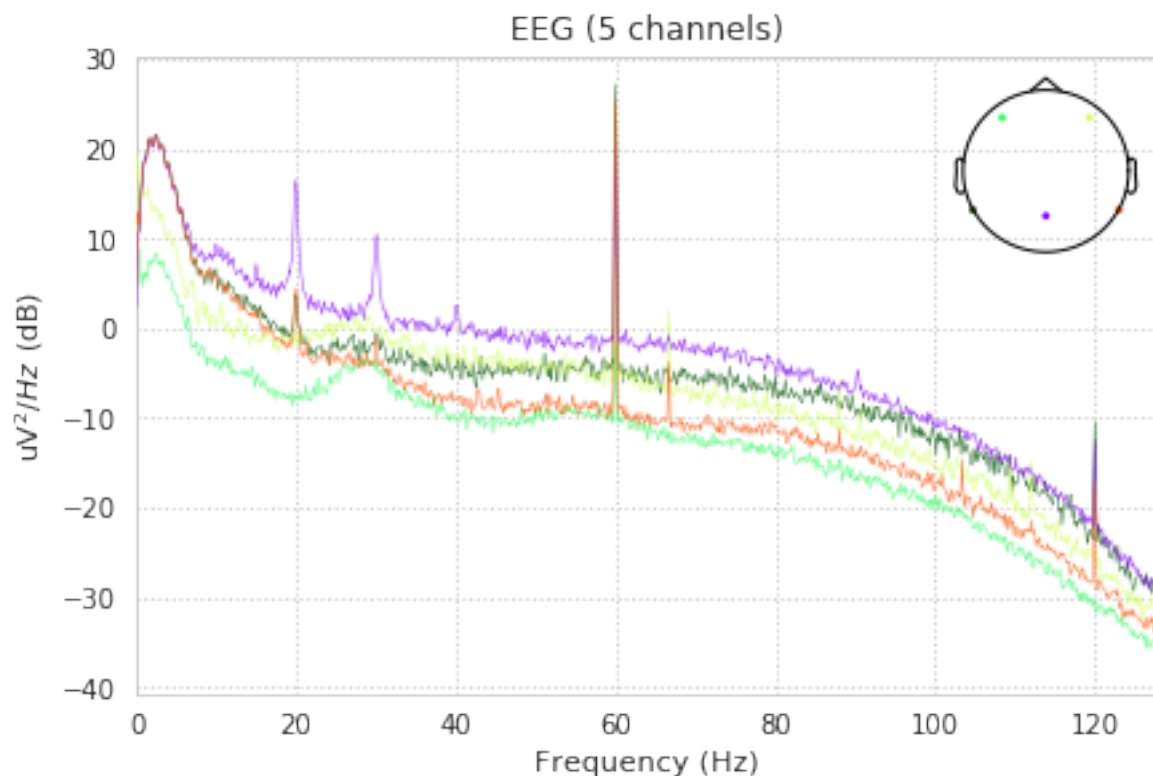
Range : 0 ... 30719 = 0.000 ... 119.996 secs
Ready.

Power Spectral Density

One way to analyze the SSVEP is to plot the power spectral density, or PSD. SSVEPs should appear as peaks in power for certain frequencies. We expect clear peaks in the spectral domain at the stimulation frequencies of 30 and 20 Hz.

```
In [67]: %matplotlib inline
raw.plot_psd();
```

Effective window size : 8.000 (s)



We can clearly see some peaks in the EEG at 20 and 30hz. The peaks are certainly strongest in the POz channel

Epoching

Next, we will chunk (epoch) the data into segments representing the data 100ms before to 800ms after each stimulus.

Note: we will not reject epochs here because the amplitude of the SSVEP at POz is so large it is difficult to separate from eye blinks

```
In [68]: events = find_events(raw)
event_id = {'30 Hz': 1, '20 Hz': 2}

epochs = Epochs(raw, events=events, event_id=event_id,
                 tmin=-0.5, tmax=4, baseline=None, preload=True,
                 verbose=False, picks=[0, 1, 2, 3, 4])
print('sample drop %: ', (1 - len(epochs.events)/len(events)) * 100)
```

```
197 events found
Event IDs: [1 2]
sample drop %: 2.538071065989844
```

Stimuli-Specific PSD

Next, we can compare the PSD of epochs specifically during 20hz and 30hz stimulus presentation

```
In [69]: from mne.time_frequency import psd_welch

f, axs = plt.subplots(2, 1, figsize=(10, 10))
psd1, freq1 = psd_welch(epochs['30 Hz'], n_fft=1028, n_per_seg=256 * 3)
psd2, freq2 = psd_welch(epochs['20 Hz'], n_fft=1028, n_per_seg=256 * 3)
psd1 = 10 * np.log10(psd1)
psd2 = 10 * np.log10(psd2)

psd1_mean = psd1.mean(0)
psd1_std = psd1.mean(0)

psd2_mean = psd2.mean(0)
psd2_std = psd2.mean(0)

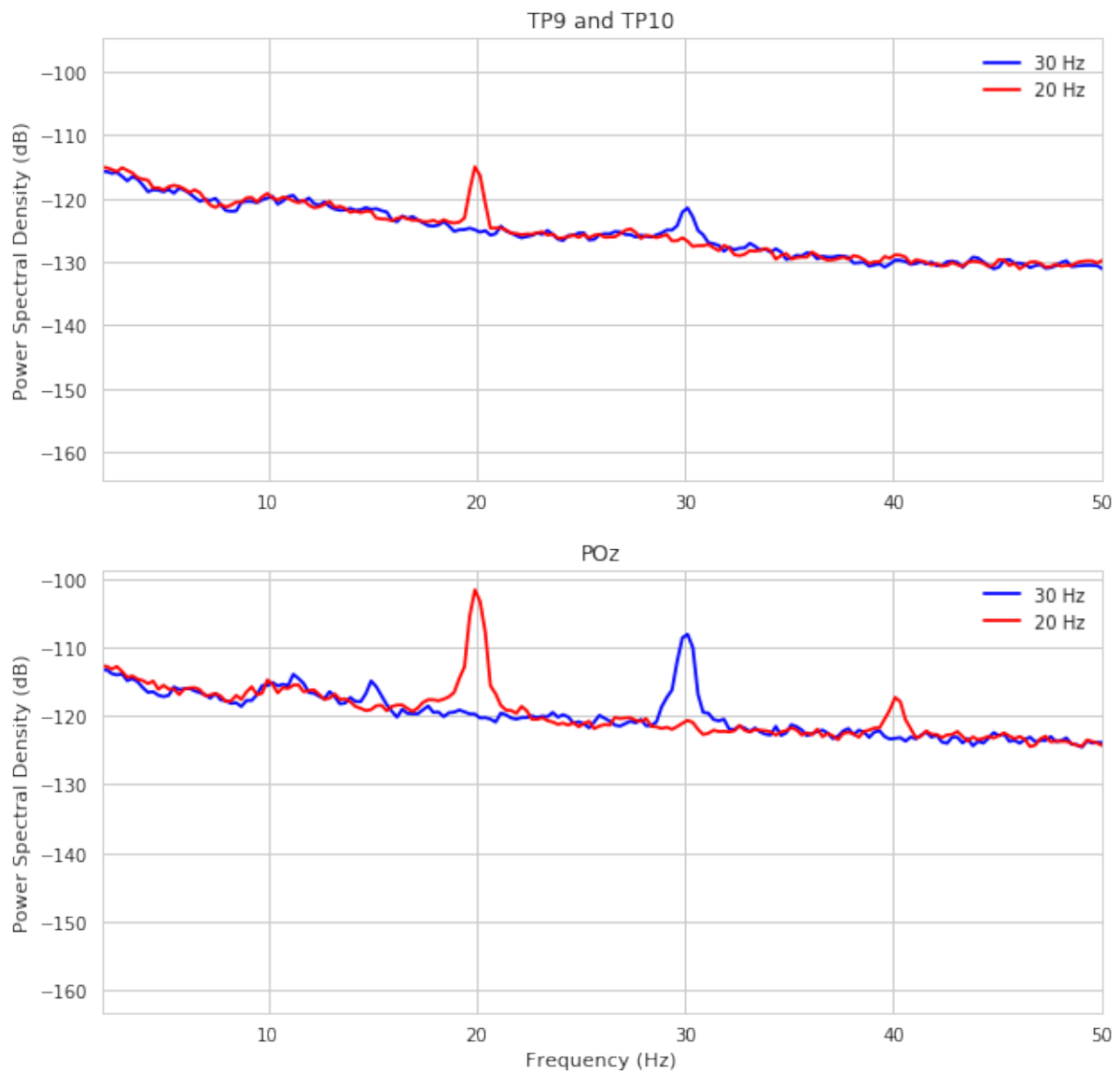
axs[0].plot(freq1, psd1_mean[[0, 3], :].mean(0), color='b', label='30 Hz')
axs[0].plot(freq2, psd2_mean[[0, 3], :].mean(0), color='r', label='20 Hz')

axs[1].plot(freq1, psd1_mean[4, :], color='b', label='30 Hz')
axs[1].plot(freq2, psd2_mean[4, :], color='r', label='20 Hz')

axs[0].set_title('TP9 and TP10')
axs[1].set_title('POz')
axs[0].set_ylabel('Power Spectral Density (dB)')
axs[1].set_ylabel('Power Spectral Density (dB)')
axs[0].set_xlim((2, 50))
axs[1].set_xlim((2, 50))
axs[1].set_xlabel('Frequency (Hz)')
axs[0].legend()
axs[1].legend()

plt.show();

Effective window size : 4.016 (s)
Effective window size : 4.016 (s)
```



With this visualization we can clearly see distinct peaks at 30hz and 20hz in the PSD, corresponding to the frequency of the visual stimulation. The peaks are much larger at the POz electrode, but still visible at TP9 and TP10

Spectrogram

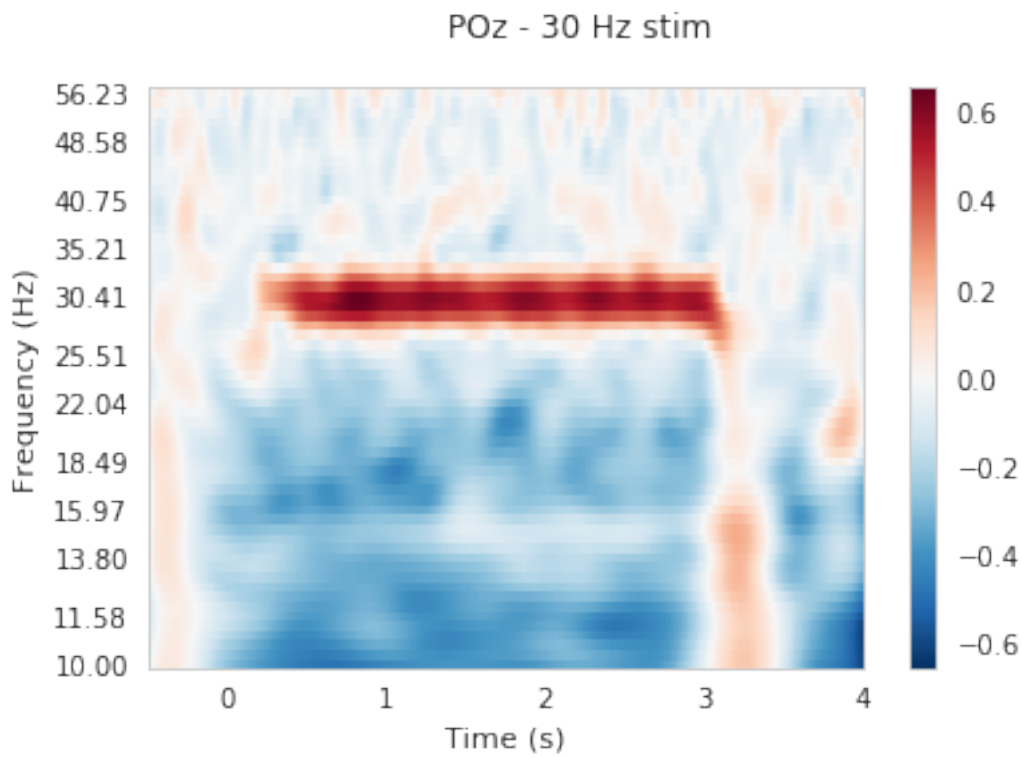
We can also look for SSVEPs in the spectrogram, which uses color to represent the power of frequencies in the EEG signal over time

```
In [70]: from mne.time_frequency import tfr_morlet

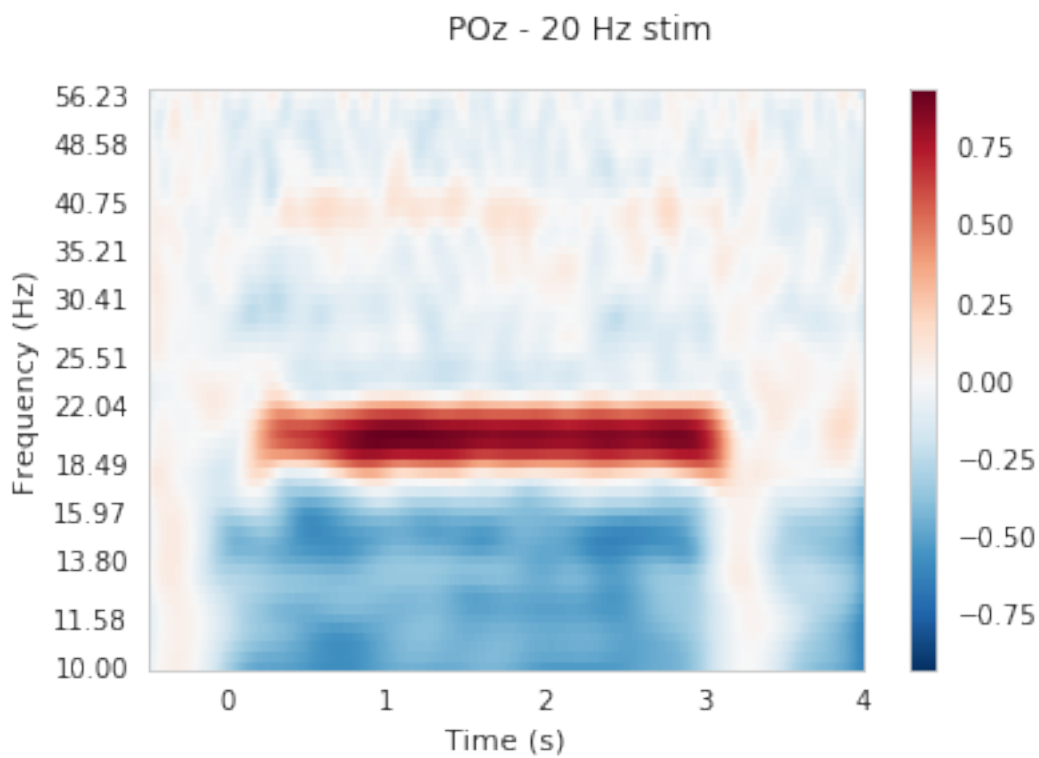
frequencies = np.logspace(1, 1.75, 60)
tfr, itc = tfr_morlet(epochs['30 Hz'], freqs=frequencies,
                      n_cycles=15, return_itc=True)
tfr.plot(picks=[4], baseline=(-0.5, -0.1), mode='logratio',
         title='POz - 30 Hz stim');

tfr, itc = tfr_morlet(epochs['20 Hz'], freqs=frequencies,
                      n_cycles=15, return_itc=True)
tfr.plot(picks=[4], baseline=(-0.5, -0.1), mode='logratio',
         title='POz - 20 Hz stim');
```

Applying baseline correction (mode: logratio)



Applying baseline correction (mode: logratio)



Once again we can see clear SSVEPs at 30hz and 20hz

Decoding

We can use a filter bank approach on the original 4 Muse electrodes (to see how the headband alone without external electrodes could be used to classify SSVEP):

1. Apply bandpass filters around both stimulation frequencies
2. Concatenate bandpass-filtered channels
3. Extract epochs (from 1 to 3 s after stimulus onset, to avoid classifying the ERP)
4. Apply common classification pipelines

In [71]: *# Bandpass filter the raw data*

```
muse_raw = raw.drop_channels(['P0z'])
raw_filt_30Hz = muse_raw.copy().filter(25, 35, method='iir')
raw_filt_20Hz = muse_raw.copy().filter(15, 25, method='iir')
raw_filt_30Hz.rename_channels(lambda x: x + '_30Hz')
raw_filt_20Hz.rename_channels(lambda x: x + '_20Hz')
```

Concatenate with the bandpass filtered channels

```
raw_all = raw_filt_30Hz.add_channels([raw_filt_20Hz],
                                     force_update_info=True)
```

Extract epochs

```
events = find_events(raw_all)
event_id = {'30 Hz': 1, '20 Hz': 2}
```

```
epochs_all = Epochs(raw_all, events=events, event_id=event_id, tmin=1,
                    tmax=3, baseline=None, reject={'eeg': 100e-6},
                    preload=True, verbose=False,)
```

Setting up band-pass filter from 25 - 35 Hz

Setting up band-pass filter from 25 - 35 Hz

Setting up band-pass filter from 25 - 35 Hz

Setting up band-pass filter from 25 - 35 Hz

Setting up band-pass filter from 25 - 35 Hz

Setting up band-pass filter from 15 - 25 Hz

Setting up band-pass filter from 15 - 25 Hz

Setting up band-pass filter from 15 - 25 Hz

Setting up band-pass filter from 15 - 25 Hz

Setting up band-pass filter from 15 - 25 Hz

Setting up band-pass filter from 15 - 25 Hz

Setting up band-pass filter from 15 - 25 Hz

197 events found

Event IDs: [1 2]

197 events found

Event IDs: [1 2]

Some events are duplicated in your different stim channels. 197 events were ignored during deduplication.

In [72]: `epochs_all.pick_types(eeg=True)`

```
X = epochs_all.get_data() * 1e6
```

```
times = epochs.times
```

```
y = epochs_all.events[:, -1]
```

Decoding the N170

Next, we will use 4 different machine learning pipelines to classify the SSVEP based on the data we collected. The

- **CSP + RegLDA** : Common Spatial Patterns + Regularized Linear Discriminant Analysis. This is a very common EEG analysis pipeline.
- **Cov + TS** : Covariance + Tangent space mapping. One of the most reliable Riemannian geometry-based pipelines.
- **Cov + MDM**: Covariance + MDM. A very simple, yet effective (for low channel count), Riemannian geometry classifier.
- **CSP + Cov + TS**: Common Spatial Patterns + Covariance + Tangent space mapping. Riemannian pipeline with the standard CSP procedure beforehand

Evaluation is done through cross-validation, with area-under-the-curve (AUC) as metric (AUC is probably the best metric for binary and unbalanced classification problem)

Note: because we're doing machine learning here, the following cell may take a while to complete

```
In [75]: import pandas as pd
        from sklearn.pipeline import make_pipeline

        from mne.decoding import Vectorizer

        from sklearn.linear_model import LogisticRegression
        from sklearn.preprocessing import StandardScaler
        from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

        from sklearn.model_selection import cross_val_score, StratifiedShuffleSplit

        from pyriemann.estimation import Covariances, ERPCovariances, XdawnCovariances
        from pyriemann.spatialfilters import CSP
        from pyriemann.tangentspace import TangentSpace
        from pyriemann.classification import MDM

        from collections import OrderedDict

        clfs = OrderedDict()

        clfs['CSP + RegLDA'] = make_pipeline(Covariances(), CSP(4), LDA(shrinkage='auto', solver='eigen'))
        clfs['Cov + TS'] = make_pipeline(Covariances(), TangentSpace(), LogisticRegression())
        clfs['Cov + MDM'] = make_pipeline(Covariances(), MDM())
        clfs['CSP + Cov + TS'] = make_pipeline(Covariances(), CSP(4, log=False), TangentSpace(), LogisticRegression())

        # define cross validation
        cv = StratifiedShuffleSplit(n_splits=20, test_size=0.25,
                                   random_state=42)

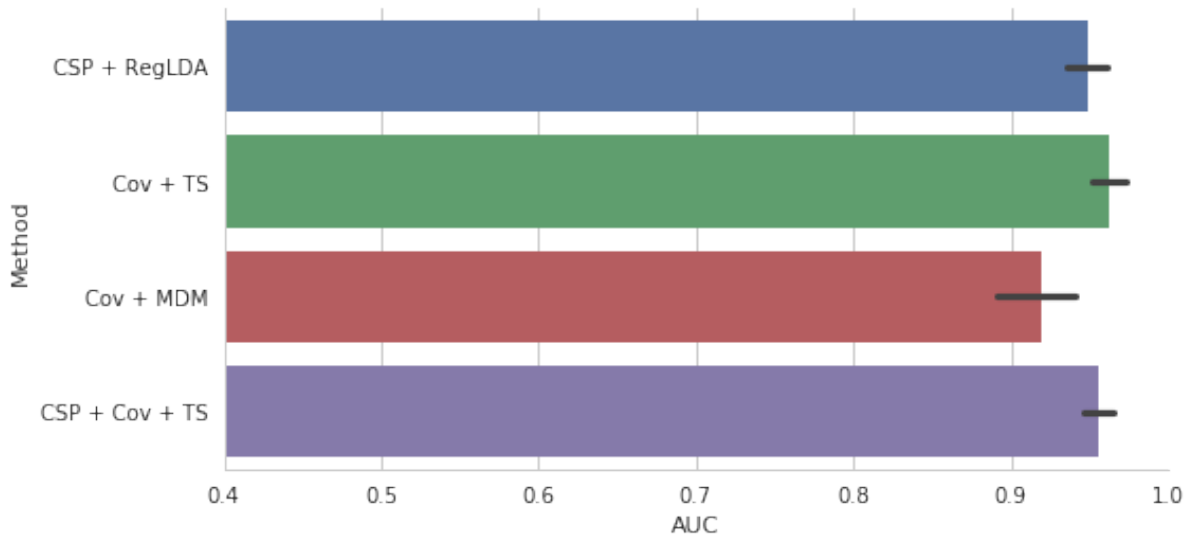
        # run cross validation for each pipeline
        auc = []
        methods = []
        for m in clfs:
            print(m)
            try:
                res = cross_val_score(clfs[m], X, y==2, scoring='roc_auc',
                                      cv=cv, n_jobs=-1)
                auc.extend(res)
                methods.extend([m]*len(res))
            except:
                pass

        results = pd.DataFrame(data=auc, columns=['AUC'])
```

```

results['Method'] = methods
CSP + RegLDA
Cov + TS
Cov + MDM
CSP + Cov + TS
In [74]: fig = plt.figure(figsize=[8,4])
         sns.barplot(data=results, x='AUC', y='Method')
         plt.xlim(0.4, 1)
         sns.despine()

```



The different classifiers get some impressive accuracy on this dataset, all around .95 AUC. This is impressive considering this pipeline only included data from the temporal electrodes

6.5 Step 6: Share your Data!

How did your experiment go? If you're excited by your results we'd love to see your data!

Follow the instructions on our [Contributions](https://github.com/NeuroTechX/eeg-notebooks/blob/master/CONTRIBUTING.md)¹⁵ page to make a pull request with your data and we'll review it to be added to the EEG notebooks project.

End of doc/ssvep.nblink

¹⁵ <https://github.com/NeuroTechX/eeg-notebooks/blob/master/CONTRIBUTING.md>

7 Using an extra electrode with Muse

Although the Muse is wonderful for ease-of-use and affordability, it suffers from a small number of electrode locations and inflexibility of electrode positioning. Fortunately, in order to partially overcome this limitation, the Muse hardware team has allowed an extra electrode to be added to the Muse 2016.

7.1 The electrode

These electrodes are not for sale anywhere; they must be made by hand. Fortunately, their construction appears pretty simple, attach an EEG electrode (any kind) to a male microUSB port with a wire.

We'll update this section with more info as it comes in from the Muse hardware team.



7.2 Attaching the extra electrode

The extra electrode can be applied anywhere on the head (provide the wire is long enough). Just inset the electrode's microUSB connector into the charging port of the Muse. In order to make sure the electrode stays in place, we recommend using a hat or scarf as pictured.



7.3 Getting data from the electrode

With the extra electrode connected to the Muse, its data is available as the Right AUX channel in the muse-lsl data stream. It will automatically appear in muse-lsl's viewer. An example of how to access this data and include it in your

analysis is shown in the P300 with Extra Electrode¹⁶ notebook

8 Technical Information about the MUSE

8.1 MUSE setup and usage

There is a lot of excellent information on MUSE setup and usage on the website of the Krigolson lab website¹⁷ at the University of Victoria, BC.

The following instructional videos are particularly worth checking out:

Introduction to the MUSE

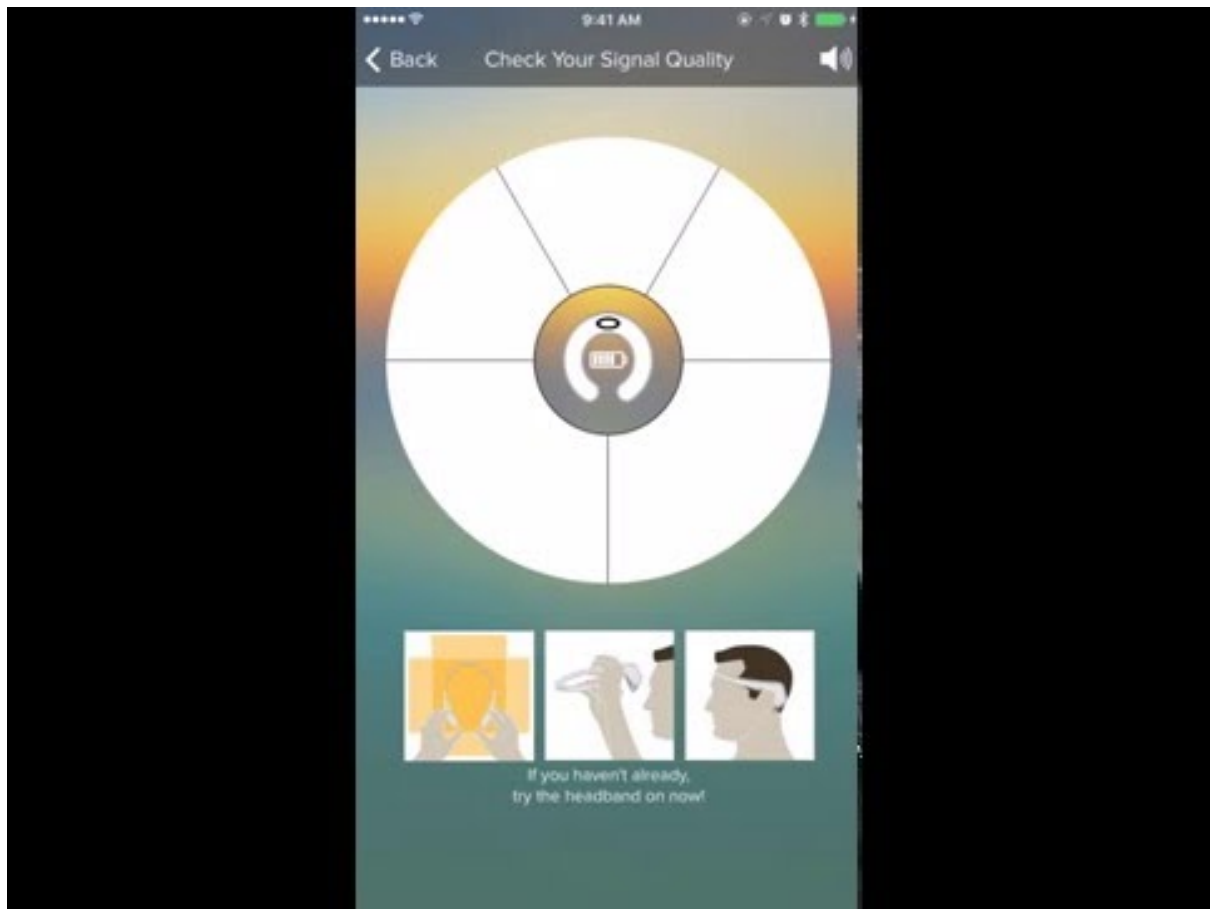


¹⁶ <https://github.com/NeuroTechX/eeg-notebooks/blob/master/notebooks/P300%20with%20Extra%20Electrode.ipynb>

¹⁷ <http://www.krigolsonlab.com/muse-help.html>

¹⁸ <https://youtu.be/LihwJxzJALw?t=1s>

Headband fit and signal quality tutorial



¹⁹ <https://youtu.be/Y-tF3iiolHU?t=1s>

Adjusting and fitting the MUSE for better signal quality



²⁰ <https://youtu.be/v8xUYqqJAig?t=1s>

Using water for better signal quality



21

8.2 Details related to Muse ERP analysis

Latency and jitter related from the Muse is 40ms \pm 20ms

In Krigolson lab's resampling analysis, a sample size of 10 was found to be needed for high statistical accuracy for N200 and reward positivity, which is similar to traditional EEG, but greater numbers of subjects were needed for P300

²¹ <https://youtu.be/gKtVIVCDHGg?t=1s>