Meditation and Neural Activities: Replication & Classifier Development

December 9, 2019

Meditation and Neural Activities: *Replication & Classifier Development*

Final Data Science Neuroscience Project A replication of Brandmeyer & Delorme (2018), with data-driven techniques. Creating new supervised and unsupervised model for classifier of expert vs. non-expert labels in mediation, using EEG brain activities.

This project is available at https://github.com/yuyang-zhong/EEG-Neural.

Yuyang Zhong

University of California, Berkeley

December 9, 2019

Cognitive Neuroscience

Jack L. Gallant, Ph.D., *Professor*

Manon Ironside, *Graduate Student Instructor*

---

1 Introduction

1.1 Background

Meditation had been claimed to have a lot of physical and mental effects for individuals who actively practice meditation on a regular basis. However, to psychologists and neuroscientists, researchers are more interested in how these effects show up. Past research had focused on whether there was significant changes in subjects' neural activities when engaged in meditation over a period of time.

The current research that this project is based on, conducted by Brandmeyer & Delorme in 2016, focuses on whether there was a significant difference in "depth" of neural activity (measured through EEG) for those who practice meditation on a more frequent basis ("expert") compared to those on a less frequent basis ("non-expert").

This project will dive into understanding the published data better, and see if a classifier could be built to label "expert" vs. "non-expert" based on neural activities during meditation.

1.2 Motivation & Significance

In the literature, probes into meditation had lead researchers to find evidence of a default-mode network, as well as differences in functional connectivity of brain activities (Berkovich-Ohana et al. 2016;, Garrison et al. 2015).

The present research (Brandmeyer & Delorme 2016), as well as this project, can potentially provide evidence for whether meditation alters one's default-mode network and functional connectivity, and attributing those change to the benefits claimed by individuals practicing meditation as part of their daily lives. Authors of this paper were also trying to probe whether default mode network was related to the frequency of mind wandering episodes during meditation, especially those not aware by the individual (Christoff et al. 2009). This project, however, will not focus as much on mind wandering, but more so on the classification of subjects for their meditation expertise.

2 Method

2.1 Dataset Overview

This dataset was made available by the authors of the present research (Brandmeyer & Delorme 2016), at multiple open data repositories. Version 2.0 of the data, published on November 19, 2018, was downloaded from **Zenodo** (https://doi.org/10.5281/zenodo.2536267).

2.1.1 Description by Author

This meditation experiment contains 24 subjects. Subjects were meditating and were interrupted about every 2 minutes to indicate their level of concentration and mind wandering.

2.1.2 Dataset organization

The dataset is organized in the Brain Image Data Structure (BIDS) format. The raw data contains the MATLAB code for session, sound files to the stimuli, folders for each subject, within with folders for each session the subject participated in. In the session folders the eeg measures and event files are provided.

2.2 Methods & Techniques of the Original Research

(Section referenced and adopted from original research article.)

EEG data was collected, using a 64-channel BioSemi system and and BioSemi 10-20 head cap montage. There are a total of 64 channels (locations of measure), mapped by the `Biosemi64Alpha` montage (not part of the standard `mne` packages, direction to load this custom montage is included below). This measure has very well temporal resolution but poor spatial resolution.

A total of 24 participants were in this study. Participants were asked to meditate for 30-90 seconds, and interrupted to rate their mindfulness depth and mind wandering level. This project will solely focus on the onset of that interruption, the period of meditation before that, as well as the short period right after in response to the interruption.

3 Data Analysis

3.1 Outline of Analysis

The following table summarizes the methods and techniques used in each section of this project.

Section	Methods	Motivation
Data Exploration	Time frequency analysis	Compute Time-Frequency Representation (TFR) using Morlet wavelet, and seeing if I can identify concentrations of epochs to focus on.
Data Exploration	Topographic Mapping: All Evoked Response	Looking at average brain activities near an event.
Data Exploration	Event-related Spectral Perturbation (ERSP): Onset Evoked Response	This would be important to help us understand whether this question is actually valid to ask - is there a difference in onset evoked response, and activities before that between the 2 subject groups?
Data Cleaning	-	The data will be shrunken down to an average of all evoked responses, 10 seconds before the onset and 5 after, for each individual subjects. This will be used for the classifier.
Data Analysis	Correlation Matrix	Looking at which channels are most and least correlated with each other.
Data Analysis	Independent Component Analysis (ICA) & Principal Component Analysis (PCA)	Looking at which channels are most significant in contribution to the overall brain activity, and try to see if I can figure out why. This would help identify components to use for our model.
Classifier	Logistic Regression with 5-Fold Cross Validation	The simpler method.
Classifier	Neural Network: Multi-Layer Perceptron Classifier	A bit fancier method to improve accuracy.
Classifier	Random Forest	A bit fancier method to improve accuracy.

3.2 Project Setup & Imports

3.2.1 Project Dependencies

This project utilizes the following Python packages: `numpy`, `pandas`, `matplotlib`, `seaborn`, `mne`, and `sklearn`.

To install a package within this jupyter notebook, utilize the command `!pip install [package-name]`. The `!` will allow you to run command line prompt within this notebook.

3.2.2 Importing Packages

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

import mne

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.model_selection import KFold
from sklearn.decomposition import PCA, FastICA
from sklearn.metrics import classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier
```

3.2.3 Suppresses Warnings

This is used for exporting the final PDF file without the warning messages. Feel free to comment this out.

[2]:
```
import warnings
warnings.filterwarnings('ignore')
```

3.3 Data Exploration on Subject 1 (Non-Expert) & 15 (Expert)

3.3.1 Importing data

[3]:
```
raw_fname1 = '../rawdata/bidsexport/sub-001/ses-01/eeg/
 ↪sub-001_ses-01_task-meditation_eeg.bdf'
raw_fname15 = '../rawdata/bidsexport/sub-015/ses-01/eeg/
 ↪sub-015_ses-01_task-meditation_eeg.bdf'

raw1 = mne.io.read_raw_bdf(raw_fname1, preload=True)
raw15 = mne.io.read_raw_bdf(raw_fname15, preload=True)
```

```
Extracting EDF parameters from /Users/yuyang.zhong/eeg/rawdata/bidsexport/sub-00
1/ses-01/eeg/sub-001_ses-01_task-meditation_eeg.bdf…
BDF file detected
Setting channel info structure…
Creating raw.info structure…
Reading 0 … 696575  =      0.000 …  2720.996 secs…
Extracting EDF parameters from /Users/yuyang.zhong/eeg/rawdata/bidsexport/sub-01
5/ses-01/eeg/sub-015_ses-01_task-meditation_eeg.bdf…
BDF file detected
```

```
Setting channel info structure…
Creating raw.info structure…
Reading 0 … 695807  =      0.000 …  2717.996 secs…
```

For the purpose of this project, we will remove all of the "channels" that are metadata of the subject/experiment. We are focusing only on the 64 EEG channels.

```
[4]: raw1.drop_channels(['EXG1', 'EXG2', 'EXG3', 'EXG4', 'EXG5', 'EXG6', 'EXG7',␣
      ↪'EXG8',
                         'GSR1', 'GSR2', 'Erg1', 'Erg2', 'Resp', 'Plet', 'Temp'])
     raw15.drop_channels(['EXG1', 'EXG2', 'EXG3', 'EXG4', 'EXG5', 'EXG6', 'EXG7',␣
      ↪'EXG8',
                         'GSR1', 'GSR2', 'Erg1', 'Erg2', 'Resp', 'Plet', 'Temp'])
```

```
[4]: <RawEDF  |  sub-015_ses-01_task-meditation_eeg.bdf, n_channels x n_times : 65 x
     695808 (2718.0 sec), ~345.2 MB, data loaded>
```

3.3.2 Loading & Setting Custom Montage `biosemi64alpha`

Since the researchers used an alphabetical (A/B) version of the standard `biosemi64` montage, we will need to load our own montage file to allow appropriate topographical mapping.

```
[5]: from os.path import abspath
     montage = mne.channels.read_montage(abspath("../biosemi64alpha.txt"))

     raw1.set_montage(montage);
     raw15.set_montage(montage);
```

3.3.3 Subject Information

Let's print 1 subject's EEG information.

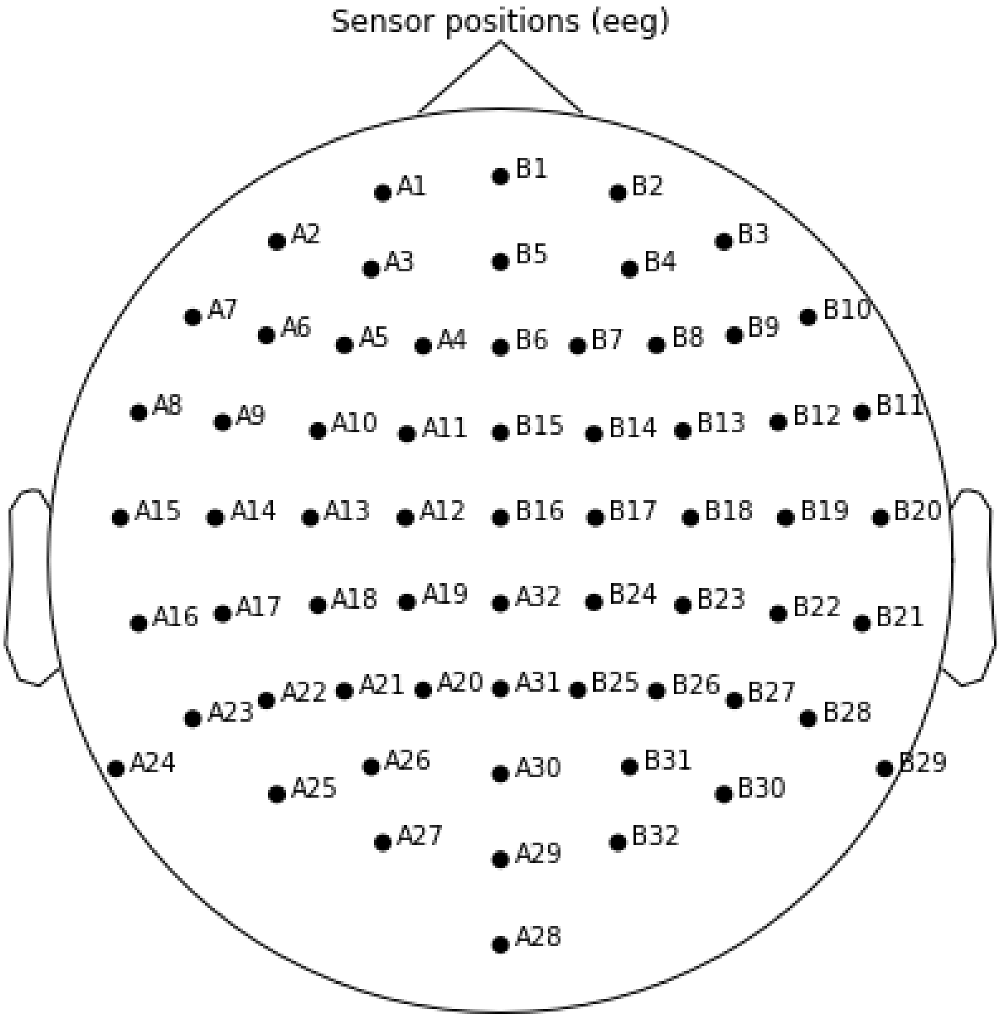
```
[6]: print(raw1)
     print(raw1.info)
```

```
<RawEDF  |  sub-001_ses-01_task-meditation_eeg.bdf, n_channels x n_times : 65 x
696576 (2721.0 sec), ~345.6 MB, data loaded>
<Info | 17 non-empty fields
    bads : list | 0 items
    ch_names : list | A1, A2, A3, A4, A5, A6, A7, A8, A9, …
    chs : list | 65 items (EEG: 64, STIM: 1)
    comps : list | 0 items
    custom_ref_applied : bool | False
    dev_head_t : Transform | 3 items
    dig : Digitization | 67 items (3 Cardinal, 64 EEG)
    events : list | 0 items
    highpass : float | 0.0 Hz
    hpi_meas : list | 0 items
    hpi_results : list | 0 items
```

```
    lowpass : float | 52.0 Hz
    meas_date : tuple | 2014-04-04 19:40:17 GMT
    nchan : int | 65
    proc_history : list | 0 items
    projs : list | 0 items
    sfreq : float | 256.0 Hz
    acq_pars : NoneType
    acq_stim : NoneType
    ctf_head_t : NoneType
    description : NoneType
    dev_ctf_t : NoneType
    device_info : NoneType
    experimenter : NoneType
    file_id : NoneType
    gantry_angle : NoneType
    helium_info : NoneType
    hpi_subsystem : NoneType
    kit_system_id : NoneType
    line_freq : NoneType
    meas_id : NoneType
    proj_id : NoneType
    proj_name : NoneType
    subject_info : NoneType
    utc_offset : NoneType
    xplotter_layout : NoneType
>
```

The location of the sensors/channels are shown below. As you can see, the labels of the channels begin with A & B as aligned with the left/right hemispheres, instead of the specific naming of the standard `biosemi64` channel names.

[7]:
```
mne.viz.plot_sensors(raw1.info, ch_type='eeg', show_names=True);
```



3.3.4 Events & Epochs

What kind of events are there in this session? This prints the top 5 events found for subject 1.

[8]:
```
events1 = mne.find_events(raw1, stim_channel='Status')
events15 = mne.find_events(raw15, stim_channel='Status')
print(events1[:5])
print(events15[:5])
```

```
Trigger channel has a non-zero initial value of 65536 (consider using
initial_event=True to detect this event)
Removing orphaned offset at the beginning of the file.
87 events found
Event IDs: [  2   4 128]
```

```
Trigger channel has a non-zero initial value of 65536 (consider using
initial_event=True to detect this event)
Removing orphaned offset at the beginning of the file.
84 events found
Event IDs: [  2   4 128]
[[18275     0   128]
 [19387     0     2]
 [20422     0     2]
 [32156     0   128]
 [46029     0   128]]
[[35569     0   128]
 [36506     0     2]
 [37545     0     4]
 [69254     0   128]
 [70197     0     4]]
```

From the `task-meditation_events.json` file, we found the following ID corresponding the events.

[9]:
```python
import json
with open("../rawdata/bidsexport/task-meditation_events.json", "r") as␣
↪events_file:
    events = json.load(events_file)

event_dict = {i: d for d, i in events['value']['Levels'].items()}
print(event_dict)
```

```
{'Response 1 (this may be a response to question 1, 2 or 3)': '2', 'Response 2
(this may be a response to question 1, 2 or 3)': '4', 'Response 3 (this may be a
response to question 1, 2 or 3)': '8', 'Indicate involuntary response': '16',
'First question onset (most important marker)': '128'}
```

Since only 3 events were used in the dataset, we will manually load those using the printout above.

[10]:
```python
event_dict = {'Response 1 (this may be a response to question 1, 2 or 3)': 2,
              'Response 2 (this may be a response to question 1, 2 or 3)': 4,
              'First question onset (most important marker)': 128}
```

Setting epochs: We are only interested in 10 seconds before the onset and 5 seconds after.

[11]:
```python
epochs1 = mne.Epochs(raw1, events1, event_id=event_dict, tmin=-10, tmax=5,␣
↪preload=True)
epochs15 = mne.Epochs(raw15, events15, event_id=event_dict, tmin=-10, tmax=5,␣
↪preload=True)
```

```
87 matching events found
Applying baseline correction (mode: mean)
Not setting metadata
0 projection items activated
Loading data for 87 events and 3841 original time points …
```

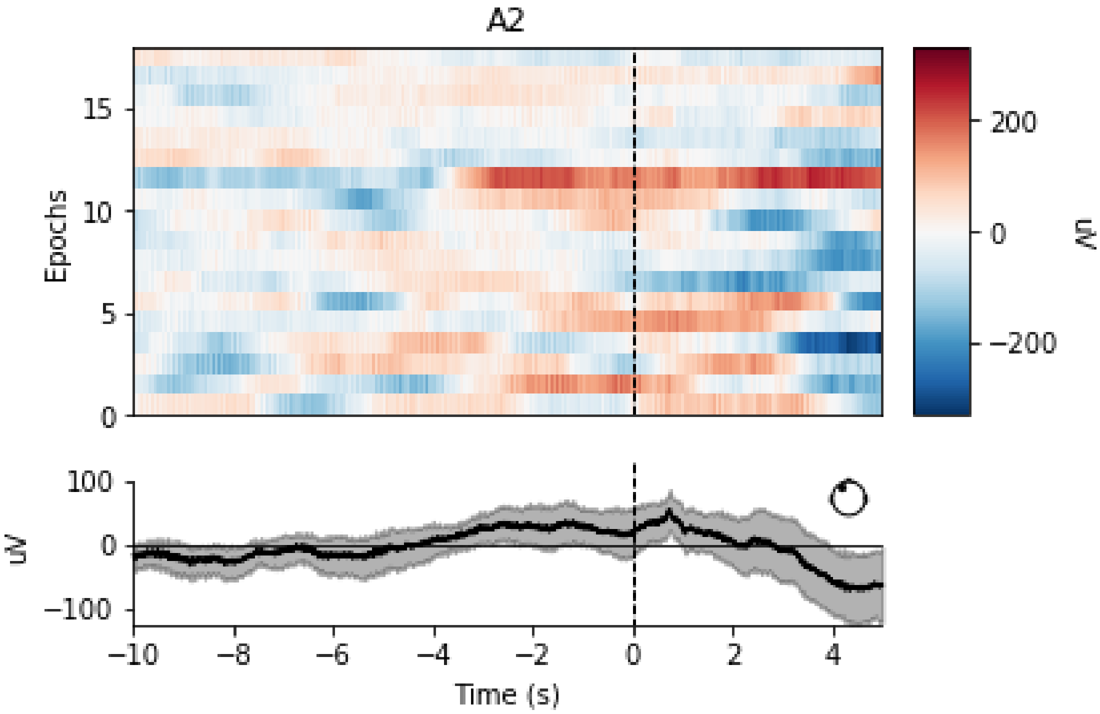
```
0 bad epochs dropped
84 matching events found
Applying baseline correction (mode: mean)
Not setting metadata
0 projection items activated
Loading data for 84 events and 3841 original time points …
0 bad epochs dropped
```

We will then select the 3 conditions we have noted earlier (the 3 events above) and equalize them. Then we will select epochs related to these conditions.

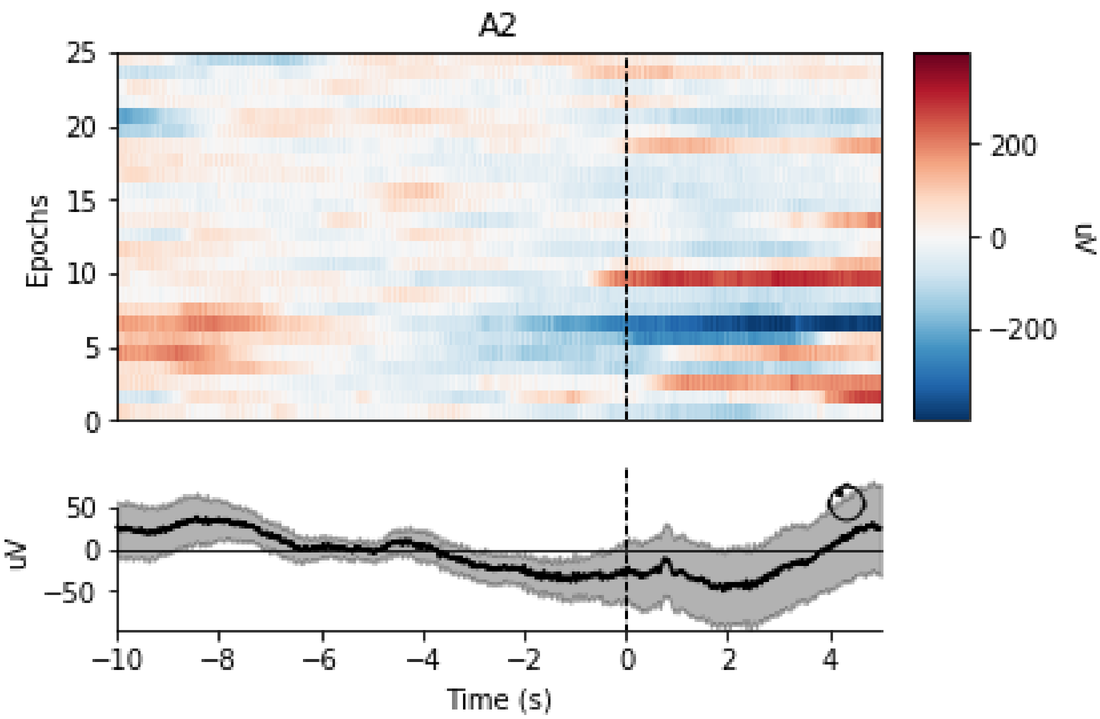
Since we only care about the onset, we will visualize the `onset_epoch` at channel `A2`, which in located in the prefrontal cortex.

[12]:
```
conds = ['Response 1 (this may be a response to question 1, 2 or 3)',
                       'Response 2 (this may be a response to question 1, 2 or␣
 ↪3)',
                      'First question onset (most important marker)']
epochs1.equalize_event_counts(conds)
epochs15.equalize_event_counts(conds)

r1_epochs1 = epochs1['Response 1 (this may be a response to question 1, 2 or␣
 ↪3)']
r2_epochs1 = epochs1['Response 2 (this may be a response to question 1, 2 or␣
 ↪3)']
onset_epochs1 = epochs1['First question onset (most important marker)']
onset_epochs1.plot_image(picks=['A2']);

r1_epochs15 = epochs15['Response 1 (this may be a response to question 1, 2 or␣
 ↪3)']
r2_epochs15 = epochs15['Response 2 (this may be a response to question 1, 2 or␣
 ↪3)']
onset_epochs15 = epochs15['First question onset (most important marker)']
onset_epochs15.plot_image(picks=['A2']);
```

```
Dropped 33 epochs
Dropped 9 epochs
18 matching events found
No baseline correction applied
Not setting metadata
0 projection items activated
0 bad epochs dropped
```




```
25 matching events found
No baseline correction applied
Not setting metadata
0 projection items activated
0 bad epochs dropped
```

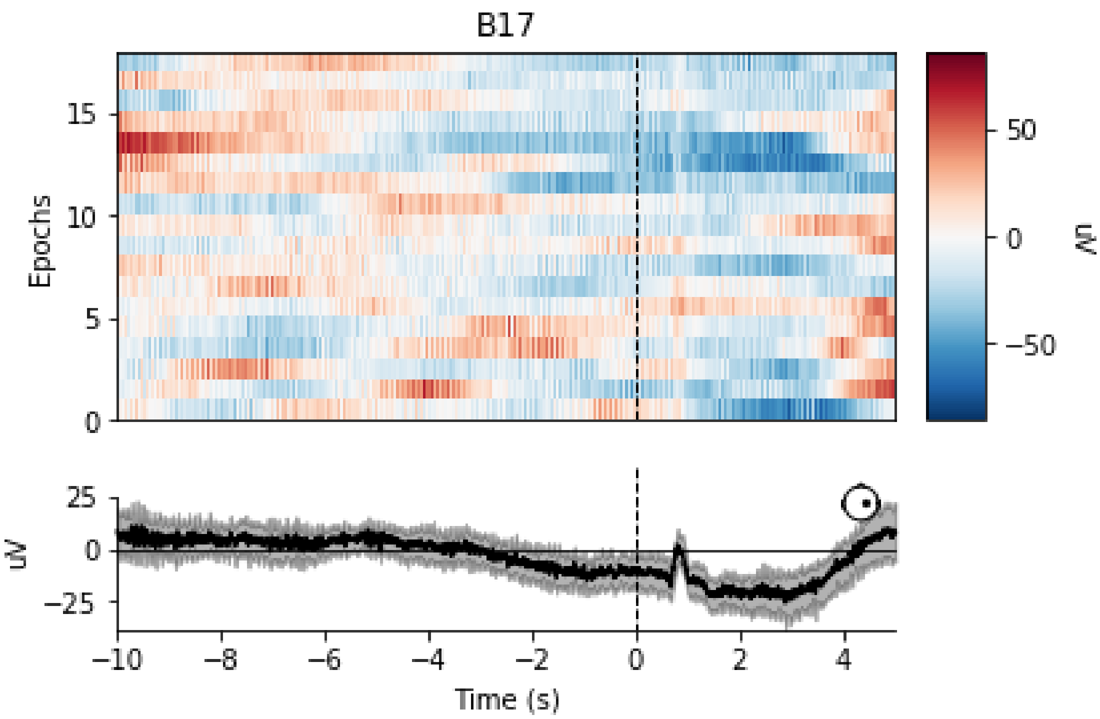


We can see that there was an interesting dip around the onset for our expert mediator (second plot), but some hill for the non-expert (first plot). The non-expert also has more variance in their activity, shown by the labels on the y-axis of the first plot. There may be related to mind wandering episodes, and that the non-expert may be invoking executive functions in the prefrontal cortex. It also seems that there are a totally inverse relationship for this channel.

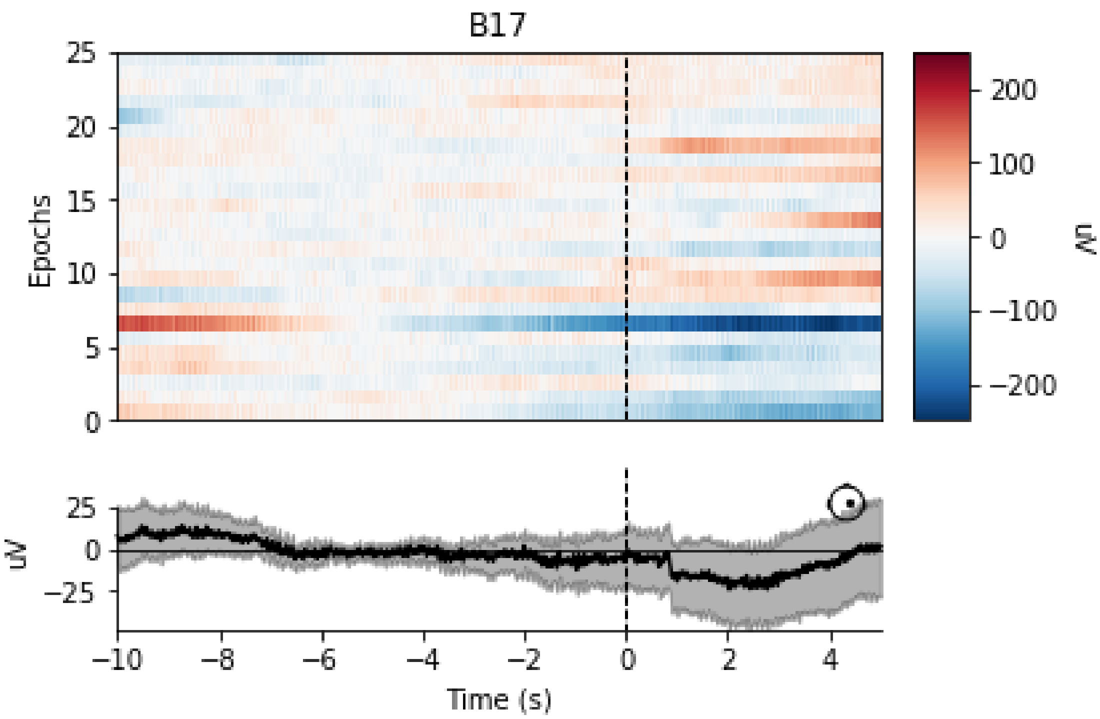
Let's try another channel, `B17`, located by the parietal lobe.

[13]:
```
onset_epochs1.plot_image(picks=['B17']);
onset_epochs15.plot_image(picks=['B17']);
```

```
18 matching events found
No baseline correction applied
Not setting metadata
0 projection items activated
0 bad epochs dropped
```




```
25 matching events found
No baseline correction applied
Not setting metadata
0 projection items activated
0 bad epochs dropped
```



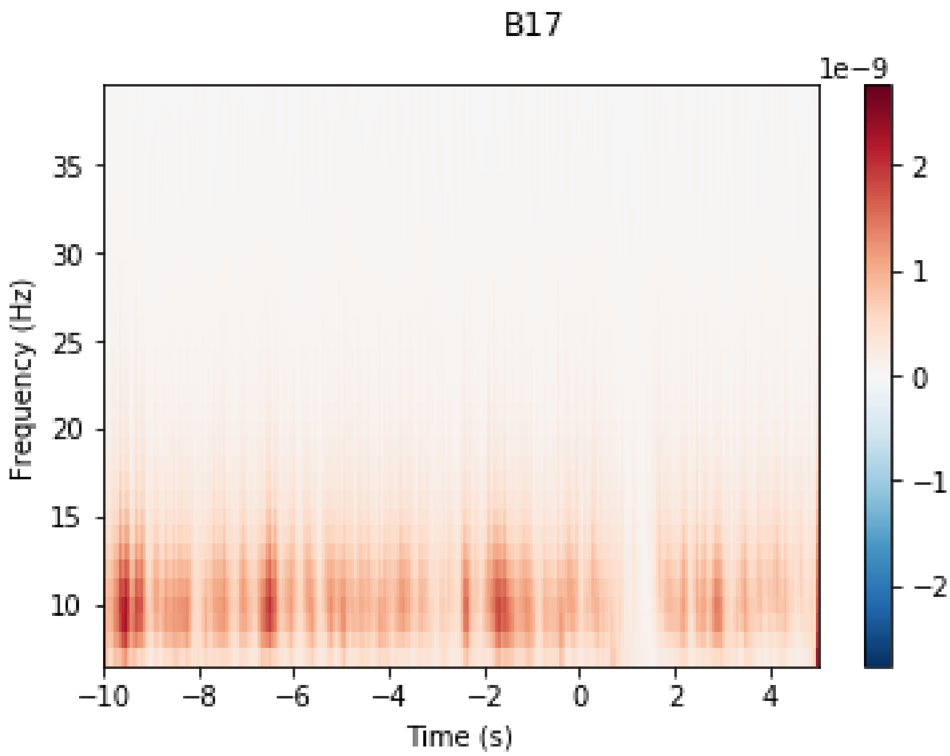
B17 is located in parietal lobe, and many research had found that this region have activity correlated with the default-mode network and overall functional connectivity of the brain. As you can see, the 2 subjects actually display similar pattern for the meditation, but a clear spike of activity in the non-expert (first plot) around the first second after the onset of interruption. We can also see a greater variability in activities before the onset. Does this tell us about the more “smooth” brain activity, even in response to interruption, for expert meditators?

3.3.5 Time Frequency Analysis

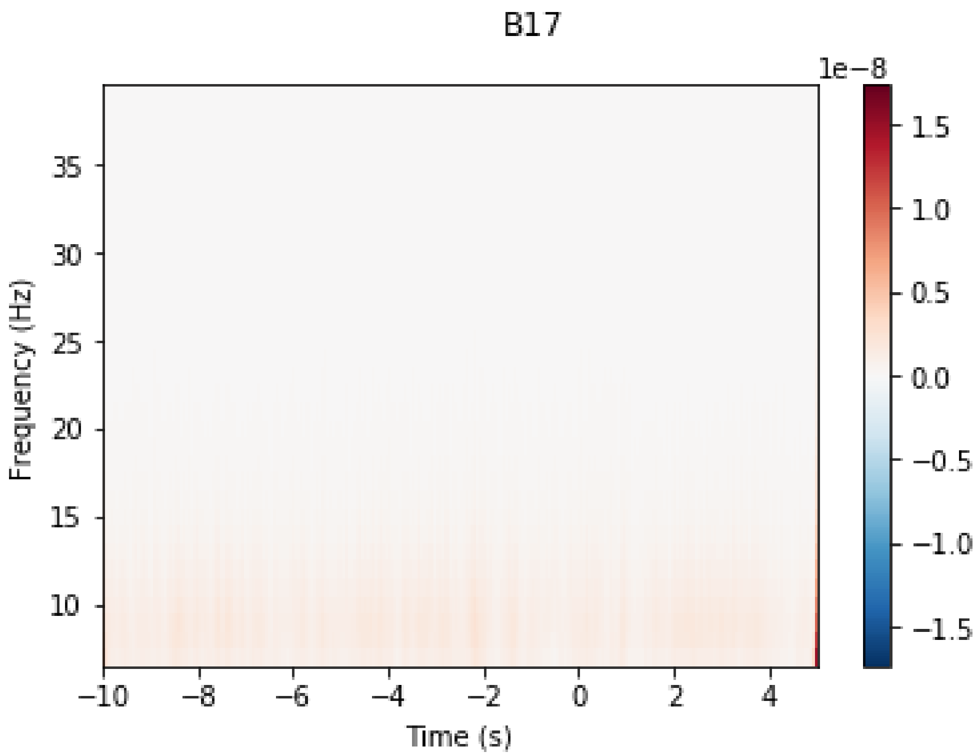
Since channel B17 seem to be showing some interesting stuff, let's run a time frequency analysis for both subjects at this channel.

[14]:
```
frequencies = np.arange(7, 40, 1)
power1 = mne.time_frequency.tfr_morlet(onset_epochs1, n_cycles=3,␣
 ↪return_itc=False,
                                        freqs=frequencies, decim=3)
power15 = mne.time_frequency.tfr_morlet(onset_epochs15, n_cycles=3,␣
 ↪return_itc=False,
                                        freqs=frequencies, decim=3)
power1.plot(['B17']);
power15.plot(['B17']);
```

```
No baseline correction applied
```



No baseline correction applied



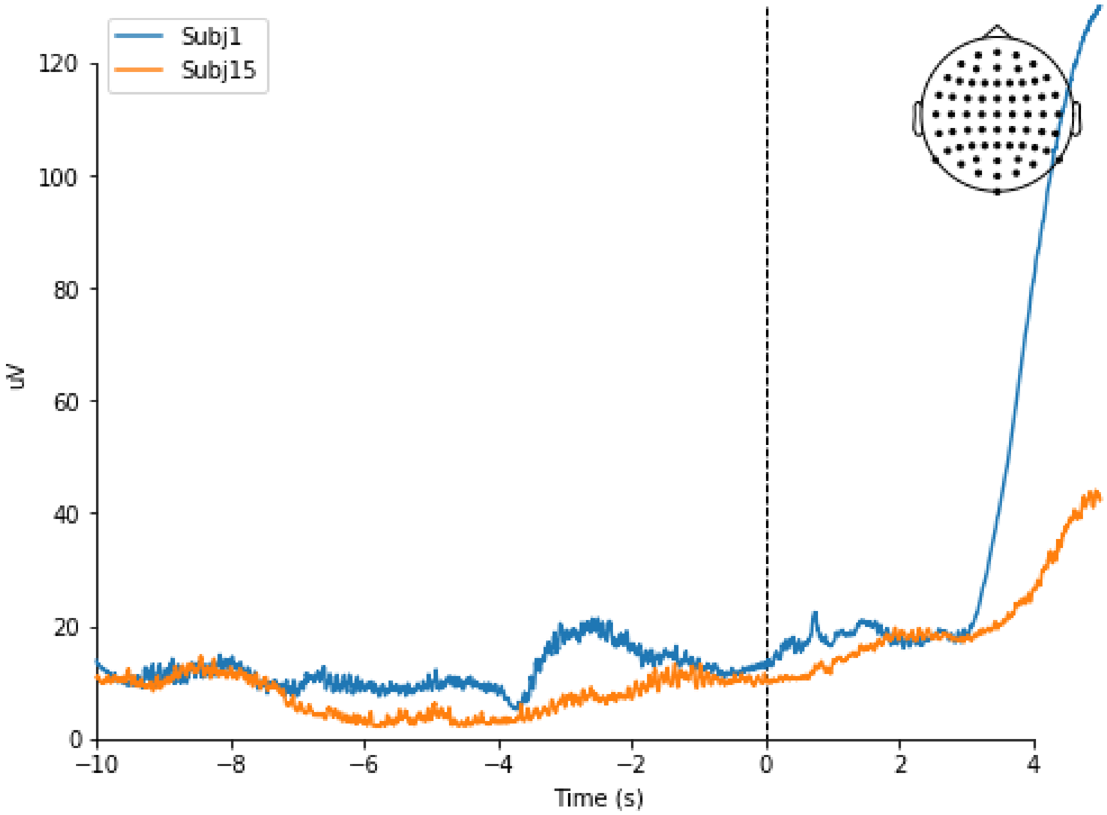
Well, it doesn't tell us much, but does confirm what we had discussed earlier. It seems that there's more variability in activity for the non-expert (first plot) than the expert (second plot) at channel `B17`.

3.3.6 Evoked Response

Since this is an event related response, we can take a look at the evoked response between the 2 subjects around the onset of the interruption. We only looked at 2 individual channels earlier, but we can also look at the aggregated response from all channels for these 2 subjects

[15]:
```
onset_evoked1 = onset_epochs1.average()
onset_evoked15 = onset_epochs15.average()

mne.viz.plot_compare_evokeds(dict(Subj1=onset_evoked1, Subj15=onset_evoked15),
                             legend='upper left', show_sensors='upper right');
```

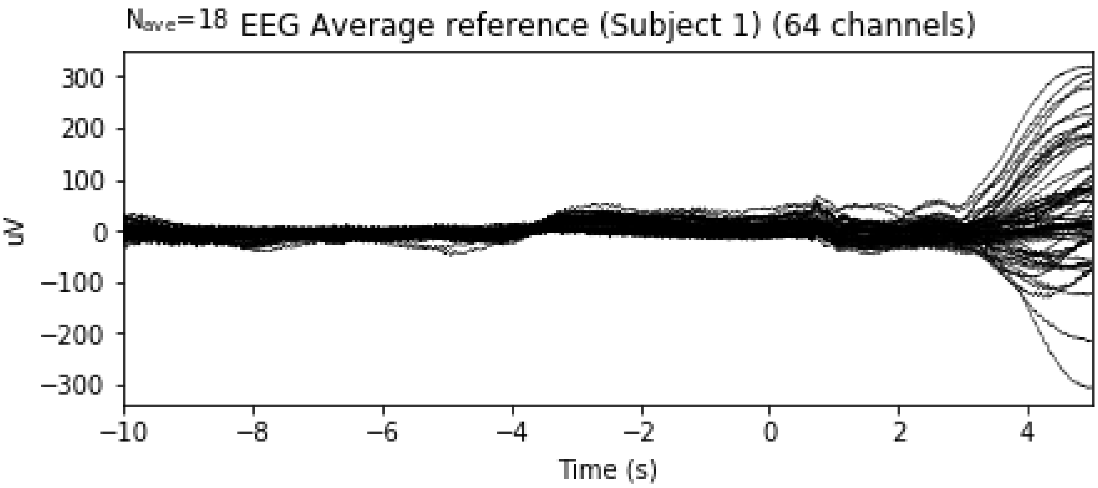
```
combining channels using "gfp"
combining channels using "gfp"
```

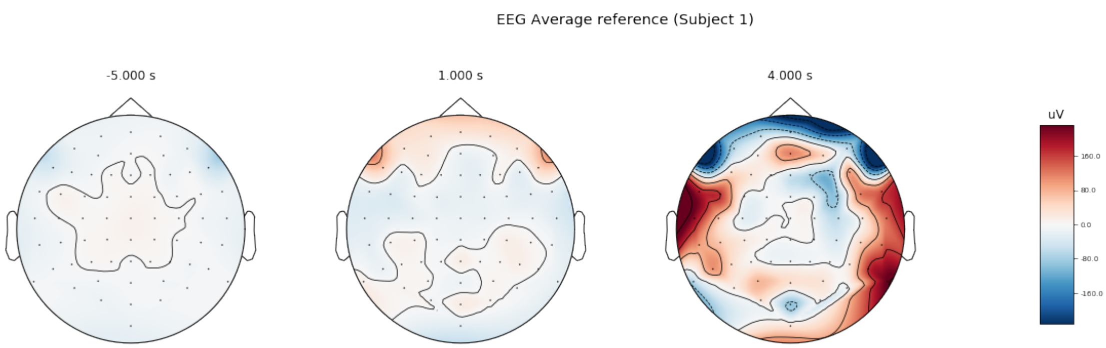


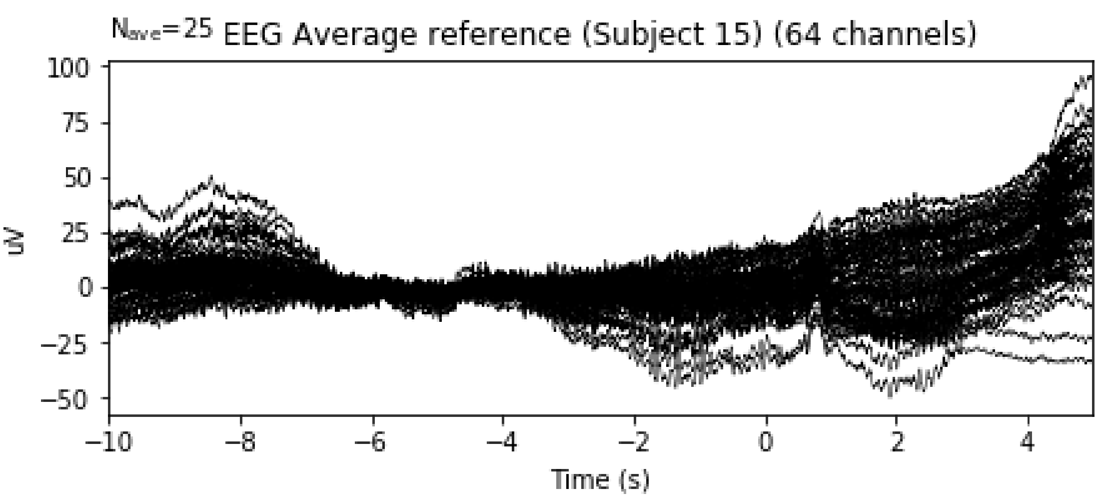
It seems that even from the aggregated results we can already tell which person is an expert and who is non-expert. There is a huge amount of variability for Subject 1 (blue line), while Subject 15's activities seemed to be relatively smooth overall. The spike around 3 seconds after onset (presumably when questions kick in) are drastically different as well.

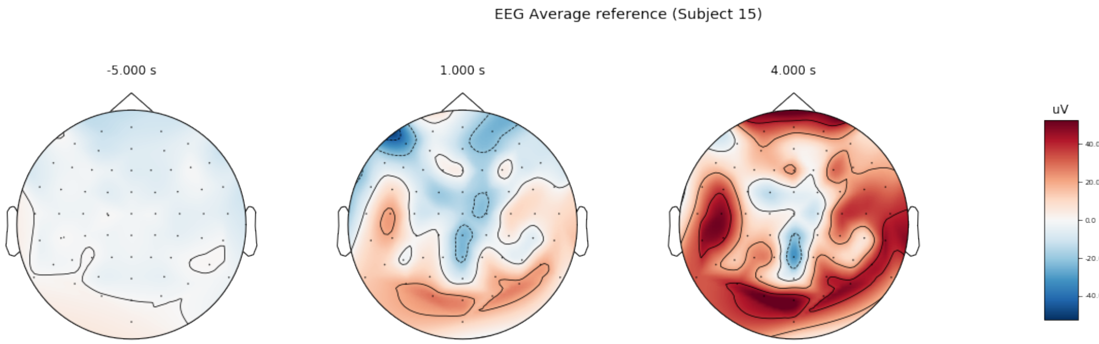
We can also look at which brain region has the most activity around onset epochs.

[16]:
```
title1 = 'EEG Average reference (Subject 1)'
onset_evoked1.plot(titles=dict(eeg=title1), time_unit='s')
onset_evoked1.plot_topomap(times=[-5,1,4], size=3., title=title1,␣
 ↪time_unit='s');

title15 = 'EEG Average reference (Subject 15)'
onset_evoked15.plot(titles=dict(eeg=title15), time_unit='s')
onset_evoked15.plot_topomap(times=[-5,1,4], size=3., title=title15,␣
 ↪time_unit='s');
```


N_ave=18 EEG Average reference (Subject 1) (64 channels)
uV
Time (s)


EEG Average reference (Subject 1)
-5.000 s
1.000 s
4.000 s
uV


N_ave=25 EEG Average reference (Subject 15) (64 channels)
uV
Time (s)



There are very drastic differences for activities between the 2 subjects! There seems to be more frontal lobe activities for subject 1 (non-expert) and more parietal and occipital lobes activities for subject 15 (expert). Subject 1, again, shows a lot more variance compared to subject 15 in averaged EEG references. We also see that for both subjects, there are activations near the temporal lobe by 4 seconds after onset, presumably due to the auditory input from the instructions.

Given all of the results found here around the onset epochs, we are more motivated to see whether this could be trained as a classifier to label the level of experience for a meditator. With this graph, it's quite convincing that it might work.

3.4 Data Cleaning

For simplicity of the final output, data cleaning is performed via a separate notebook `data_cleaning.ipynb` in the same `notebooks` folder. A `*.csv` export of the clean up data will be used and loaded here.

This is going to take a bit to load. Good time for a tea break. Come back in a few minutes.

[17]:
```
cleaned = pd.read_csv('../final_data.csv').rename(columns={'Unnamed: 0':␣
↪'Time'})
```

3.5 Data Analysis

3.5.1 Separating Data: X, Y, Train, Validation, Test

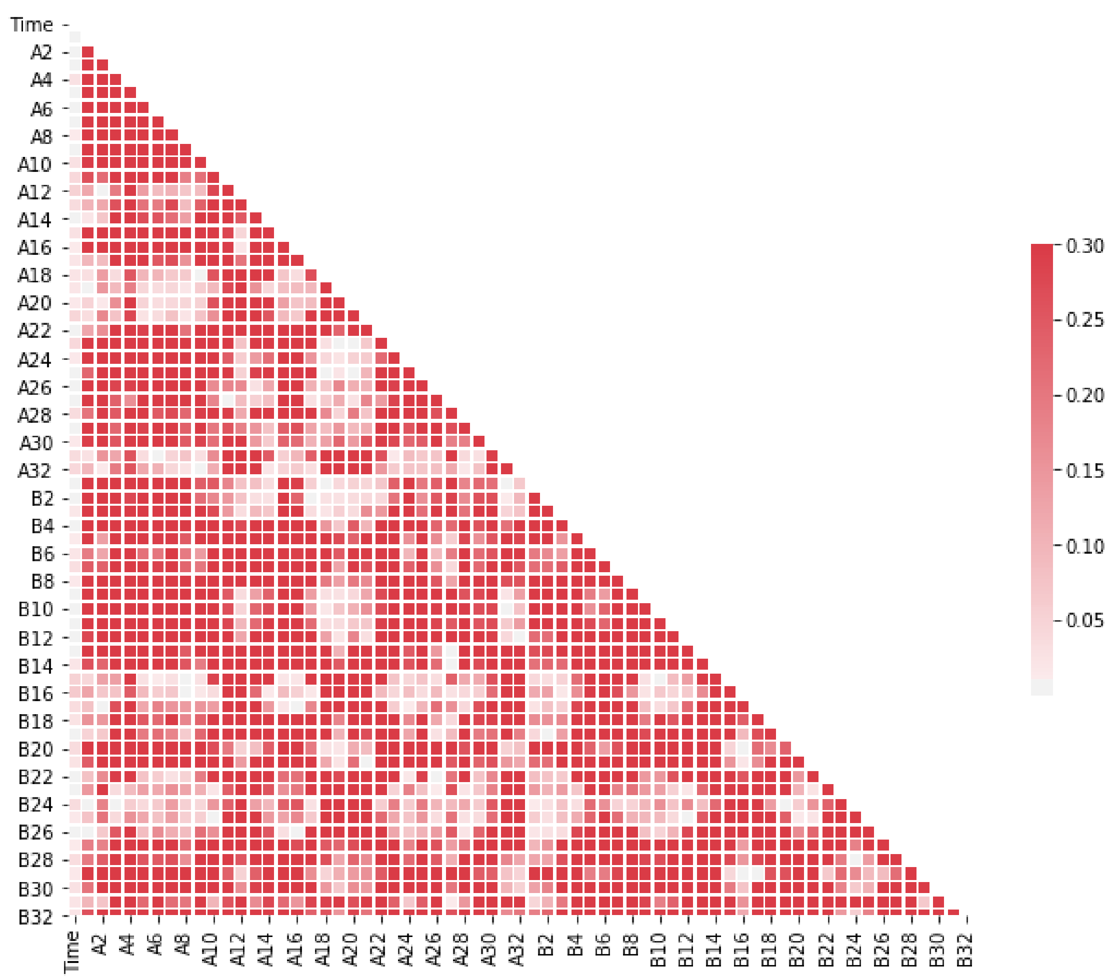
We will be separating the data into 80% as training set, 20% as test set.

[18]:
```
# Setting Random Seed
np.random.seed(45)

# Partitioning X matrix and y vector
X = cleaned.drop('expert', axis=1)
y = cleaned['expert']

# Splitting into Train and Test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

3.5.2 Correlation Matrix

Let's take a look at which channels are most correlated with each other.

[19]:
```python
# Create correlation matrix
corr_matrix = X_train.corr().abs()

# Generate a mask for the upper triangle
mask = np.zeros_like(corr_matrix, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(11, 9))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(220, 10, as_cmap=True)

sns.heatmap(corr_matrix, mask=mask, cmap=cmap, vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5});
```



It seems that a lot of these regions are loosely correlated with each other, but not a lot. There are several channels that are completely not related (lighter areas), say `B17` and `A2`, which were the 2 we discussed earlier, being in completely different regions of the brain (parietal vs. frontal).

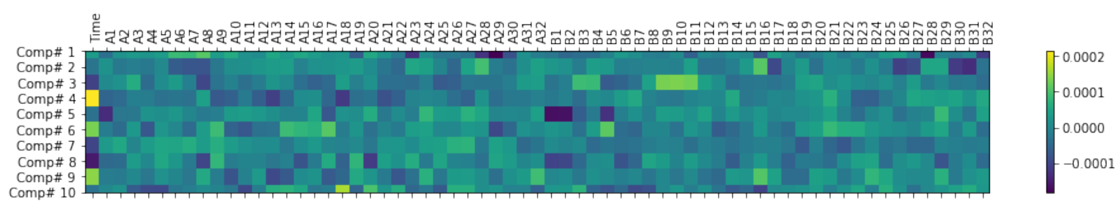
3.5.3 Independent Component Analysis (ICA)

Let's run ICA on our `X_train` and see what features stand out.

```
[20]: sc = StandardScaler()
      X_train = sc.fit_transform(X_train)
      ica = FastICA(n_components=10) # 10 independent components
      X_train_transformed = ica.fit_transform(X_train)
```

This plot will visualize what channels contributed to which ICs the most.

```
[21]: plt.figure(figsize=(16,2))
      plt.matshow(ica.components_, cmap='viridis', fignum=1, aspect='auto')
      labs = []
      plt.yticks(range(ica.components_.shape[0]),
                 [f'Comp# {i}' for i in range(1, ica.components_.
       ↪shape[0]+11)],fontsize=10)
      plt.colorbar()
      plt.xticks(range(ica.components_.shape[1]), np.array(X.
       ↪columns),rotation=90,ha='left')
      plt.rc('xtick', labelsize=10)
      plt.rc('ytick', labelsize=10)
      plt.rc('axes', labelsize=10)
      plt.show();
```



This ICA component matrix tells us that there are several channels of higher importances, even though with a very limited amount of variances explained. It does seem that time has something to do with the components as well.

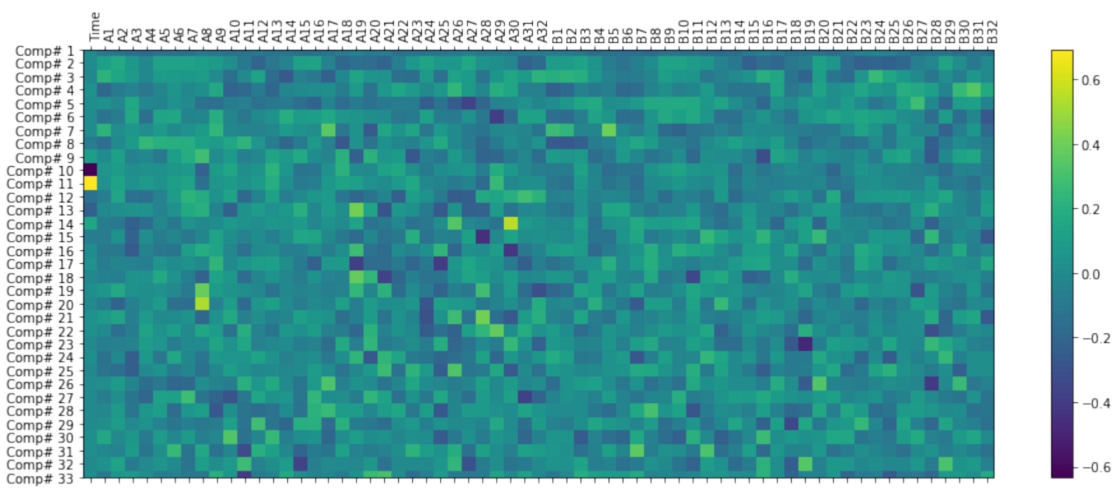
3.5.4 Principal Component Analysis (PCA)

We will try running PCA and see if any matching channels would show up from our results.

[22]:
```python
pca = PCA(n_components=.95) # 95% variance explained
X_train_transformed = pca.fit_transform(X_train)
```

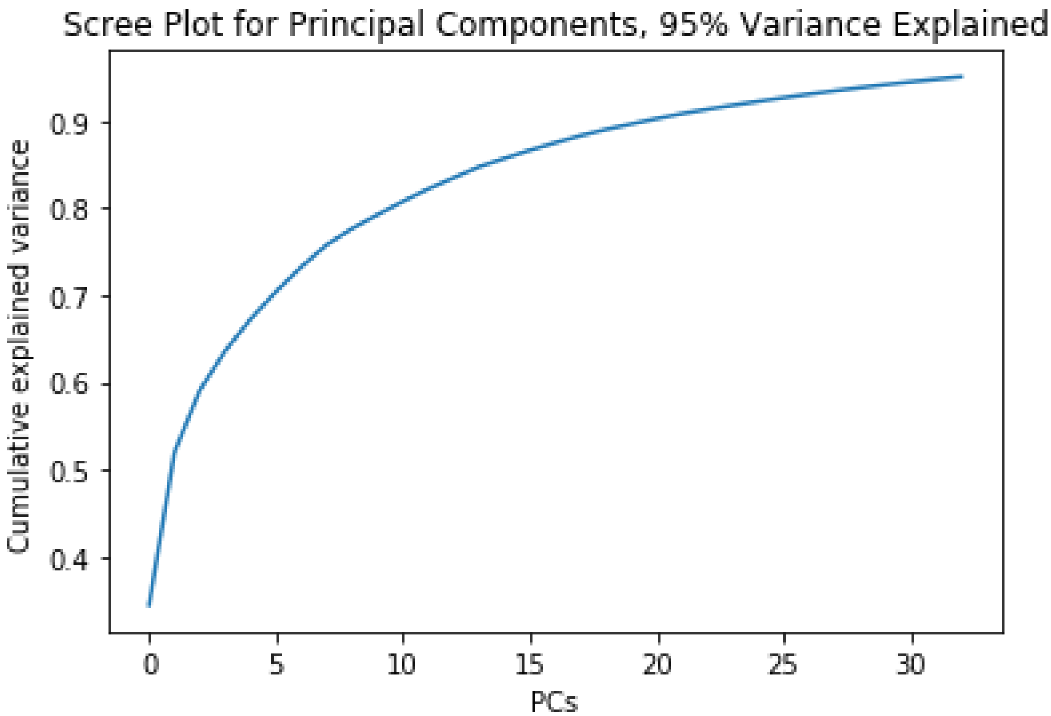
Similarly, this plot will visualize what channels contributed to which PCs the most.

[23]:
```python
plt.figure(figsize=(16,6))
plt.matshow(pca.components_, cmap='viridis', fignum=1, aspect='auto')
labs = []
plt.yticks(range(pca.components_.shape[0]),
           [f'Comp# {i}' for i in range(1, pca.components_.
 ↪shape[0]+1)],fontsize=10)
plt.colorbar()
plt.xticks(range(pca.components_.shape[1]), np.array(X.
 ↪columns),rotation=90,ha='left')
plt.rc('xtick', labelsize=10)
plt.rc('ytick', labelsize=10)
plt.rc('axes', labelsize=10)
plt.show();
```



The PCA yielded a total of 33 principal components and it seems that there are quiet a few channels highly related to some PCs. It seems that time is a huge factor for PC #10 and #11. We can plot the following scree plot to see whether any PCs stand out extremely from everything else.

[24]:
```python
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('PCs')
plt.ylabel('Cumulative explained variance')
plt.title('Scree Plot for Principal Components, 95% Variance Explained')
plt.show()
```



It seems that the first few PCs explain over 50%+ of the variance. However, that is probably not enough for our classifier.

3.6 Classifiers

Please note the the following analyses will take quite some time to run. It's time for a meal break, I'd say! The last 2 methods in total will probably take 45 minutes to an hour to run.

Moving on to building our classifier. Our goal is to see whether we can use these onset epoch data to create a classifier to label expert and non-expert meditator solely based on EEG measures.

3.6.1 Logistic Regression (L-BFGS) with 5-Fold Cross Validation

We can now try to run a simple logistic regression model to predict the labels in the y-vector.

[25]:
```
# Create Logistic Function
logistic = LogisticRegression(solver='lbfgs', multi_class='multinomial',␣
 ↪max_iter=500)

# Cross Validation
cross_val_score(logistic, X_train, y_train, cv=5)
```

[25]: array([0.54178606, 0.54268821, 0.54252523, 0.54246067, 0.54301355])

Our logistic regression using L-BFGS solver yielded a ~54.2% accuracy for our model.

3.6.2 Logistic Regression (LASSO) with 5-Fold Cross Validation

We will introduce a L1 penalty and see if our result improves.

[26]:
```
# Create Logistic Function with L1 Penalty
logistic_l1 = LogisticRegression(penalty='l1', solver='saga',␣
↪multi_class='multinomial', max_iter=500)

# Cross Validation
cross_val_score(logistic_l1, X_train, y_train, cv=5)
```

[26]: array([0.54177944, 0.54269814, 0.54254509, 0.54246067, 0.54300196])

This yielded similar results of a ~54.2% accuracy…. It's bad, but at least it's slightly better than chance (50%).

3.6.3 Neural Network: Multi-Layer Perceptron classifier

We will deploy an MLP classifier to see if our result will improve. This will take about 20 minutes to run, if no other tasks are running on your computer.

[27]:
```
mlp = MLPClassifier(activation='logistic', random_state=45)
mlp.fit(X_train, y_train)
```

[27]:
```
MLPClassifier(activation='logistic', alpha=0.0001, batch_size='auto',
              beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(100,), learning_rate='constant',
              learning_rate_init=0.001, max_iter=200, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=45, shuffle=True, solver='adam', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
```

[28]:
```
mlp.score(X_train, y_train)
```

[28]: 0.9926910952332711

[29]:
```
mlp.score(X_test, y_test)
```

[29]: 0.5000999813279904

It appears that our neural network is really bad at not overfitting and yielded something of 99.27% accuracy on our training set (!!!!!), but only ~50% on our test set… The model performance from the test set actually was worse than our Logistic Regression CV model (depends on the random seed, at least when I set it to 45 it will remain 50% for rerun). This is essentially a chance model that cannot be used.

Let's see if random forest performs a little better.

3.6.4 Random Forest

We will now deploy a random forest classifier on our data to see how well we did. This will take about 25 minutes to run, if no other tasks are running on your computer.

[30]:
```
clf = RandomForestClassifier(n_estimators=20, criterion='entropy',␣
 ↪random_state=45)
clf.fit(X_train, y_train)
```

[30]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='entropy',
                       max_depth=None, max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=20,
                       n_jobs=None, oob_score=False, random_state=45, verbose=0,
                       warm_start=False)
```

[31]:
```
clf.score(X_train, y_train)
```

[31]:
```
0.9998715470829601
```

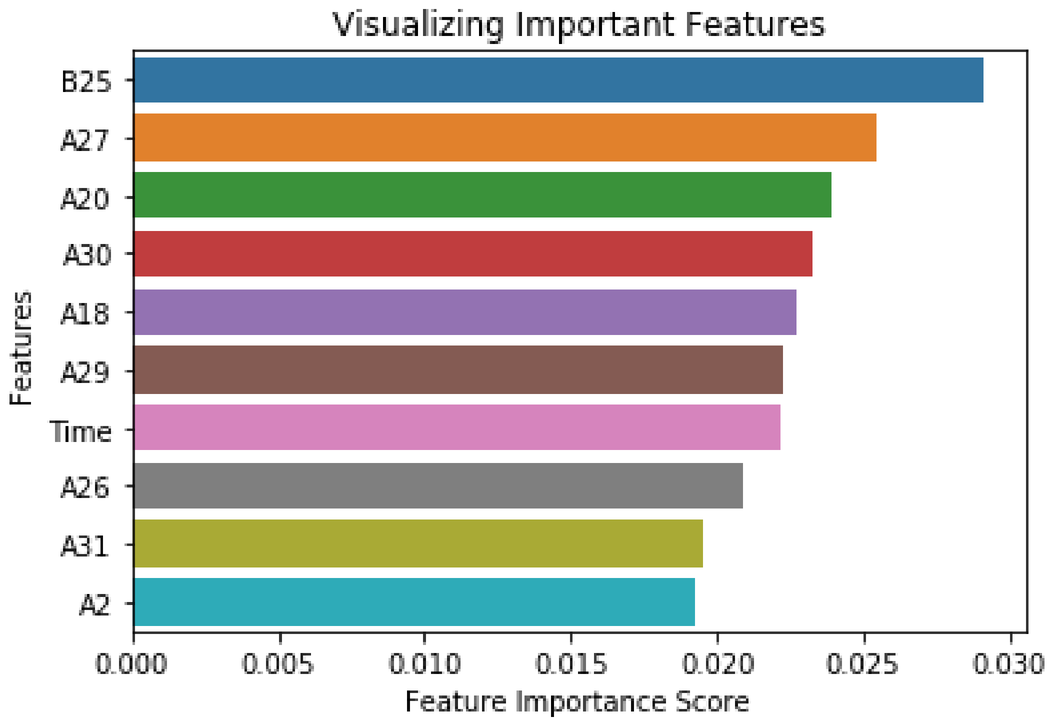
[32]:
```
clf.score(X_test, y_test)
```

[32]:
```
0.5554300455146787
```

Similar to our MLP Classifier, we also way overfit on the training set, reaching a whopping 99.99% accuracy. But we do see a slight increase on our test set accuracy, which is at 55.54%, best in all of our methods.

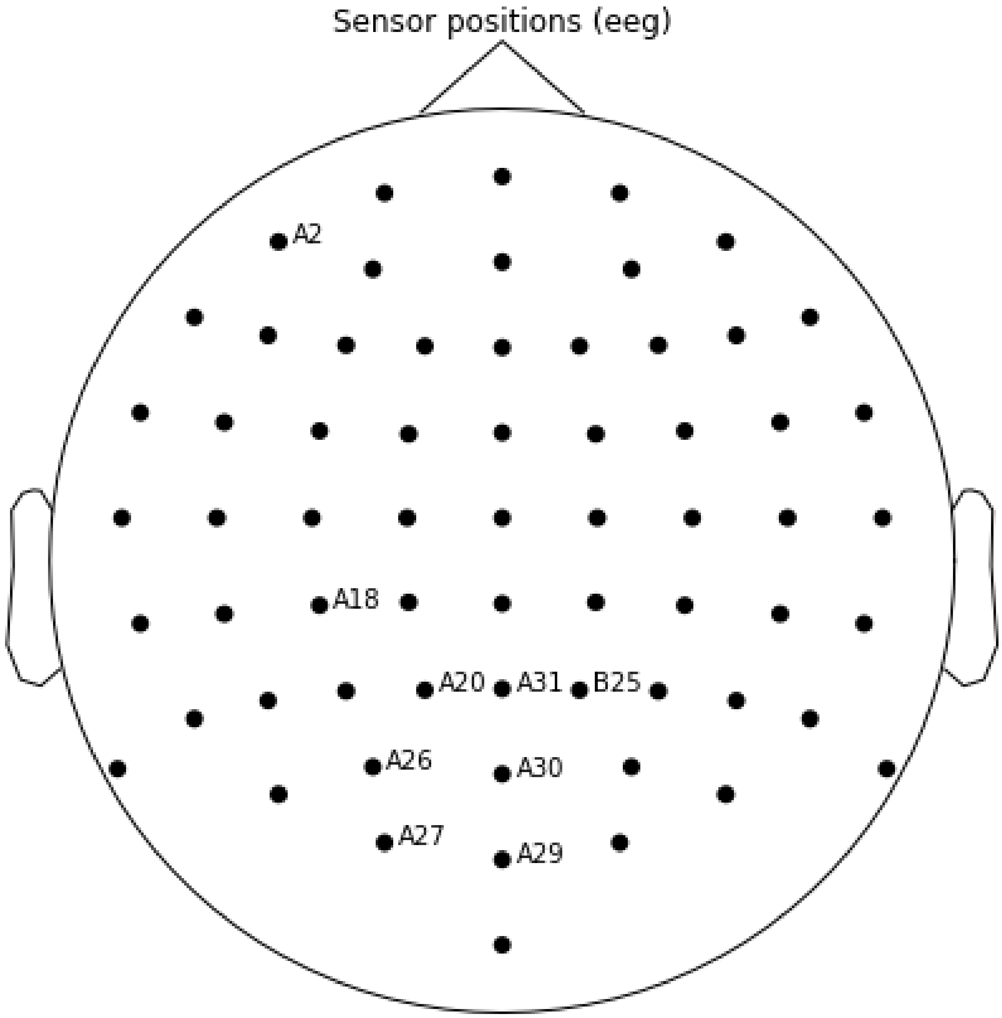
Let's take a moment to see which channels were most important in building our random forest. We will select the top 10 features.

[33]:
```
feature_imp = pd.Series(clf.feature_importances_,
                        index=X.columns).sort_values(ascending=False)[:10]

sns.barplot(x=feature_imp, y=feature_imp.index)
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title("Visualizing Important Features")
plt.show();
```



We can see that the time seem to be some sort of determinant factor in our analysis, but not sure what. We can capture these important features/channels and visualize them on the sensor plot below.

[34]:
```
# Deleting Time as a "channel"
imp_chs = list(np.delete(np.array(feature_imp.index), list(feature_imp.index).
 ↪index('Time')))
mne.viz.plot_sensors(raw1.info, ch_type='eeg', show_names=imp_chs);
```



From this plot we can see that the important factors are mostly located near the parietal and occipital lobes, which had been found to have some correlation with functional connectivity and default mode network in other related research.

4 Discussion

4.1 Conclusion

This project was able to roughly follow how the original research was set up to do, but instead of focusing on evidence of mind wondering (maybe yes given the amount of variance in activities we see in a non-expert subject), this project focused on what regions are more significant attributing the state of meditation, and whether or not those regions could be used to label brain activities during meditation as experts or non-experts.

This project went through a pretty complete data science life cycle by first exploring extensively with the data, then settling on what regions of interests and types of analyses to be performed. The data was cleaned and stripped down to the onset epochs we care about, and taken as differences at 0-second/the onset as reference. The analyses were ranked simpler to more complicated, and all perform around a 52%+/- range in correctly labeling our test data. Our best model, the Random Forest classifier, though overfitted on the training set, was still able to outperform all other models in the test set, returning with model accuracy close to 56% in labeling expert/non-expert from brain activities around the onset opochs alone. Given the limited scope of this project, as well as my limited neuroscience expertise, this is a decent result for a machine learning classification task. (As my data science friends said, this is good enough for a start-up!)

4.2 Limitations & Future Research

There are several limitations to this project. First, this project completely neglects individual differences by removing subject and sessions and aggregating the brain data. It also removed continuous time scale and center each individual epoch around onset, and most likely neglecting the effect of adaptation with such repeated tasks of interruption. Second, little feature selection was performed to improve our model. Some basic steps were taken to attempt to select features, but more could be done to solidify that. Third, as the author in the original paper noted, we can take more of the responses to questions (depth of meditation/concentration and mind wandering awareness) into account, and seeing whether there would be a difference in terms of subject's labeling of their own mental states.

5 References

1. **Berkovich-Ohana, A., Harel, M., Hahamy, A., Arieli, A., Malach, R. (2016).** Data for default network reduced functional connectivity in meditators, negatively correlated with meditation expertise. *Data Brief* 8:910–914.

2. **Brandmeyer, T. & Delorme, A. (2018).** Reduced mind wandering in experienced meditators and associated EEG correlates. *Exp Brain Res* 236: 2519.

3. **Christoff, K., Gordon, A.M., Smallwood, J., Smith, R., Schooler, J.W. (2009).** Experience sampling during fMRI reveals default network and executive system contributions to mind wandering. *Proc Natl Acad Sci* 106(21):8719–8724.

4. **Delorme, A. (2018).** BIDS formatted EEG meditation experiment data (Version 2.0) [Data set]. *Zenodo.* http://doi.org/10.5281/zenodo.2536267.

5. **Garrison, K.A., Zeffiro, T.A., Scheinost, D., Constable, R.T., Brewer, J.A.. (2015).** Meditation leads to reduced default mode network activity beyond an active task. *Cogn Affect Behav Neurosci* 15(3):712–720.

6 Footnotes/Comments

1. Many of these codes are adopted from the `mne` package's examples and tutorials, as well as from the documentation of `scikit-learn` and `seaborn`.

2. See Appendix A (`data_cleaning.ipynb`) in the `notebooks` folder for the data cleaning steps.

3. This project took about 25 hours from beginning to end, concentrated on 3-4 days over the span of a month. Since I had zero prior knowledge of working with brain data in general and EEG specifically, the bulk of the time (~10 hours) was me mindlessly learning how to actually process the EEG data using the `mne` package, and figuring out the custom montage from scratch. Running the more advanced methods took a lot of time in the background. About 2-3 hours were spent on cleaning up the final output notebook and rerunning to ensure everything was intact.

4. All parts of this project (except data due to the large size) are available at my GitHub repository (https://github.com/yuyang-zhong/EEG-Neural). Link to the original data files are provided in the `README.md` document.