An Easy-to-Follow Handbook for EEG Data Analysis based on Python

Zitong Lu[a,*], Wanru Li[b], Lu Nie[c], and Kuangshi Zhao[d]

[a]Department of Psychology, The Ohio State University, Columbus, Ohio, USA
[b]Peking-Tsinghua Center of Life Sciences, Peking University, Beijing, China
[c]Department of Psychology, Sun Yat-Sen University, Guangzhou, China
[d]Neuracle Technology (Changzhou) Co., Ltd, Changzhou, China

Correspondence*: Zitong Lu lu.2637@osu.edu

Abstract

This easy-to-follow handbook offers a straightforward guide to EEG data analysis using Python, aimed at all EEG researchers in cognitive neuroscience and related fields. It covers the journey from single-subject data preprocessing to advanced processing techniques across subjects, with a focus on practical application using Python libraries such as NumPy, MNE-Python, SciPy, NeuroRA, etc. Designed for easy comprehension, our handbook can serve as an essential tool for anyone looking to delve into the field of EEG data analysis with Python.

GitHub website: https://github.com/ZitongLu1996/Python-EEG-Handbook
For Chinese version: https://github.com/ZitongLu1996/Python-EEG-Handbook-CN

Motivation and Overview

Nearly all EEG novices begin their journey into the preprocessing of EEG data with EEGLAB [1], whose user-friendly GUI interface and MATLAB-based script operations have influenced an entire generation of EEG researchers. However, with the rapid development of the straightforward and accessible Python language, a wealth of community resouces has also expanded into the field of cognitive neuroscience . A plethora of related toolkits, such MNE-Python [2], Nilearn [3], Nibabel [4], and NeuroRA [5], have emerged, offering us the opportunity to analyze various neural datasets using Python. Regrettably, these tools have not been widely adopted and remain underutilized. To address this gap, with the dual aims of encouraging more psychology and neuroscience researchers to join the Python community and providing a conduit to more advanced EEG data operations, we have endeavored to create a Python EEG processing tutorial. The effort has culminated in this "Python Handbook for EEG Data Analysis".

"What if I'm not good at programming?" or "What if I find it difficult to learn data processing with Python?" Indeed, code-based data processing can be daunting for many. However, through this easy-to-follow handbook, we aim to assure you that there is no need for trepidation. By taking it step by step, learning and understanding gradually, your programming skill will undoubtedly improve, and you will become proficient in EEG data processing.

Three years ago, we embarked on this project with an attitude of rigor, sincerity, and public service, releasing our initial Chinese version of the Python EEG data processing handbook to the simplified Chinese community during the summer of 2021. Over these years, we have been continuously learning and embracing feedback and suggestions from our readers. This has

allowed us to refine the handbook, culminating in the presentation of this relatively more comprehensive and complete English version before you today.

This handbook is divided into four main chapters: Preprocessing Single-Subject Data, Basic Python Data Operations, Multiple-Subject Analysis, and Advanced EEG Analysis (Figure 1). In the Preprocessing Single-Subject Data chapter, we aim to provide a standardized procedure for preprocessing EEG data of individual subjects primarily using the MNE-Python package. In the Basic Python Data Operations chapter, we introduce common Python martix operations, data reading and storage, as well as the basics of statistical analysis and implementation relevant to EEG data processing. In the Multiple-Subject Analysis chapter, we guide readers through detailed examples on how to read data from multiple subjects, conduct ERP and time-frequency analyses, and visualize the results based on an open dataset of a face perception task [6]. In the Advanced EEG Analysis chapter, we delve into three popular analyses methodologies, Classification-based decoding [7,8], Representational Similarity Analysis (RSA) [9,10], and Inverted Encoding Model (IEM) [11–14], through practical examples based on an open dataset of a visual working memory task [15] mainly using NeuroRA [5] and enhanced inverted-encoding [11] packages.

We sincerely hope our EEG handbook offers valuable insights and suggestions. While we have endeavored to present a very easy-to-follow tutorial, it understandably cannot be directly applied to your own EEG data in its entirely. However, we believe that with some straightforward modificiations, we might find yourself adeptly processing your data in no time. Embracing the principle of "teaching someone to fish rather than giving them a fish", we hope our readers and users can apply the knowledge broadly. The content in this handbook may not delve deeply into every aspect; thus, the nuances and mysteries, techniques and philosphies, are left for you to discover through diligent practice and experience accumulation in your own journey.

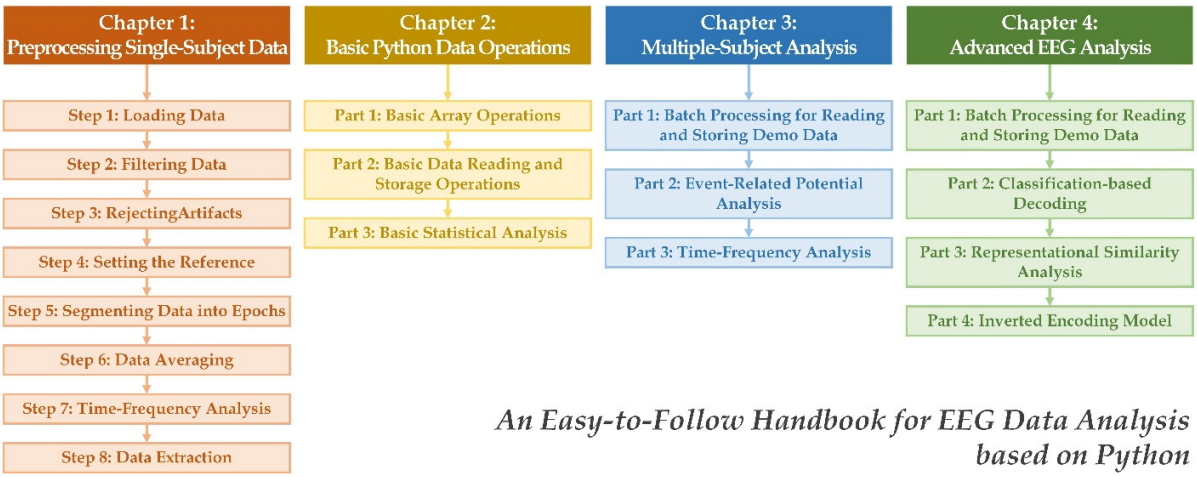


Figure 1 Overview of our handbook.

Main Body of the Handbook

(We saved the content from the Jupyter-Notebooks.)

Chapter 1: Preprocessing Single-subject Data

In this single-subject analysis chapter, EEG pre-processing has been divided into the following 8 steps:

- **Step 1: Loading Data**
- **Step 2: Filtering Data**
- **Step 3: Rejecting Artifacts**
- **Step 4: Setting the Reference**
- **Step 5: Segmenting Data into Epochs**
- **Step 6: Data Averaging**
- **Step 7: Time-Frequency Analysis**
- **Step 8: Data Extraction**

Step 1 Loading Data

Reading Raw Data

Since MATLAB-based EEGLAB is the the most widely used EEG data analysis toolbox that most researchers are more familar with, here we use the classic dataset ('eeglab_data.set') in EEGLAB as an example to teach you how to use Python to deal with EEG data.

In [2]:
```
!pip install mne

import numpy as np
import mne
import os
import gdown
import zipfile
from mne.preprocessing import ICA
from mne.time_frequency import tfr_morlet

import warnings
warnings.filterwarnings('ignore', category=UserWarning, module='matplotlib')

%matplotlib inline

# Download the sample_data

data_dir = "data/"
if not os.path.exists(data_dir):
  os.makedirs(data_dir)

url = "https://drive.google.com/file/d/1bXD-_dDnH5Mv3DQrV7V9fYM4-xYsZ0DN/view?usp=sharing"
filename = "sample_data"
filepath = data_dir + filename + ".zip"

# Download the data
gdown.download(url=url, output=filepath, quiet=False, fuzzy=True)
print("Download completes!")
# unzip the data
with zipfile.ZipFile(filepath, 'r') as zip:
    zip.extractall(data_dir)
print("Unzip completes!")

data_path = data_dir + 'sample_data/eeglab_data.set'

# About data path
# - If dataset has been downloaded in the sample_data folder of the EEGLAB folder,
# navigate to your Downloads directory
# for example:
# data_path = "/Users/zitonglu/Desktop/EEG/eeglab14_1_2b/sample_data/eeglab_data.set"

# Multiple formats of EEG data are supported in MNE-Python:

# *** If the data suffix is '.set '(Data from EEGLAB)
#     use mne.io.read_raw_eeglab()
# *** If the data suffix is '.vhdr' (BrainVision)
```

```
#     use mne.io.read_raw_brainvision()
# *** If the data suffix is '.edf'
#     use mne.io.read_raw_edf()
# *** If the data suffix is '.bdf' (BioSemi)
#     use mne.io.read_raw_bdf()
# *** If the data suffix is '.gdf'
#     use mne.io.read_raw_gdf()
# *** If the data suffix is '.cnt' (Neuroscan)
#     use mne.io.read_raw_cnt()
# *** If the data suffix is '.egi' or '.mff'
#     use mne.io.read_raw_egi()
# *** If the data suffix is '.data'
#     use mne.io.read_raw_nicolet()
# *** If the data suffix is '.nxe' (Nexstim eXimia)
#     use mne.io.read_raw_eximia()
# *** If the data suffix is '.lay' or '.dat' (Persyst)
#     use mne.io.read_raw_persyst()
# *** If the data suffix is '.eeg' (Nihon Kohden)
#     use mne.io.read_raw_nihon()

# Reading data
raw = mne.io.read_raw_eeglab(data_path, preload=True)
```

```
Requirement already satisfied: mne in /Users/zitonglu/anaconda3/lib/python3.11/site-packages (1.6.1)
Requirement already satisfied: numpy>=1.21.2 in /Users/zitonglu/anaconda3/lib/python3.11/site-packag
es (from mne) (1.23.5)
Requirement already satisfied: scipy>=1.7.1 in /Users/zitonglu/anaconda3/lib/python3.11/site-package
s (from mne) (1.11.1)
Requirement already satisfied: matplotlib>=3.5.0 in /Users/zitonglu/anaconda3/lib/python3.11/site-pa
ckages (from mne) (3.6.3)
Requirement already satisfied: tqdm in /Users/zitonglu/anaconda3/lib/python3.11/site-packages (from
mne) (4.65.0)
Requirement already satisfied: pooch>=1.5 in /Users/zitonglu/anaconda3/lib/python3.11/site-packages
(from mne) (1.8.1)
Requirement already satisfied: decorator in /Users/zitonglu/anaconda3/lib/python3.11/site-packages
(from mne) (5.1.1)
Requirement already satisfied: packaging in /Users/zitonglu/anaconda3/lib/python3.11/site-packages
(from mne) (23.1)
Requirement already satisfied: jinja2 in /Users/zitonglu/anaconda3/lib/python3.11/site-packages (fro
m mne) (3.1.2)
Requirement already satisfied: lazy-loader>=0.3 in /Users/zitonglu/anaconda3/lib/python3.11/site-pac
kages (from mne) (0.3)
Requirement already satisfied: contourpy>=1.0.1 in /Users/zitonglu/anaconda3/lib/python3.11/site-pac
kages (from matplotlib>=3.5.0->mne) (1.0.5)
Requirement already satisfied: cycler>=0.10 in /Users/zitonglu/anaconda3/lib/python3.11/site-package
s (from matplotlib>=3.5.0->mne) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in /Users/zitonglu/anaconda3/lib/python3.11/site-pa
ckages (from matplotlib>=3.5.0->mne) (4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /Users/zitonglu/anaconda3/lib/python3.11/site-pa
ckages (from matplotlib>=3.5.0->mne) (1.4.4)
Requirement already satisfied: pillow>=6.2.0 in /Users/zitonglu/anaconda3/lib/python3.11/site-packag
es (from matplotlib>=3.5.0->mne) (9.4.0)
Requirement already satisfied: pyparsing>=2.2.1 in /Users/zitonglu/anaconda3/lib/python3.11/site-pac
kages (from matplotlib>=3.5.0->mne) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in /Users/zitonglu/anaconda3/lib/python3.11/site
-packages (from matplotlib>=3.5.0->mne) (2.8.2)
Requirement already satisfied: platformdirs>=2.5.0 in /Users/zitonglu/anaconda3/lib/python3.11/site-
packages (from pooch>=1.5->mne) (3.10.0)
Requirement already satisfied: requests>=2.19.0 in /Users/zitonglu/anaconda3/lib/python3.11/site-pac
kages (from pooch>=1.5->mne) (2.31.0)
Requirement already satisfied: MarkupSafe>=2.0 in /Users/zitonglu/anaconda3/lib/python3.11/site-pack
ages (from jinja2->mne) (2.1.1)
Requirement already satisfied: six>=1.5 in /Users/zitonglu/anaconda3/lib/python3.11/site-packages (f
rom python-dateutil>=2.7->matplotlib>=3.5.0->mne) (1.16.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /Users/zitonglu/anaconda3/lib/python3.11/
site-packages (from requests>=2.19.0->pooch>=1.5->mne) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in /Users/zitonglu/anaconda3/lib/python3.11/site-package
s (from requests>=2.19.0->pooch>=1.5->mne) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /Users/zitonglu/anaconda3/lib/python3.11/site-p
ackages (from requests>=2.19.0->pooch>=1.5->mne) (1.26.16)
Requirement already satisfied: certifi>=2017.4.17 in /Users/zitonglu/anaconda3/lib/python3.11/site-p
ackages (from requests>=2.19.0->pooch>=1.5->mne) (2023.7.22)
```

```
Downloading...
From: https://drive.google.com/uc?id=1bXD-_dDnH5Mv3DQrV7V9fYM4-xYsZ0DN
To: /Users/zitonglu/Downloads/Python-EEG-Handbook-master/data/sample_data.zip
100%|████████████████████████████████████| 7.30M/7.30M [00:00<00:00, 8.09MB/s]
Download completes!
Unzip completes!
Reading /Users/zitonglu/Downloads/Python-EEG-Handbook-master/data/sample_data/eeglab_data.fdt
Reading 0 ... 30503  =      0.000 ...   238.305 secs...
```

Checking Raw Data Information

In [3]:
```
print(raw)
print(raw.info)
```

```
<RawEEGLAB | eeglab_data.fdt, 32 x 30504 (238.3 s), ~7.5 MB, data loaded>
<Info | 7 non-empty values
 bads: []
 ch_names: EEG 000, EEG 001, EEG 002, EEG 003, EEG 004, EEG 005, EEG 006, ...
 chs: 32 EEG
 custom_ref_applied: False
 highpass: 0.0 Hz
 lowpass: 64.0 Hz
 meas_date: unspecified
 nchan: 32
 projs: []
 sfreq: 128.0 Hz
>
```

We can get some basic and important information of the EEG data: 32 channels with 30504 time points; sampling rate is 128Hz; data length is 238.3s; data size is about 7.5 MB. Here, we set preload=True when we load the data, so the data is preloaded into memory.
The names of channels are 'EEG 000', 'EEG 001', 'EEG 002', 'EEG 003', etc.
0.0Hz High-pass filtering and 64.0 Hz low-pass filtering.

Localizing Channels

Here, the channel names in this dataset are not using standard names. If you face similar situations, you need to manually import the electrode location information of the EEG data.
In this case, the dataset is associated with a .locs file which includes electrode location information. Thus, we need to pass this information into our EEG data.

In [4]:
```
# .locs filepath
locs_info_path = data_dir + "sample_data/eeglab_chan32.locs"
# import channel location information
montage = mne.channels.read_custom_montage(locs_info_path)
# import correct channel names
new_chan_names = np.loadtxt(locs_info_path, dtype=str, usecols=3)
# get old channel names
old_chan_names = raw.info["ch_names"]
# create a dictionary to match old channel names and new (correct) channel names
chan_names_dict = {old_chan_names[i]:new_chan_names[i] for i in range(32)}
# update the channel names in the dataset
raw.rename_channels(chan_names_dict)
#  check location information
raw.set_montage(montage)
```

Out[4]:

▼ **General**

Measurement date	Unknown
Experimenter	Unknown
Participant	Unknown

▼ **Channels**

Digitized points	35 points
Good channels	32 EEG
Bad channels	None
EOG channels	Not available
ECG channels	Not available

▼ **Data**

Sampling frequency	128.00 Hz
Highpass	0.00 Hz
Lowpass	64.00 Hz
Filenames	eeglab_data.fdt
Duration	00:03:59 (HH:MM:SS)

If the EEG data (hat & recording system) you are working with belongs to a specific systems, you might be able to directly use mne.channels.make_standard_montage function to generate channel location information.
Take the standard international 10-20 system as an example, the corresponding code could be changed as follows:

> montage = mne.channels.make_standard_montage("standard_1020")

The available montage of channel location information from other systems in MNE can be found at the following website:
https://mne.tools/stable/auto_tutorials/intro/40_sensor_locations.html#sphx-glr-auto-tutorials-intro-40-sensor-locations-py

Setting Channel Types

In [5]:
```
# The default type of channels in MNE is 'eeg'.
# Here, we need to set the channel type of two EOG channels as 'eog'

chan_types_dict = {"EOG1":"eog", "EOG2":"eog"}
raw.set_channel_types(chan_types_dict)
```

Out[5]:

▼ General

Measurement date	Unknown
Experimenter	Unknown
Participant	Unknown

▼ Channels

Digitized points	35 points
Good channels	30 EEG, 2 EOG
Bad channels	None
EOG channels	EOG1, EOG2
ECG channels	Not available

▼ Data

Sampling frequency	128.00 Hz
Highpass	0.00 Hz
Lowpass	64.00 Hz
Filenames	eeglab_data.fdt
Duration	00:03:59 (HH:MM:SS)

Checking Data Information after Modification

In [6]:
```
# Print information about the modified data

print(raw.info)
```

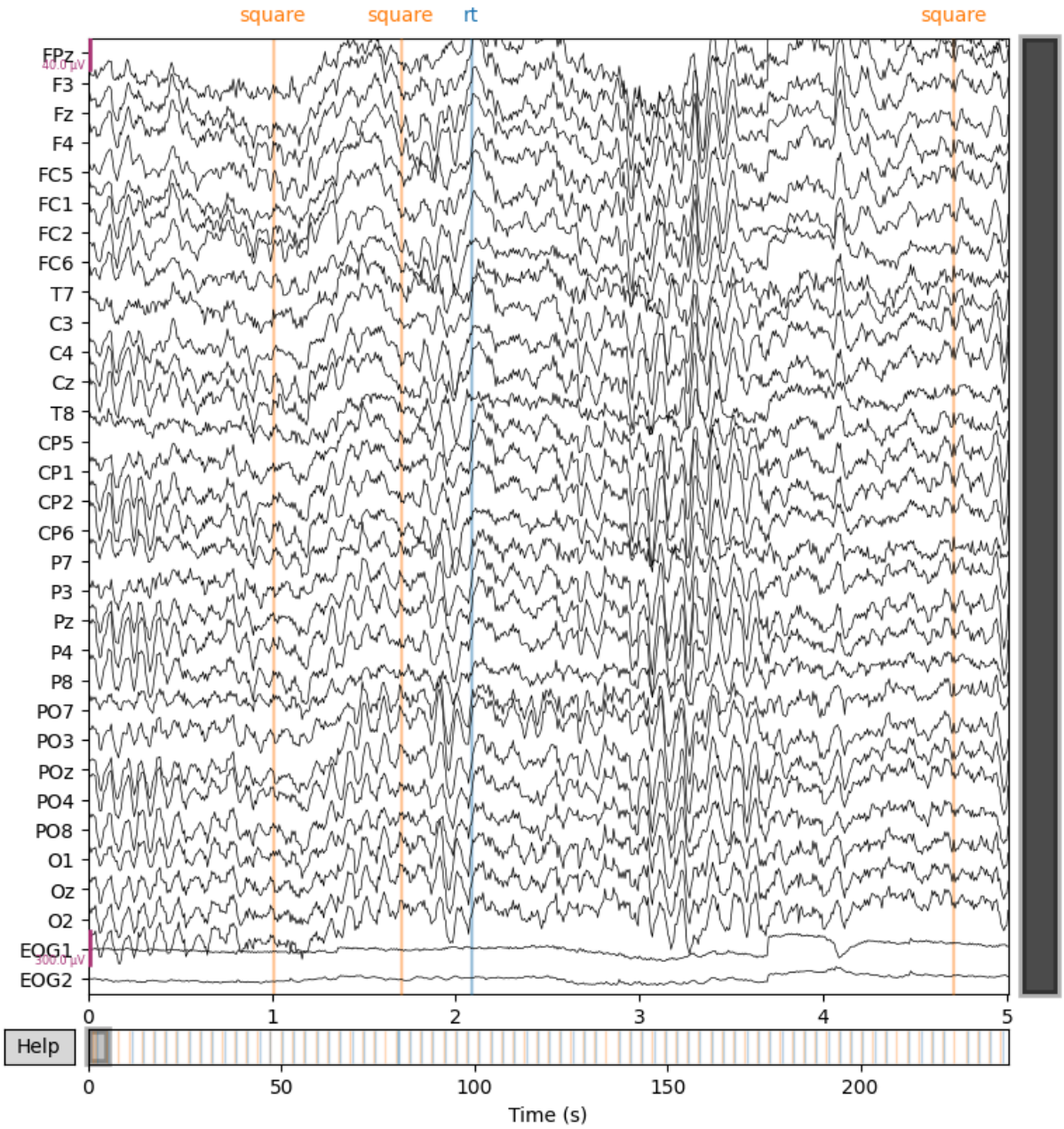
```
<Info | 8 non-empty values
 bads: []
 ch_names: FPz, EOG1, F3, Fz, F4, EOG2, FC5, FC1, FC2, FC6, T7, C3, C4, Cz, ...
 chs: 30 EEG, 2 EOG
 custom_ref_applied: False
 dig: 35 items (3 Cardinal, 32 EEG)
 highpass: 0.0 Hz
 lowpass: 64.0 Hz
 meas_date: unspecified
 nchan: 32
 projs: []
 sfreq: 128.0 Hz
>
```

Visualizing Raw Data

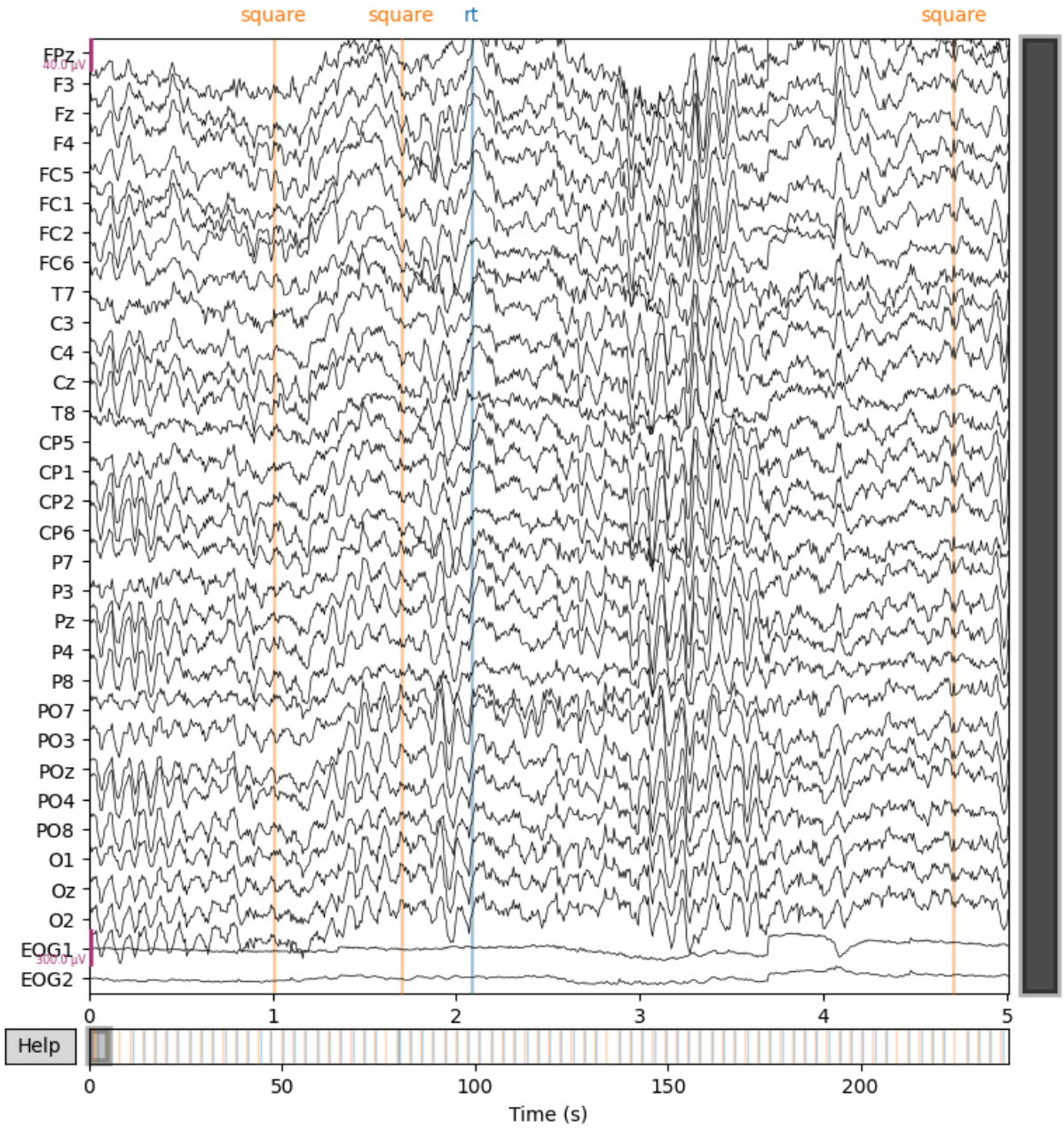
Plot raw data waveforms

In [7]:
```
raw.plot(duration=5, n_channels=32, clipping=None)
```

```
Using matplotlib as 2D backend.
```


square
square
rt
square
FPz
F3
Fz
F4
FC5
FC1
FC2
FC6
T7
C3
C4
Cz
T8
CP5
CP1
CP2
CP6
P7
P3
Pz
P4
P8
PO7
PO3
POz
PO4
PO8
O1
Oz
O2
EOG1
EOG2
40.0 µV
300.0 µV
0
1
2
3
4
5
Help
0
50
100
150
200
Time (s)

Out[7]:

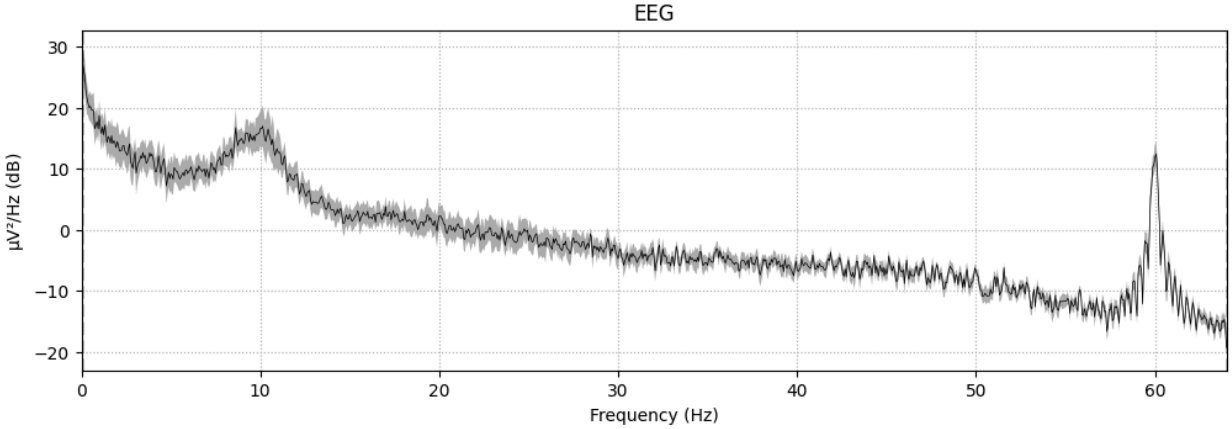


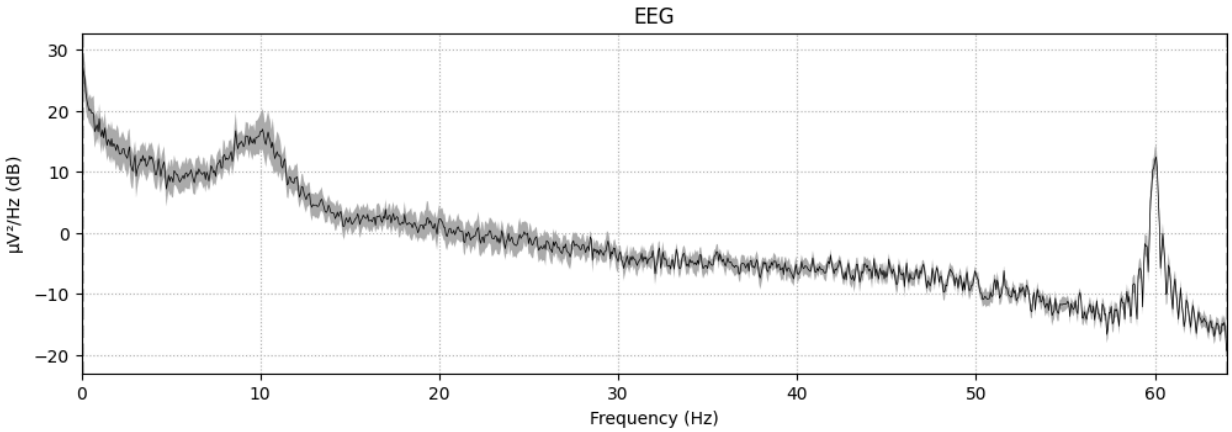
Plot power spectral density across channels

In [8]:
```
raw.plot_psd(average=True)
```

```
NOTE: plot_psd() is a legacy function. New code should use .compute_psd().plot().
Effective window size : 16.000 (s)
/Users/zitonglu/anaconda3/lib/python3.11/site-packages/mne/viz/utils.py:165: UserWarning: Matplotlib
is currently using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so cannot
show the figure.
  (fig or plt).show(**kwargs)
```

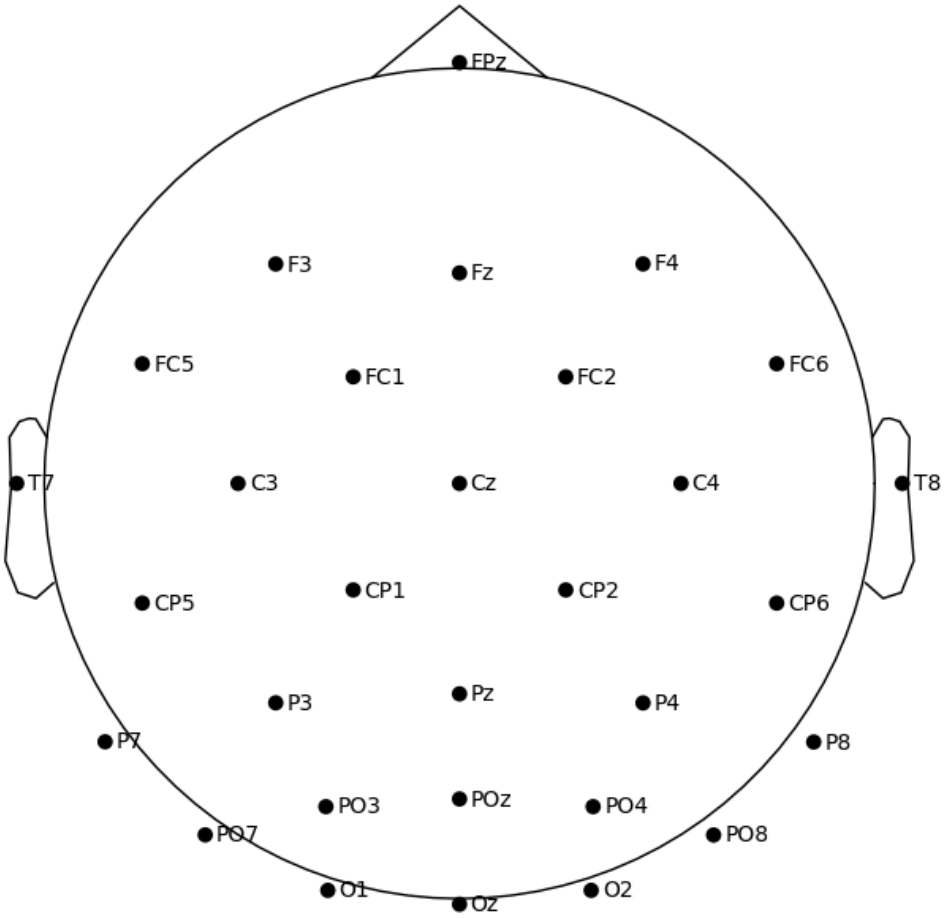
Out[8]:



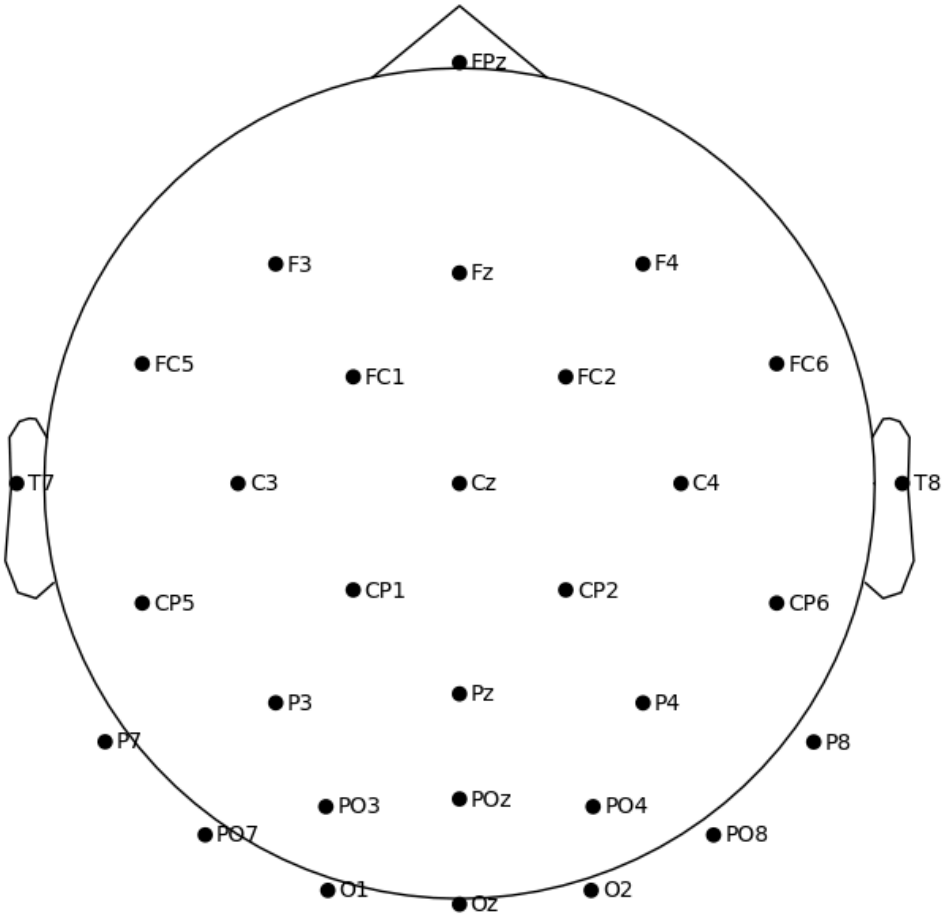


Plot the channel locations map

In [9]:
```
raw.plot_sensors(ch_type='eeg', show_names=True)
```



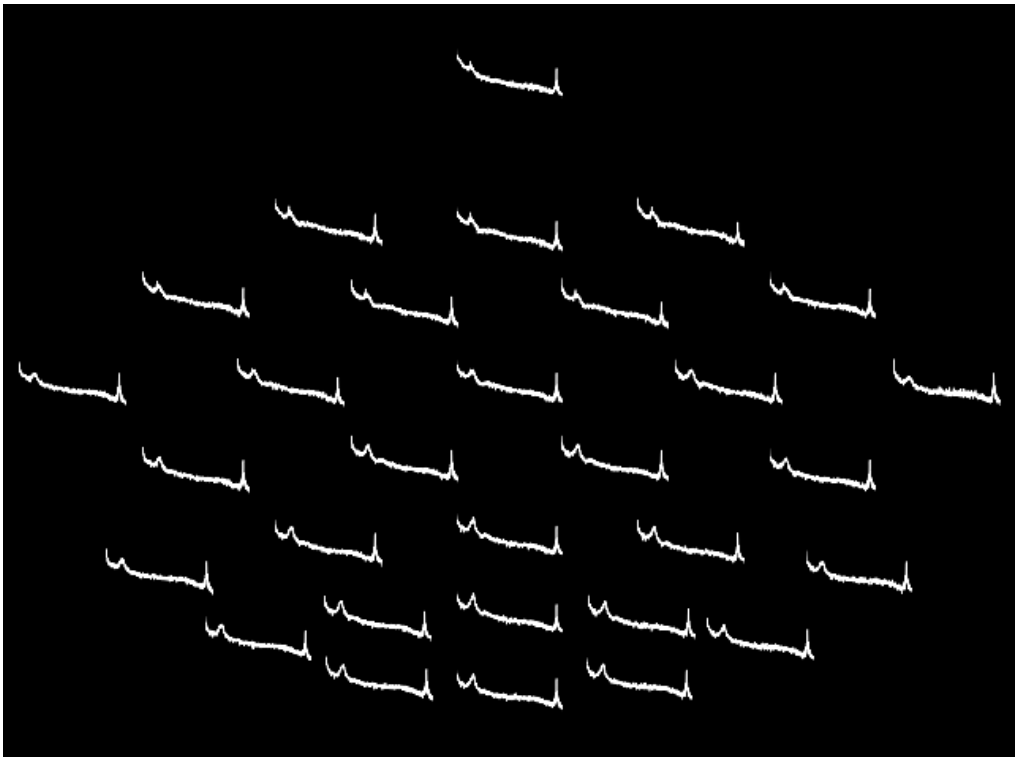
Out[9]:



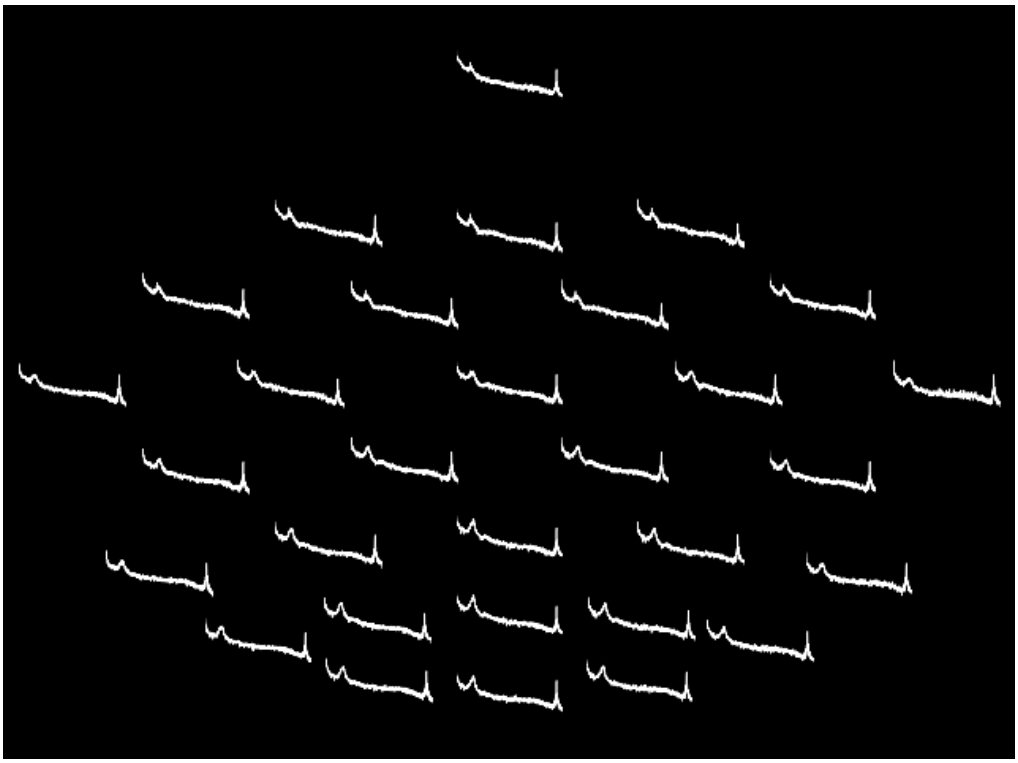
Plot channel-wise frequency spectra as topography

In [10]:
```
raw.compute_psd().plot_topo()
```

```
Effective window size : 16.000 (s)
```



Out[10]:



Step 2 Filtering Data

Notch Filtering

Through the power spectrum in Step 1, we can see that there might be ambient noise at around 60Hz.
Here first use the trap filter to remove utility frequency. Different countries and areas may have different utility frequencies. Remember to judge it by the power spectrum.

In [11]:
```
raw = raw.notch_filter(freqs=(60))
```

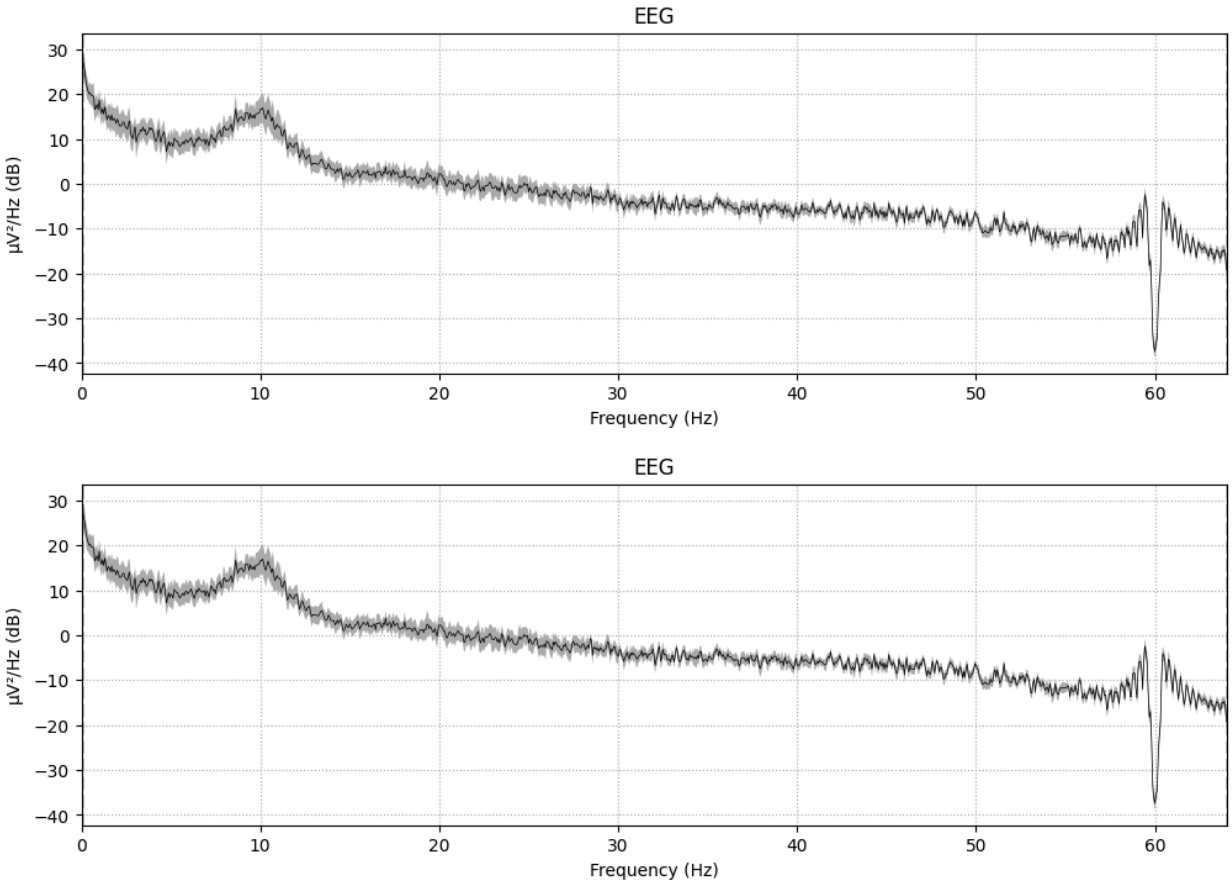
```
Filtering raw data in 1 contiguous segment
Setting up band-stop filter from 59 - 61 Hz

FIR filter parameters
---------------------
Designing a one-pass, zero-phase, non-causal bandstop filter:
- Windowed time-domain design (firwin) method
- Hamming window with 0.0194 passband ripple and 53 dB stopband attenuation
- Lower passband edge: 59.35
- Lower transition bandwidth: 0.50 Hz (-6 dB cutoff frequency: 59.10 Hz)
- Upper passband edge: 60.65 Hz
- Upper transition bandwidth: 0.50 Hz (-6 dB cutoff frequency: 60.90 Hz)
- Filter length: 845 samples (6.602 s)

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  30 out of  30 | elapsed:    0.1s finished
```

Plot the power spectrum

In [12]:
```
raw.plot_psd(average=True)
```

```
NOTE: plot_psd() is a legacy function. New code should use .compute_psd().plot().
Effective window size : 16.000 (s)
/Users/zitonglu/anaconda3/lib/python3.11/site-packages/mne/viz/utils.py:165: UserWarning: Matplotlib is currently using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so cannot show the figure.
  (fig or plt).show(**kwargs)
```

Out[12]:



High/Low-Pass Filtering

In pre-processing, the high-pass filtering is usually a necessary step.

The most common filtering operation is to use a low-pass filter at 30 Hz and a high-pass filter at 0.1 Hz.

High-pass filtering is used to eliminate voltage drift, and low-pass filtering is used to eliminate high frequency noises.

In [13]:
```
raw = raw.filter(l_freq=0.1, h_freq=30)
```

```
Filtering raw data in 1 contiguous segment
Setting up band-pass filter from 0.1 - 30 Hz

FIR filter parameters
---------------------
Designing a one-pass, zero-phase, non-causal bandpass filter:
- Windowed time-domain design (firwin) method
- Hamming window with 0.0194 passband ripple and 53 dB stopband attenuation
- Lower passband edge: 0.10
- Lower transition bandwidth: 0.10 Hz (-6 dB cutoff frequency: 0.05 Hz)
- Upper passband edge: 30.00 Hz
- Upper transition bandwidth: 7.50 Hz (-6 dB cutoff frequency: 33.75 Hz)
- Filter length: 4225 samples (33.008 s)

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  30 out of  30 | elapsed:    0.3s finished
```

In MNE, the default filtering method is FIR. If you want to use IIR filtering method, you can just modify the parameter method:

method='fir' by default, using IIR is to set method='iir'.

The corresponding code is below:

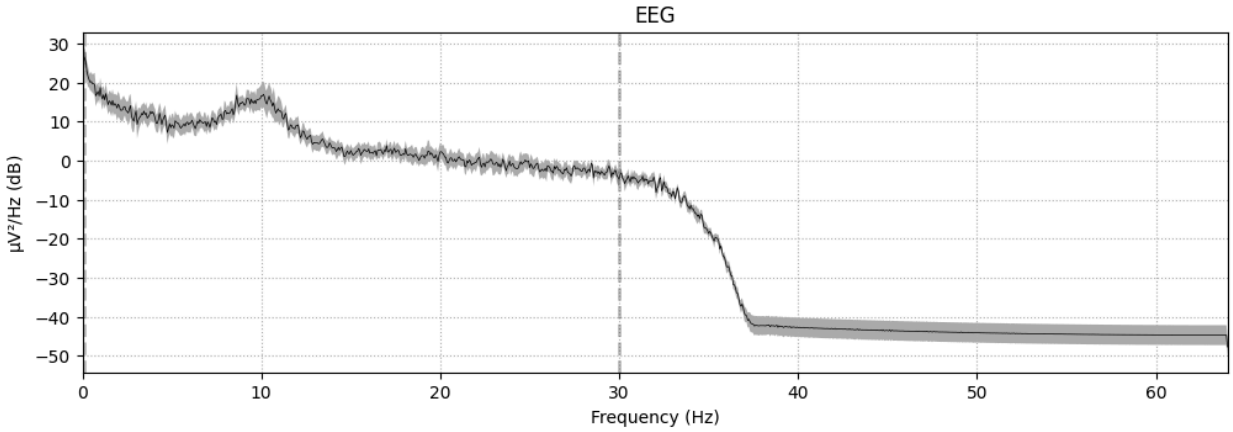
> raw = raw.filter(l_freq=0.1, h_freq=30, method='iir')

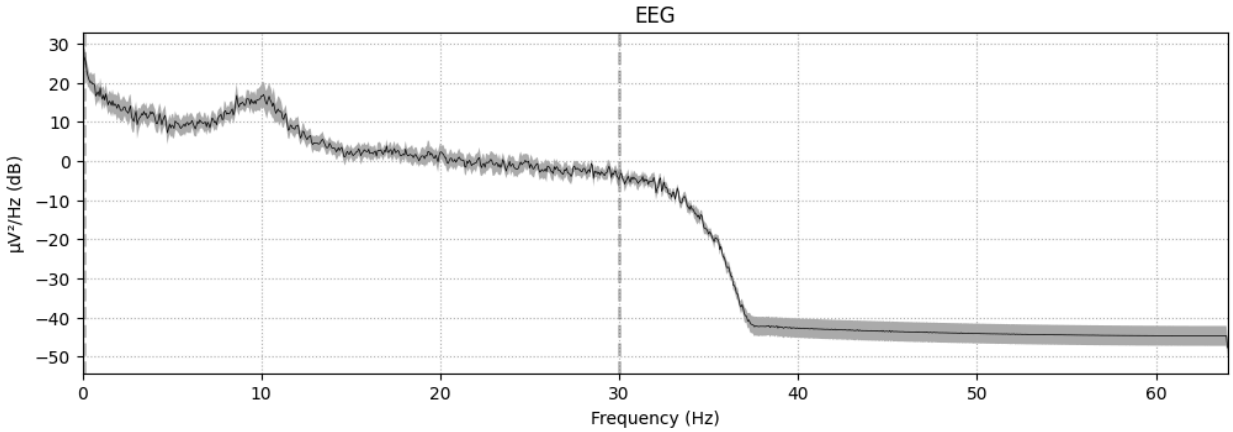
Plot the power spectrum

In [14]:
```
raw.plot_psd(average=True)
```

```
NOTE: plot_psd() is a legacy function. New code should use .compute_psd().plot().
Effective window size : 16.000 (s)
/Users/zitonglu/anaconda3/lib/python3.11/site-packages/mne/viz/utils.py:165: UserWarning: Matplotlib
is currently using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so cannot
show the figure.
  (fig or plt).show(**kwargs)
```

Out[14]:





Step 3 Rejecting Artifacts

Remove Bad Segments

Bad segment marking can be done manually via MNE-Python GUI

```
fig = raw.plot()
fig.fake_keypress('a')
```

Press 'a' to open this small GUI window, then press 'add new label' to add a marker for marking bad segments
In MNE, the bad segments will be not deleted directly. But the data will be marked with bad markers.
In the subsequent data processing, we can set the parameter 'reject_by_annotation' as True in functions to automatically exclude the marked segments while doing data processing. If you encounter the problem that GUI window isn't pop up, please add the following code on the top of the script.

```
import matplotlib
matplotlib.use('TkAgg')
```

Note: Not recommend to open GUI in Jupyter notebook (GUI always crashes).

Remove Bad Channels

The bad channels in MNE will not be deleted directly. MNE just marks the bad channels with 'bads' labels. In the example below, we assume that channel 'FC5' is bad. Thus, we can mark 'FC5' as 'bads':

In [15]:
```
# Marking the bad channel
raw.info['bads'].append('FC5')
```

```
# Printing the bad channel's name
print(raw.info['bads'])
```

```
['FC5']
```

Of course, we can add multiple bad channels

For example, if both 'FC5' and 'C3' are bad channels, we can mark them via the following line:

> raw.info['bads'].extend(['FC5', 'C3'])

Reconstruct Bad Channels

The bad channel reconstruction of MNE is to reconstruct signal of the channels marked as 'bads'

In [16]:
```
raw = raw.interpolate_bads()
```

```
Setting channel interpolation method to {'eeg': 'spline'}.
Interpolating bad channels.
    Automatic origin fit: head of radius 95.0 mm
Computing interpolation matrix from 29 sensor positions
Interpolating 1 sensors
```

The 'bads' marker will be removed by default after signal reconstruction.

If you do not want to remove the markings of the original bad channels, set the reset_bads parameter as False.

The corresponding code is as follow:

> raw = raw.interpolate_bads(reset_bads=False)

Independent Components Analysis (ICA)

Run ICA

The programming idea of ICA in MNE is to first build an ICA object (which can be understood as building an ICA analyzer) then use this ICA analyzer to analyze the EEG data (through methods of ICA object).

Since ICA is not effective for low frequency data, here ICA and artifact components removal are based on high-pass 1Hz data and then applied to high-pass 0.1Hz data.

In [17]:
```
ica = ICA(max_iter='auto')
raw_for_ica = raw.copy().filter(l_freq=1, h_freq=None)
ica.fit(raw_for_ica)
```

```
Filtering raw data in 1 contiguous segment
Setting up high-pass filter at 1 Hz

FIR filter parameters
---------------------
Designing a one-pass, zero-phase, non-causal highpass filter:
- Windowed time-domain design (firwin) method
- Hamming window with 0.0194 passband ripple and 53 dB stopband attenuation
- Lower passband edge: 1.00
- Lower transition bandwidth: 1.00 Hz (-6 dB cutoff frequency: 0.50 Hz)
- Filter length: 423 samples (3.305 s)

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.1s remaining:    0.0s
Fitting ICA to data using 30 channels (please be patient, this may take a while)
[Parallel(n_jobs=1)]: Done  30 out of  30 | elapsed:    0.5s finished
Selecting by non-zero PCA components: 29 components
Fitting ICA took 4.6s.
```

Out[17]:

Method	fastica
Fit parameters	algorithm=parallel fun=logcosh fun_args=None max_iter=1000
Fit	117 iterations on raw data (30504 samples)
ICA components	29
Available PCA components	30
Channel types	eeg
ICA components marked for exclusion	—

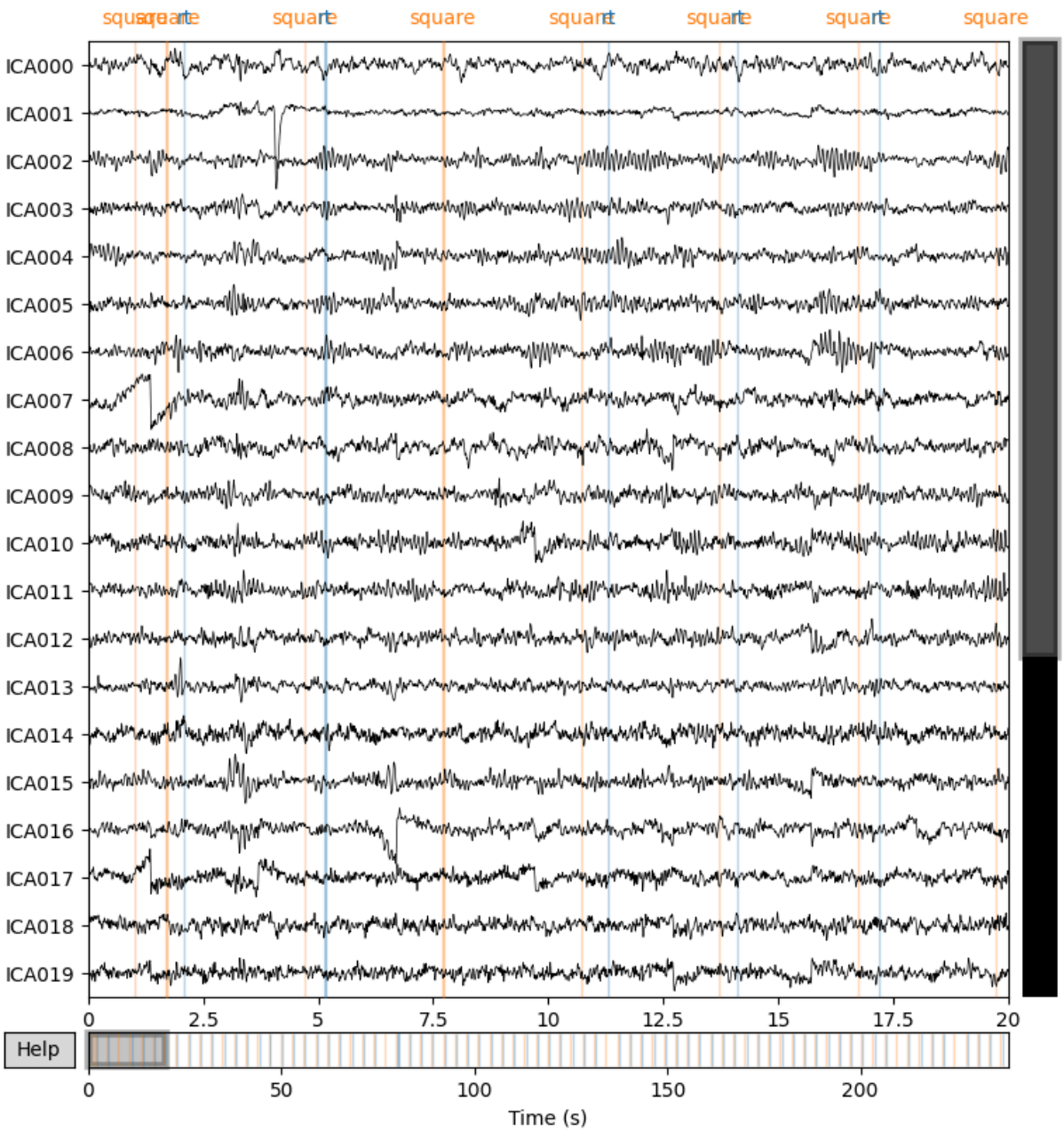
We set n_components here, i.e. the number of ICA components which is automatically selected by MNE's ICA analyzer.
Similar to EEGLAB, if you want the number of ICA components to be fixed, you can customize the setting (n_components<=n_channels).
Taking 30 independent components as an example, the corresponding code can be changed as follows：

> ica = ICA(n_components=30, max_iter='auto')

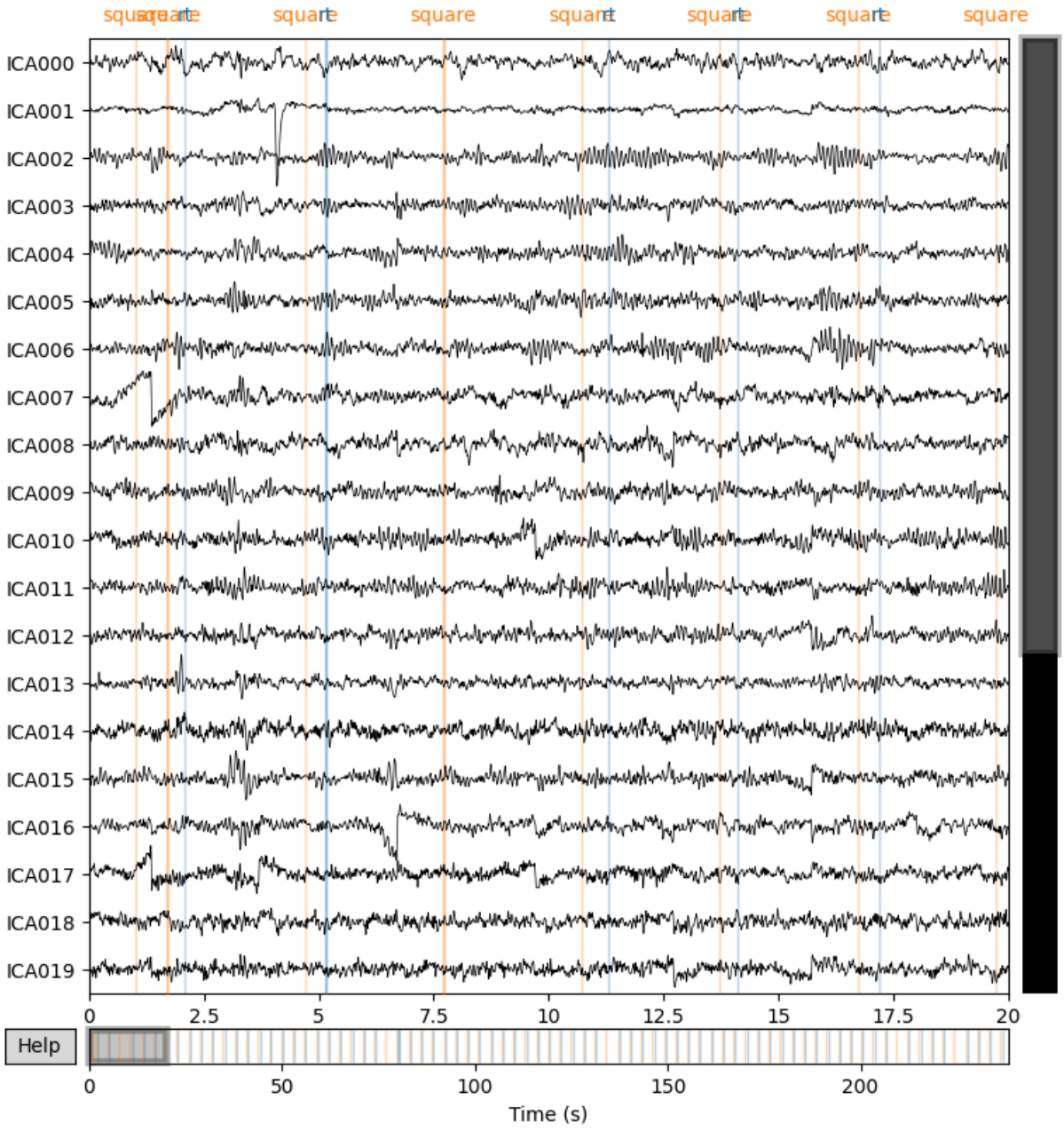
Plot the timing signal of each component

In [18]:
```
ica.plot_sources(raw_for_ica)
```

```
Creating RawArray with float64 data, n_channels=31, n_times=30504
    Range : 0 ... 30503 =      0.000 ...   238.305 secs
Ready.
```

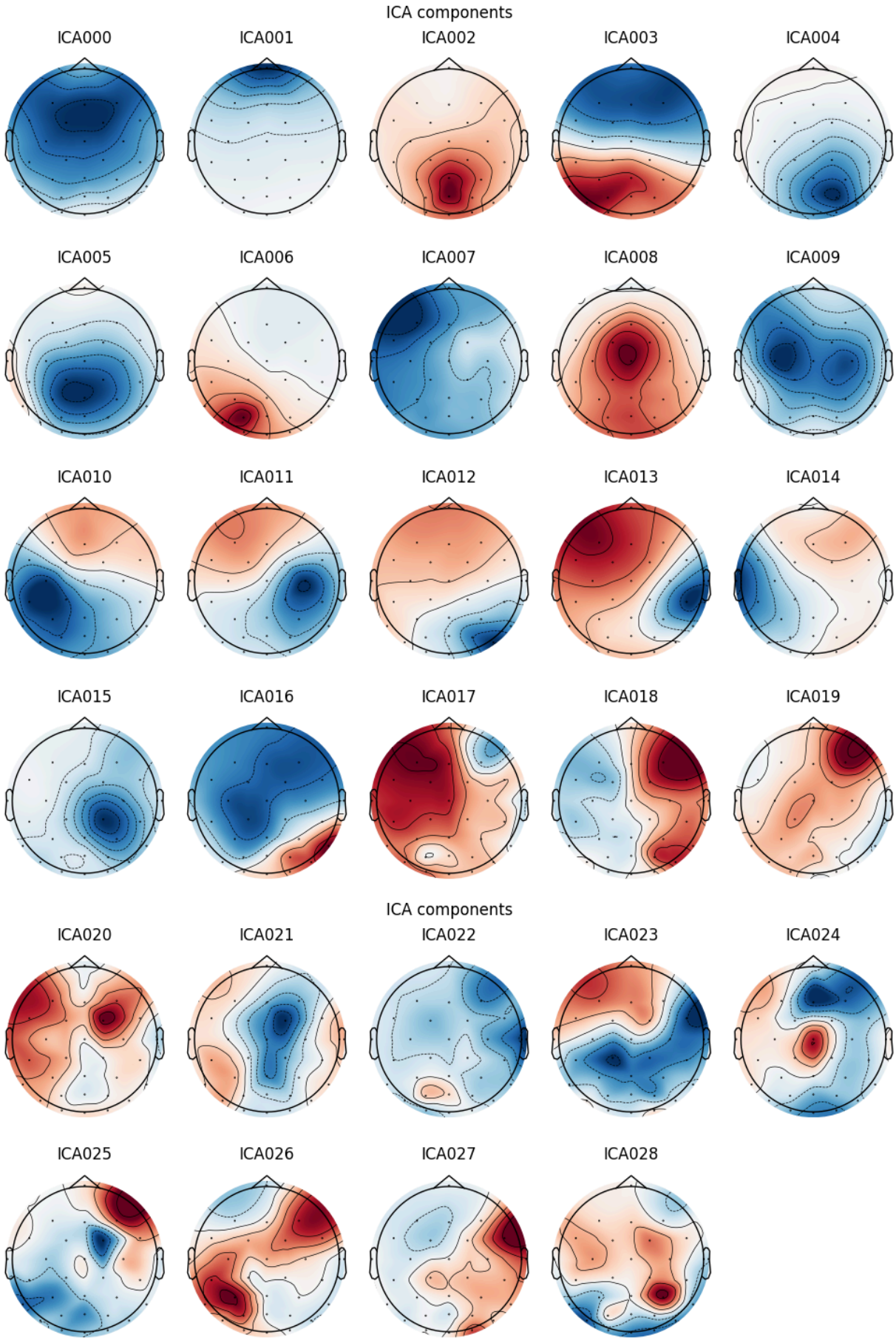

square
ICA000
ICA001
ICA002
ICA003
ICA004
ICA005
ICA006
ICA007
ICA008
ICA009
ICA010
ICA011
ICA012
ICA013
ICA014
ICA015
ICA016
ICA017
ICA018
ICA019
0
2.5
5
7.5
10
12.5
15
17.5
20
Help
0
50
100
150
200
Time (s)

Out[18]:



Plot the topography of each component

In [19]:
```
ica.plot_components()
```



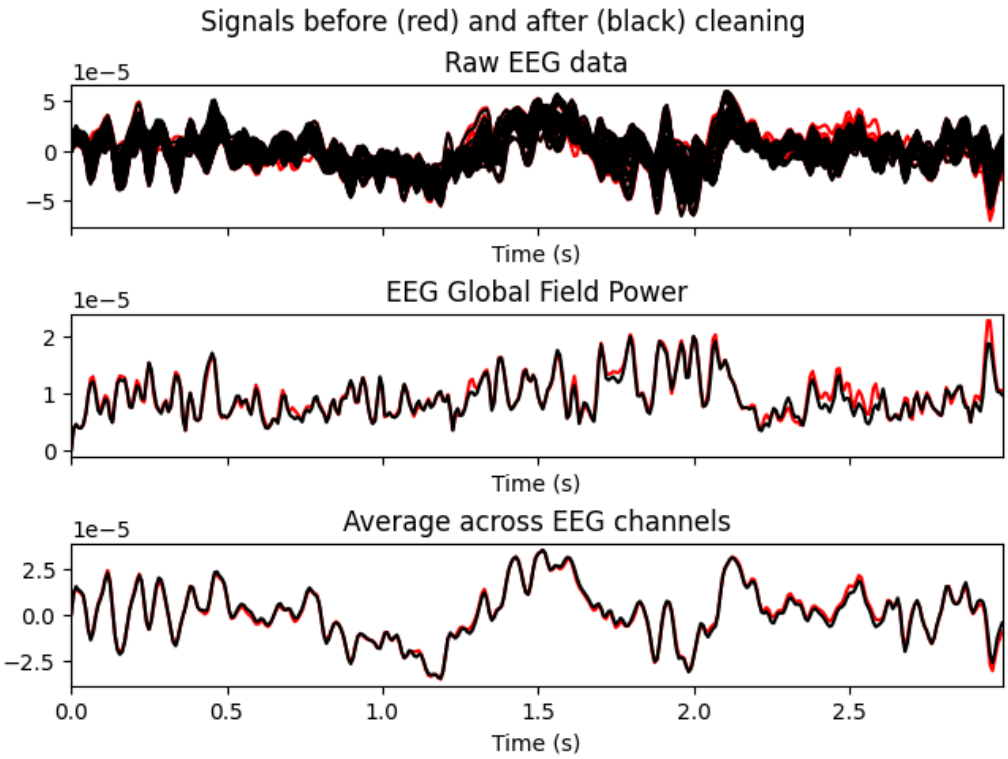
Out[19]: [<MNEFigure size 975x967 with 20 Axes>, <MNEFigure size 975x496 with 9 Axes>]

Check the signal difference before and after the removal of a component / several component.

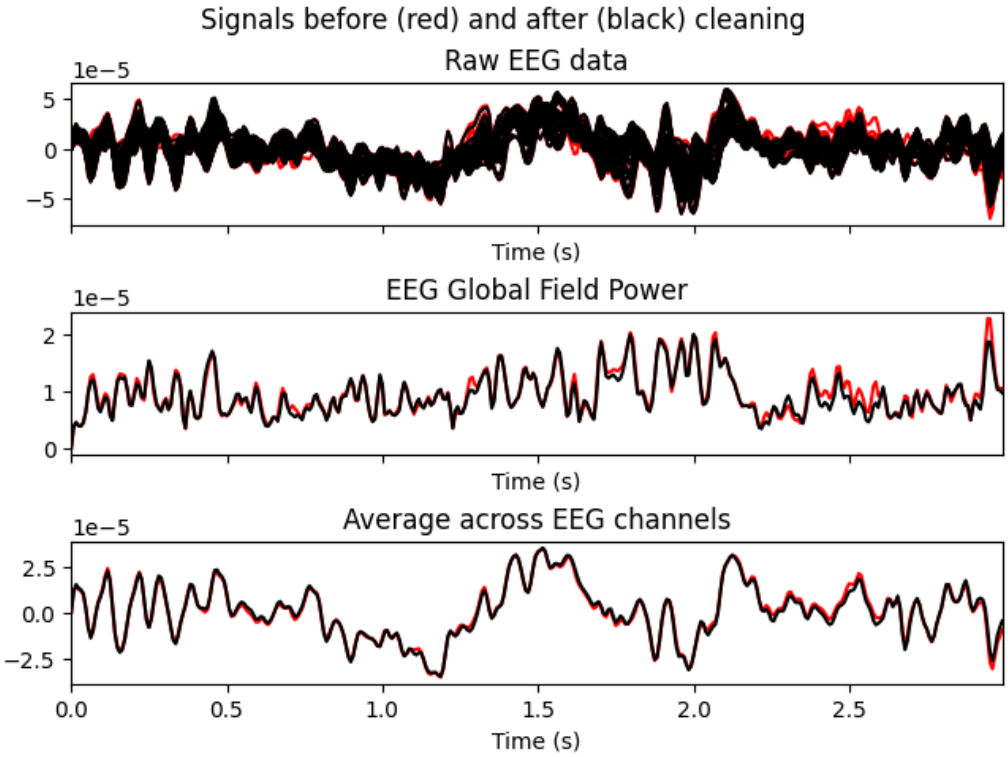
Here is an example of removing the 2rd component (i.e. ICA001)

In [20]:
```
ica.plot_overlay(raw_for_ica, exclude=[1])
```

```
Applying ICA to Raw instance
    Transforming to ICA space (29 components)
    Zeroing out 1 ICA component
    Projecting back using 30 PCA components
```



Out[20]:

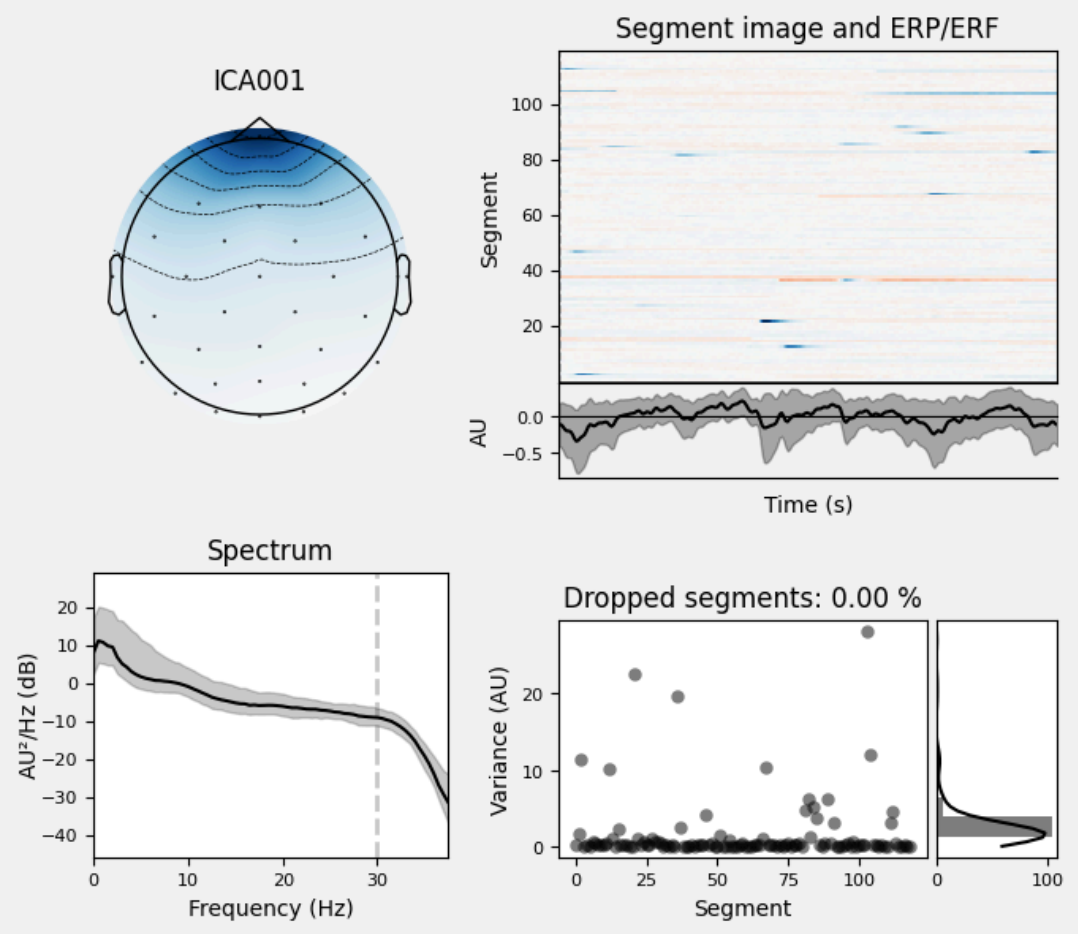


Visualize each component

In the example, we visualize the 2nd component (ICA001).

In [21]:
```
ica.plot_properties(raw, picks=[1])
```

```
    Using multitaper spectrum estimation with 7 DPSS windows
Not setting metadata
119 matching events found
No baseline correction applied
0 projection items activated
```



Out[21]: [<Figure size 700x600 with 6 Axes>]

The power of component ICA001 is higher in the frontal area and is higher at low frequencies, with significant enhancement in some trials.
It can be judged as an EOG component.

Exclude components

In [22]:
```
# set the component index(es) to exclude
ica.exclude = [1]
# apply to EEG data
ica.apply(raw)
```

```
Applying ICA to Raw instance
    Transforming to ICA space (29 components)
    Zeroing out 1 ICA component
    Projecting back using 30 PCA components
```

Out[22]:

▼ General

Measurement date	Unknown
Experimenter	Unknown
Participant	Unknown

▼ Channels

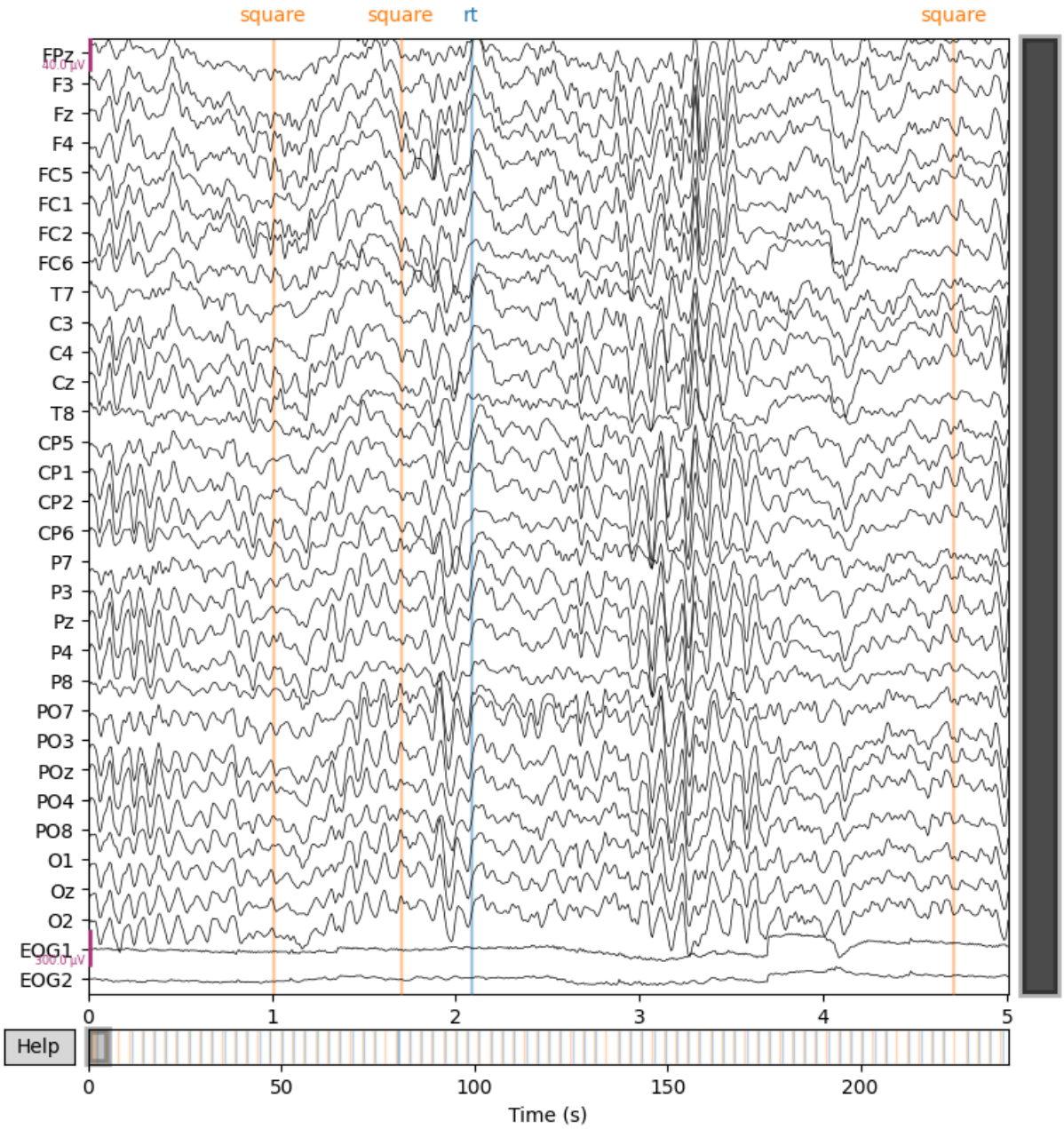
Digitized points	35 points
Good channels	30 EEG, 2 EOG
Bad channels	None
EOG channels	EOG1, EOG2
ECG channels	Not available

▼ Data

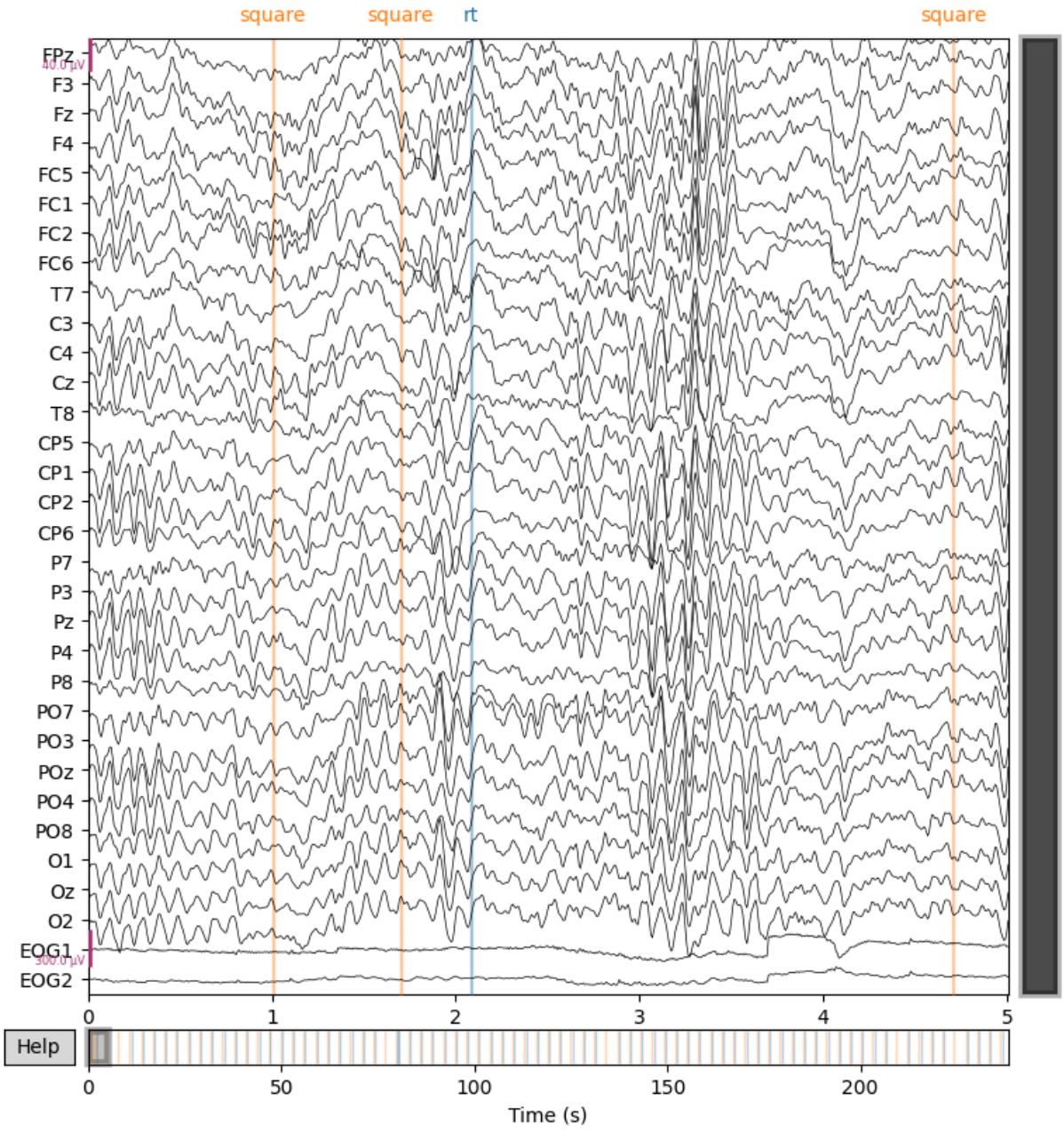
Sampling frequency	128.00 Hz
Highpass	0.10 Hz
Lowpass	30.00 Hz
Filenames	eeglab_data.fdt
Duration	00:03:59 (HH:MM:SS)

Plot the waveform of the data after ICA

In [23]:
```
raw.plot(duration=5, n_channels=32, clipping=None)
```


square
square
rt
square
FPz
F3
Fz
F4
FC5
FC1
FC2
FC6
T7
C3
C4
Cz
T8
CP5
CP1
CP2
CP6
P7
P3
Pz
P4
P8
PO7
PO3
POz
PO4
PO8
O1
Oz
O2
EOG1
EOG2
40.0 µV
300.0 µV
0
1
2
3
4
5
Help
0
50
100
150
200
Time (s)

Out[23]:



Step 4 Setting the Reference

If you want to use the papillary reference method, usually we use 'TP9' and 'TP10' as reference channels. You could use the following codes:

```
raw.set_eeg_reference(ref_channels=['TP9','TP10'])
```

If you want to use the average reference method, you could use the following code:

```
raw.set_eeg_reference(ref_channels='average')
```

If you want to use the REST reference method, you could use the following code： Here you need to pass in a forward parameter, for details see the corresponding MNE introduction at:
https://mne.tools/stable/auto_tutorials/preprocessing/55_setting_eeg_reference.html

```
raw.set_eeg_reference(ref_channels='REST', forward=forward)
```

If you want to use a bipolar reference method, you could use the following code： (Here 'EEG X' and 'EEG Y' correspond to the anode and cathode leads used for reference, respectively)

```
raw_bip_ref = mne.set_bipolar_reference(raw, anode=['EEG X'], cathode=['EEG Y'])
```

Step 5 Segmenting Data into Epochs

Extracting Events Information

First, we need to identify the markers used for the segmentation.
Check the markers in the data:

In [24]:
```
print(raw.annotations)
```

```
<Annotations | 154 segments: rt (74), square (80)>
```

It can be seen that the data contains two types of markers, 'square' and 'rt'

MNE has two data structures to store event information, Events and Annotations

For the Annotations object, it uses string to represent the time type, as shown in the printout above.
The time information is represented as different time- points. And it contains the duration of each marker. If it is a instantaneous marker, the duration is 0.
The internal data type is List.

In [25]:
```
# Print event duration of each event in data based on Annotations

print(raw.annotations.duration)
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

In [26]:
```
# Print description information of events in data based on Annotations

print(raw.annotations.description)
```

```
['square' 'square' 'rt' 'square' 'rt' 'square' 'square' 'rt' 'square' 'rt'
 'square' 'rt' 'square' 'rt' 'square' 'rt' 'square' 'rt' 'square' 'rt'
 'square' 'rt' 'square' 'rt' 'square' 'rt' 'square' 'rt' 'square' 'rt'
 'square' 'rt' 'square' 'rt' 'square' 'rt' 'square' 'rt' 'square' 'rt'
 'square' 'rt' 'square' 'rt' 'square' 'rt' 'square' 'rt' 'square' 'rt'
 'square' 'square' 'rt' 'square' 'rt' 'square' 'rt' 'square' 'rt' 'square'
 'rt' 'square' 'rt' 'square' 'rt' 'square' 'rt' 'square' 'rt' 'square'
 'rt' 'square' 'rt' 'square' 'rt' 'square' 'rt' 'square' 'rt' 'square'
 'rt' 'square' 'rt' 'square' 'rt' 'square' 'rt' 'square' 'square' 'rt'
 'square' 'rt' 'square' 'rt' 'square' 'rt' 'square' 'rt' 'square' 'rt'
 'square' 'rt' 'square' 'rt' 'square' 'rt' 'square' 'rt' 'square' 'rt'
 'square' 'rt' 'square' 'rt' 'square' 'rt' 'square' 'rt' 'square' 'rt'
 'square' 'rt' 'square' 'rt' 'square' 'rt' 'square' 'rt' 'square' 'rt'
 'square' 'rt' 'square' 'rt' 'square' 'rt' 'square' 'square' 'rt' 'square'
 'rt' 'square' 'rt' 'square' 'rt' 'square' 'square' 'rt' 'square' 'rt'
 'square' 'rt' 'square' 'rt']
```

In [27]:
```
# Print start time of events in data based on Annotations

print(raw.annotations.onset)
```

```
[  1.000068   1.695381   2.082407   4.703193   5.148224   7.711006
  10.718818  11.303858  13.726631  14.116658  16.734443  17.187474
  19.742256  20.199287  22.750068  23.136095  25.757881  26.124906
  28.765693  29.140719  31.773506  32.116529  34.781318  35.24635
  37.789131  38.254163  40.796943  41.229973  43.804756  44.30079
  46.812568  47.156592  49.820381  50.273412  52.828193  53.242222
  55.836006  56.211031  58.843818  59.237845  61.851631  62.183654
  64.859443  65.257471  67.867256  68.598306  70.875068  71.277096
  73.882881  74.323911  76.890693  79.898506  80.269531  82.906318
  83.312346  85.914131  86.313158  88.921943  89.280968  91.929756
  92.323783  94.937568  95.296593  97.945381  98.37141  100.953193
 101.363221 103.961006 104.332031 106.968818 107.343844 109.976631
 110.425662 112.984443 113.410472 115.992256 116.367281 119.000068
 119.359093 122.007881 122.409908 125.015693 125.441722 128.023506
 128.422533 131.031318 131.433346 134.039131 137.046943 137.488974
 140.054756 140.445783 143.062568 143.507599 146.070381 146.527412
 149.078193 149.504222 152.086006 152.465032 155.093818 155.479845
 158.101631 158.452655 161.109443 161.515471 164.117256 164.46928
 167.125068 167.543097 170.132881 170.507906 173.140693 173.636727
 176.148506 176.597537 179.156318 179.62535  182.164131 182.59016
 185.171943 185.678978 188.179756 188.585784 191.187568 191.675602
 194.195381 194.551405 197.203193 197.707228 200.211006 200.609033
 203.218818 203.667849 206.226631 206.65566  209.234443 212.242256
 212.621282 215.250068 215.641095 218.257881 218.667909 221.265693
 221.617717 224.273506 227.281318 227.718348 230.289131 230.71116
 233.296943 233.729973 236.304756 236.753787]
```

For the Events object, it is an event record data type used for data segmentation.
It uses 'Event ID' which is an integer to encode the event type, and represents the time in the form of samples.
And it does not contain the duration information of the marker. Its internal data type is NumPy-Array.

Event Information Data Type Conversion

Convert event information of Annotations to Events:

In [28]:
```
events, event_id = mne.events_from_annotations(raw)
```

```
Used Annotations descriptions: ['rt', 'square']
```

'events' is the matrix of time related records. 'event_id' is the dictionary information of different markers.
Here, we print the shape and event_id of the events matrix.

In [29]:
```
print(events.shape, event_id)
```

```
(154, 3) {'rt': 1, 'square': 2}
```

'rt' marker corresponds to the number 1, 'square' marker corresponds to the number 2.
There are 154 markers in total.

Segmentating Data

Data segmentation is based on Events.
Here, we extract the data from 1 second before the stimullus to 2 seconds after the stimulus, i.e. the data from -1s to 2s with the 'square' marker.
Take the baseline time interval from 0.5s before the stimulus to the onset of the stimulus. And set a threshold to exclude epochs i.e., the epoch that the difference between the largest and smallest values in it is greater than 2×10^-4 then will be excluded.
Note: The threshold value we use here is pretty large. Generally we recommend you to set the threshold between 5×10^-5 and 1×10^4 for good data quality.

In [30]:
```
epochs = mne.Epochs(raw, events, event_id=2, tmin=-1, tmax=2, baseline=(-0.5, 0),
                    preload=True, reject=dict(eeg=2e-4))
```

```
Not setting metadata
80 matching events found
Applying baseline correction (mode: mean)
0 projection items activated
Using data from preloaded Raw for 80 events and 385 original time points ...
0 bad epochs dropped
```

The segmented data is stored as the Epochs object.
Print epochs to see the information about the segmented data.

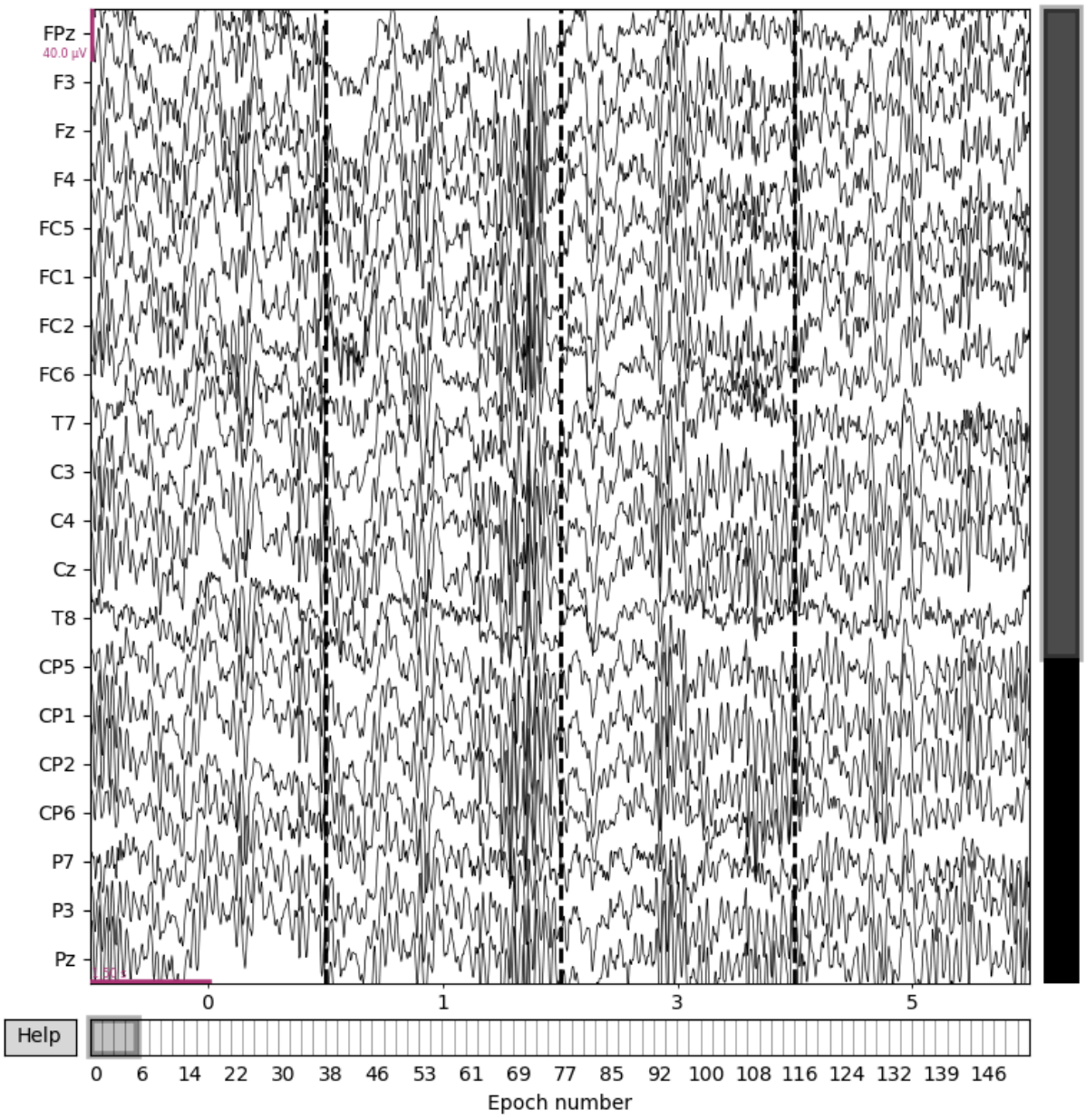
In [31]:
```
print(epochs)
```

```
<Epochs |  80 events (all good), -1 – 2 s, baseline -0.5 – 0 s, ~7.6 MB, data loaded,
 '2': 80>
```

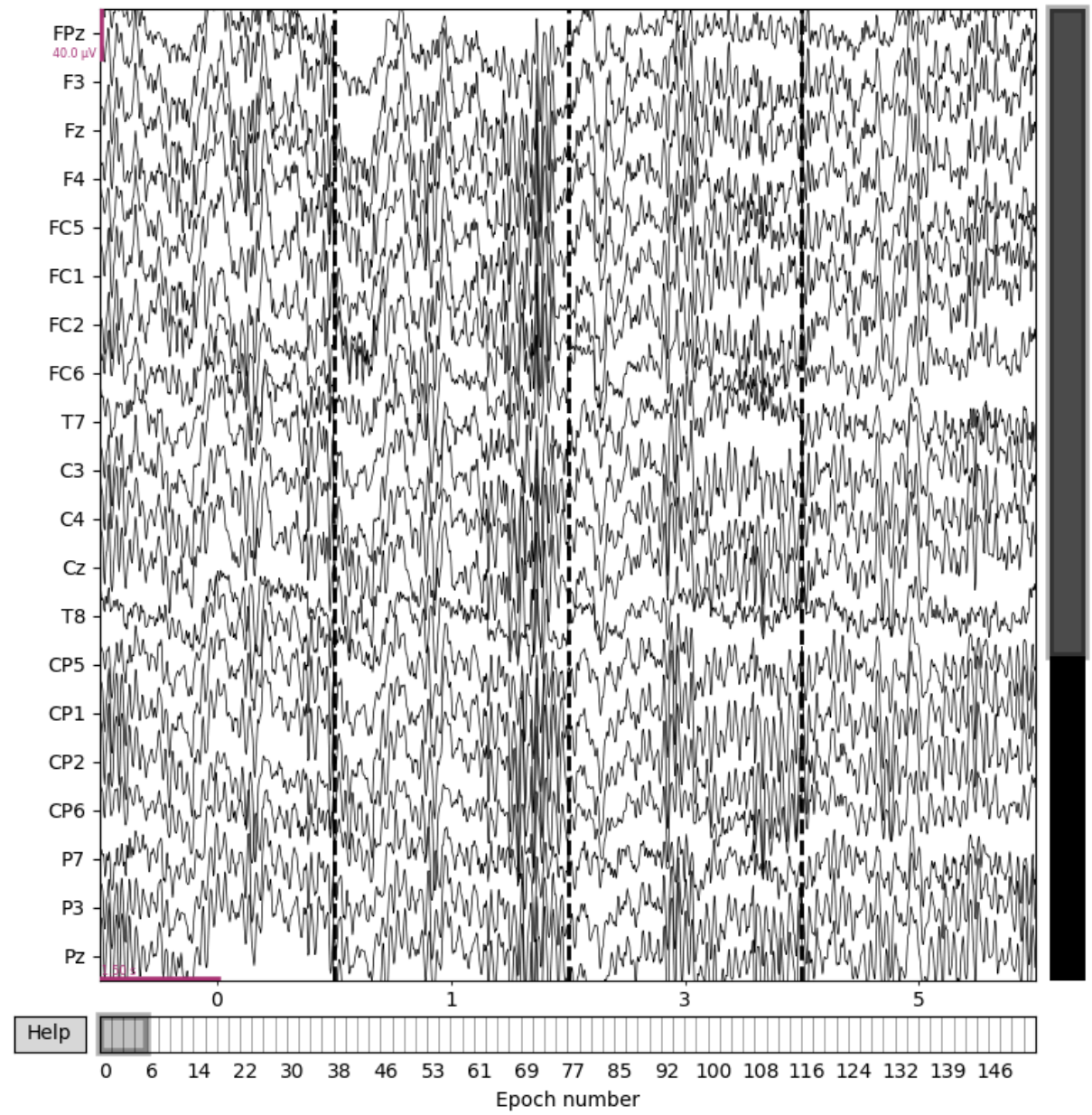
Visualizing Segmented Data

Visualize the segmented data (here, we show 4 epochs).

In [32]:
```
epochs.plot(n_epochs=4)
```



Out[32]:

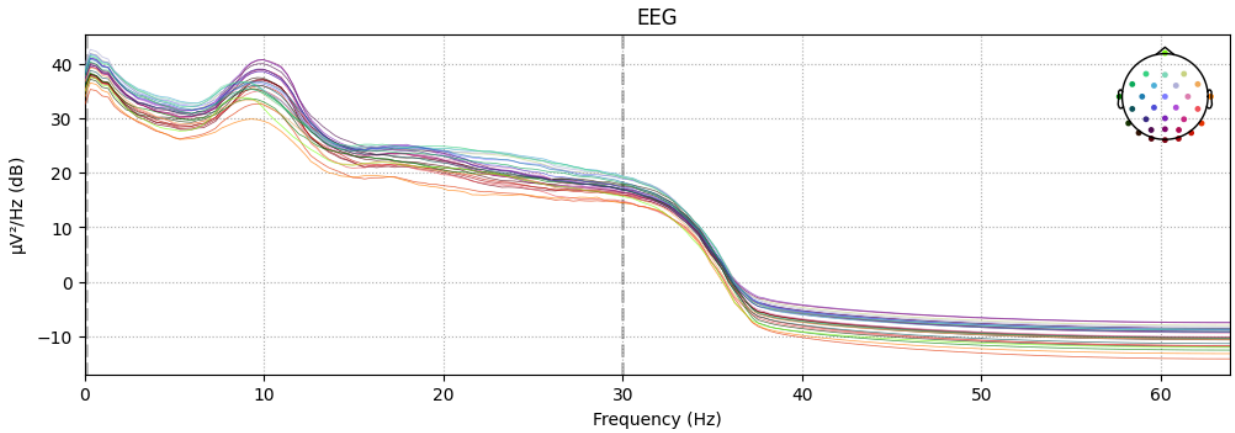


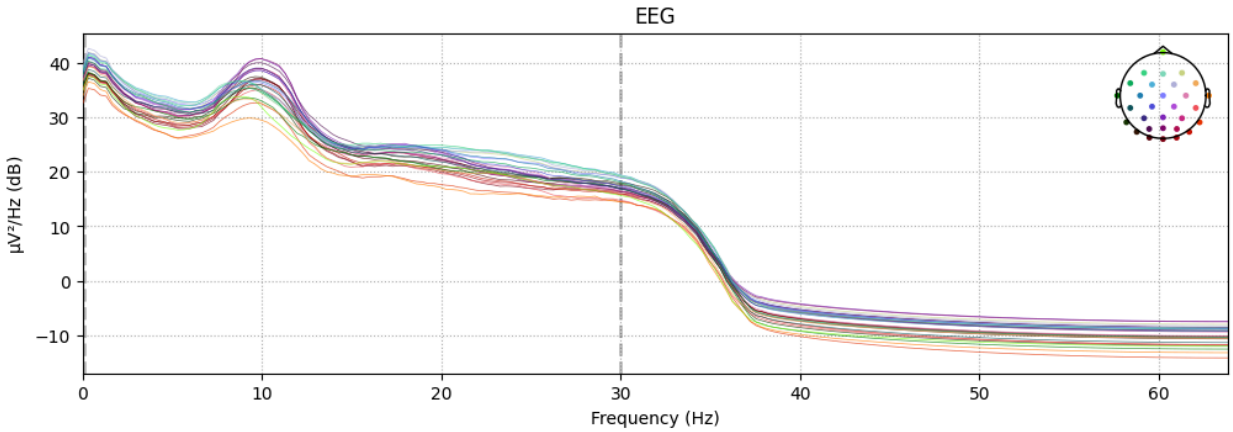
Plot the power spectrum (channel-by-channel):

In [33]:
```
epochs.compute_psd().plot(picks='eeg')
```

```
    Using multitaper spectrum estimation with 7 DPSS windows
Averaging across epochs...
/Users/zitonglu/anaconda3/lib/python3.11/site-packages/mne/viz/utils.py:165: UserWarning: Matplotlib
is currently using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so cannot
show the figure.
  (fig or plt).show(**kwargs)
```

Out[33]:

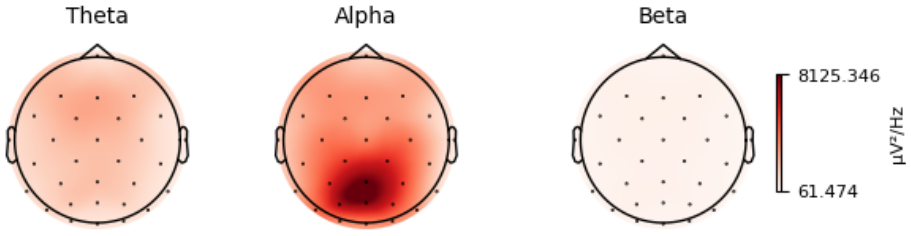




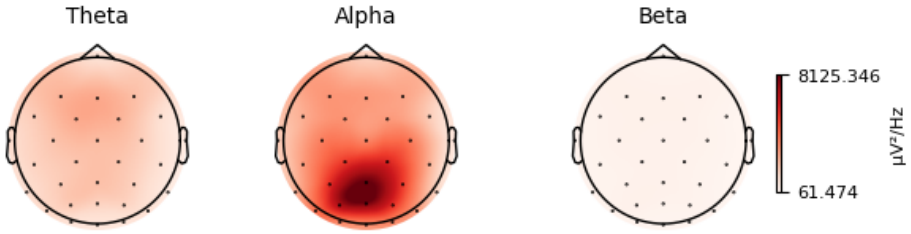
Plot the power spectrum topology (of Theta, Alpha and Beta bands):

In [34]:
```
bands = [(4, 8, 'Theta'), (8, 12, 'Alpha'), (12, 30, 'Beta')]
epochs.plot_psd_topomap(bands=bands, vlim='joint')
```

```
NOTE: plot_psd_topomap() is a legacy function. New code should use .compute_psd().plot_topomap().
    Using multitaper spectrum estimation with 7 DPSS windows
converting legacy list-of-tuples input to a dict for the `bands` parameter
```



Out[34]:



Step 6 Data Averaging

MNE uses the Epochs Class to store segmented data and the Evoked Class to store evoked (after averaging) data

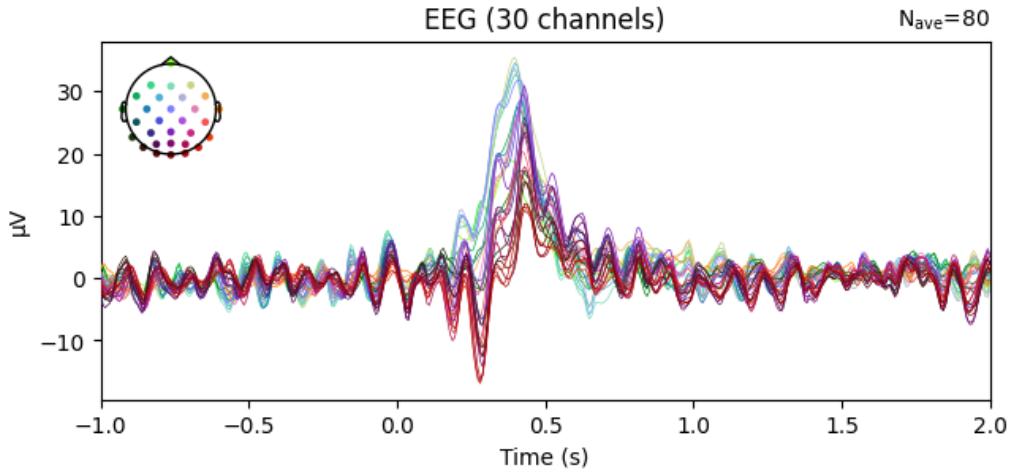
Average Data

In [35]:
```
evoked = epochs.average()
```

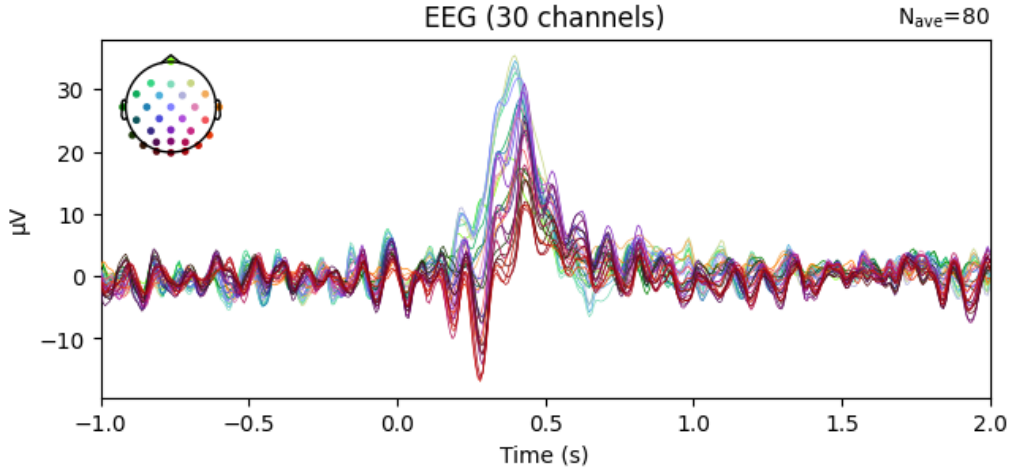
Visualize Evoked Data

Plot channel-wise timing signals

In [36]:
```
evoked.plot()
```

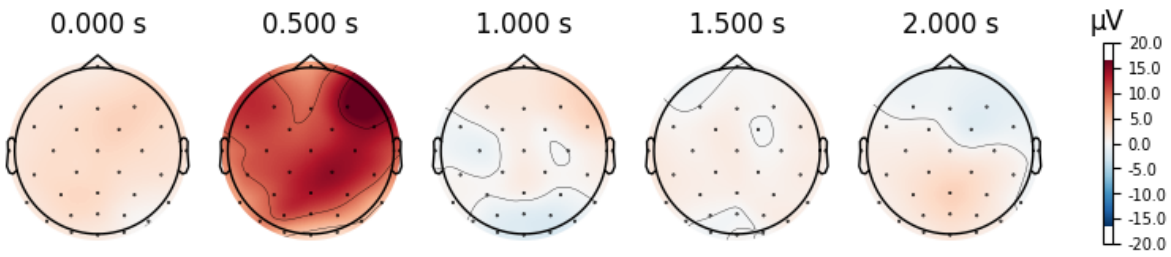


Out[36]:

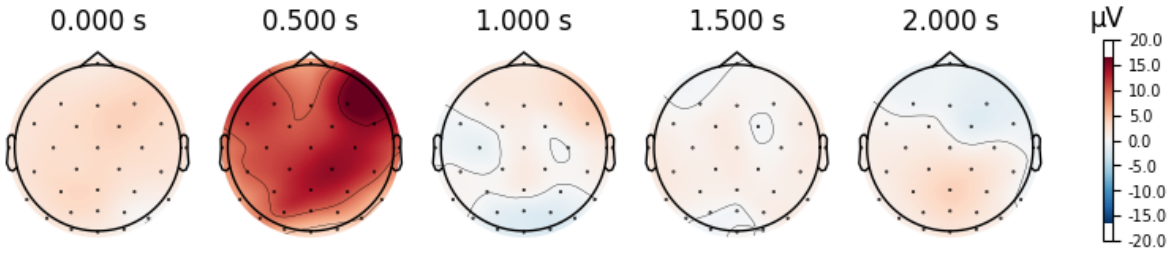


Plot topographic maps

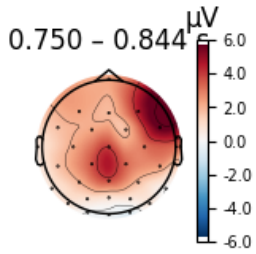
In [37]:
```
# Plot the topographic map at 0ms、0.5s、1s、1.5s and 2s
times = np.linspace(0, 2, 5)
evoked.plot_topomap(times=times, colorbar=True)
```



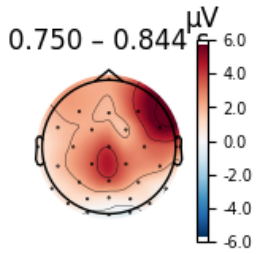
Out[37]:



In [38]:
```
# Plot the topographic map at a particular time-point
# In this example, we plot the figure at 0.8s
# where we take the mean value from 0.75 to 0.85s

evoked.plot_topomap(times=0.8, average=0.1)
```



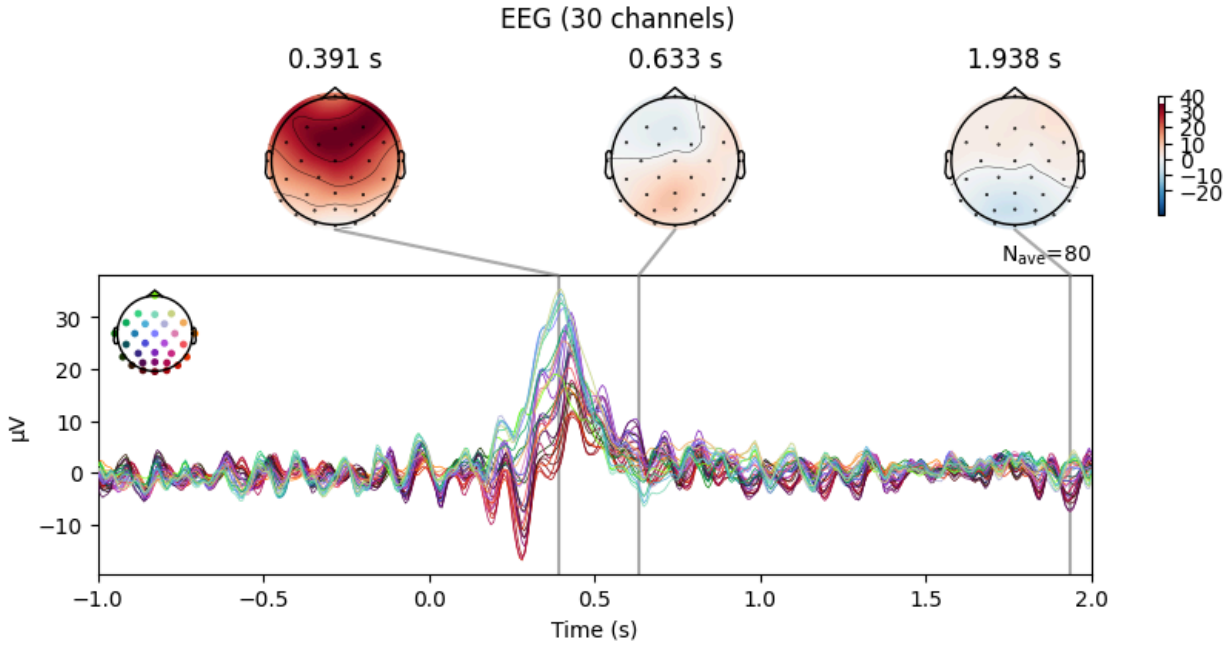
Out[38]:



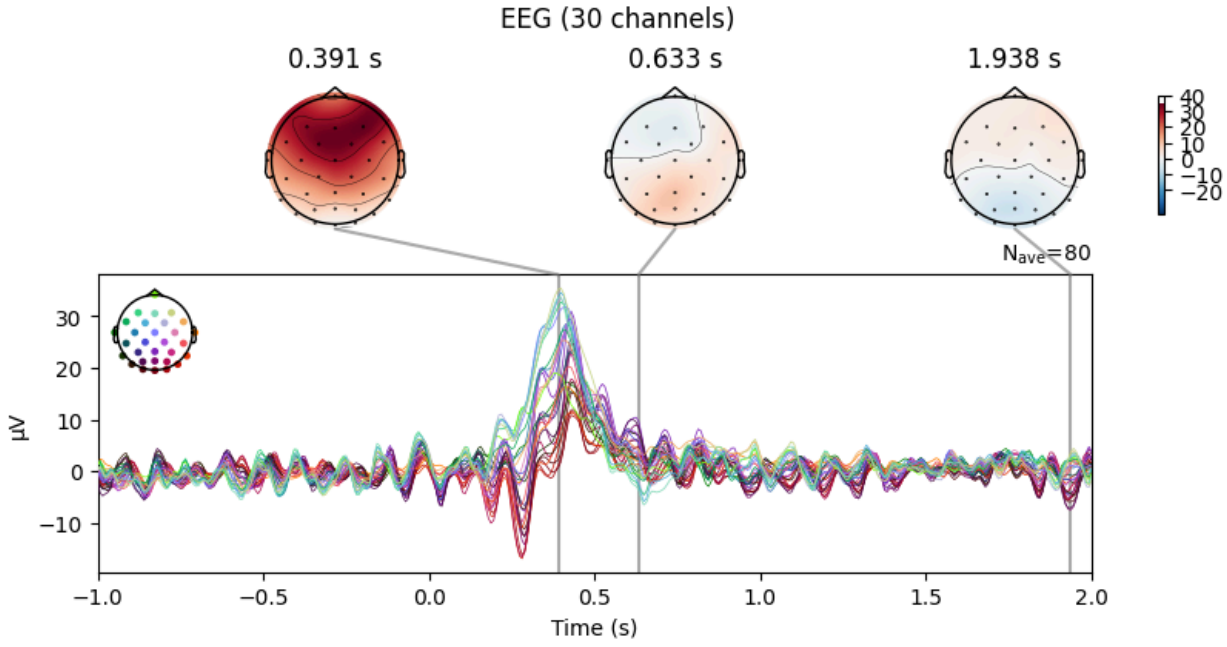
Plot evoked data as butterfly plot and add topomaps

In [39]:
```
evoked.plot_joint()
```

```
No projector specified for this dataset. Please consider the method self.add_proj.
```

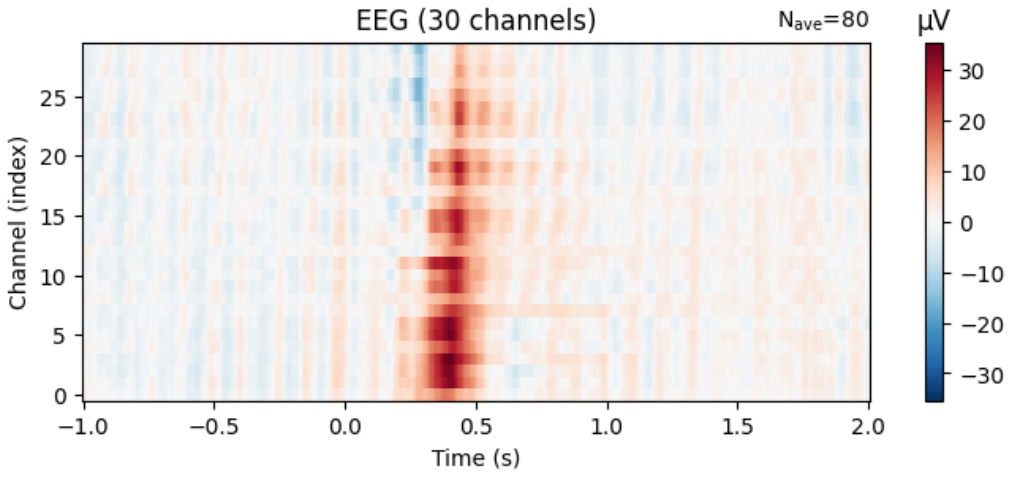


Out[39]:

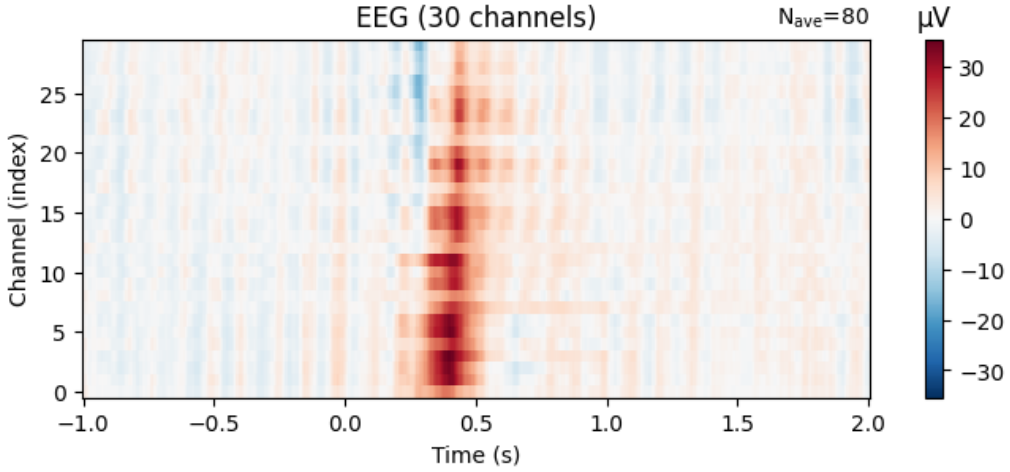


Plotting channel-wise hotmaps

In [40]:
```
evoked.plot_image()
```

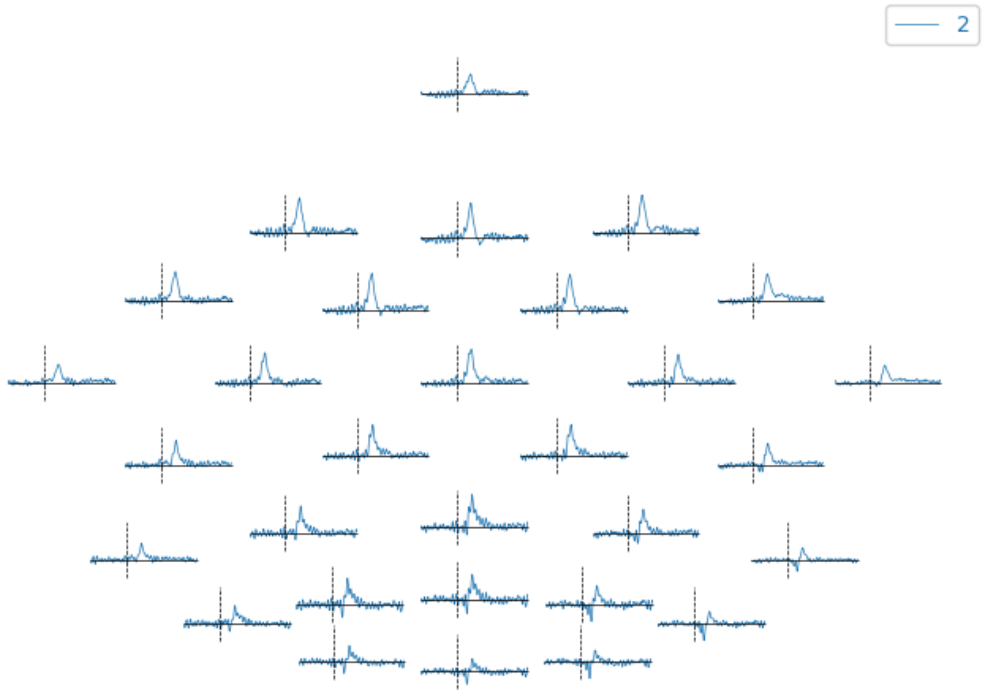


Out[40]:

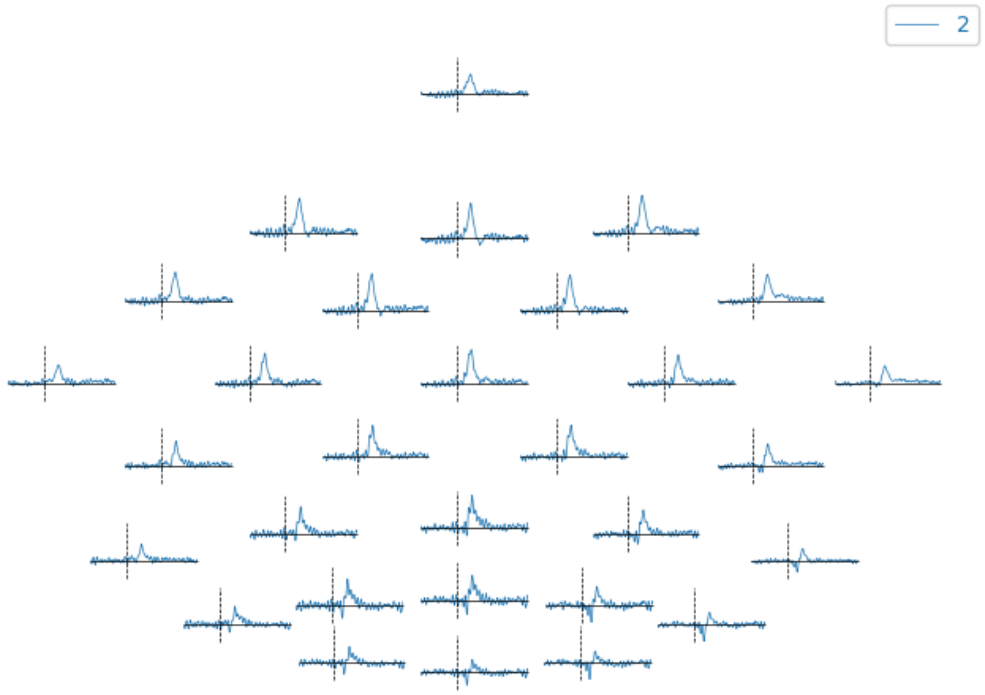


Plot 2D topography

In [41]:
```
evoked.plot_topo()
```



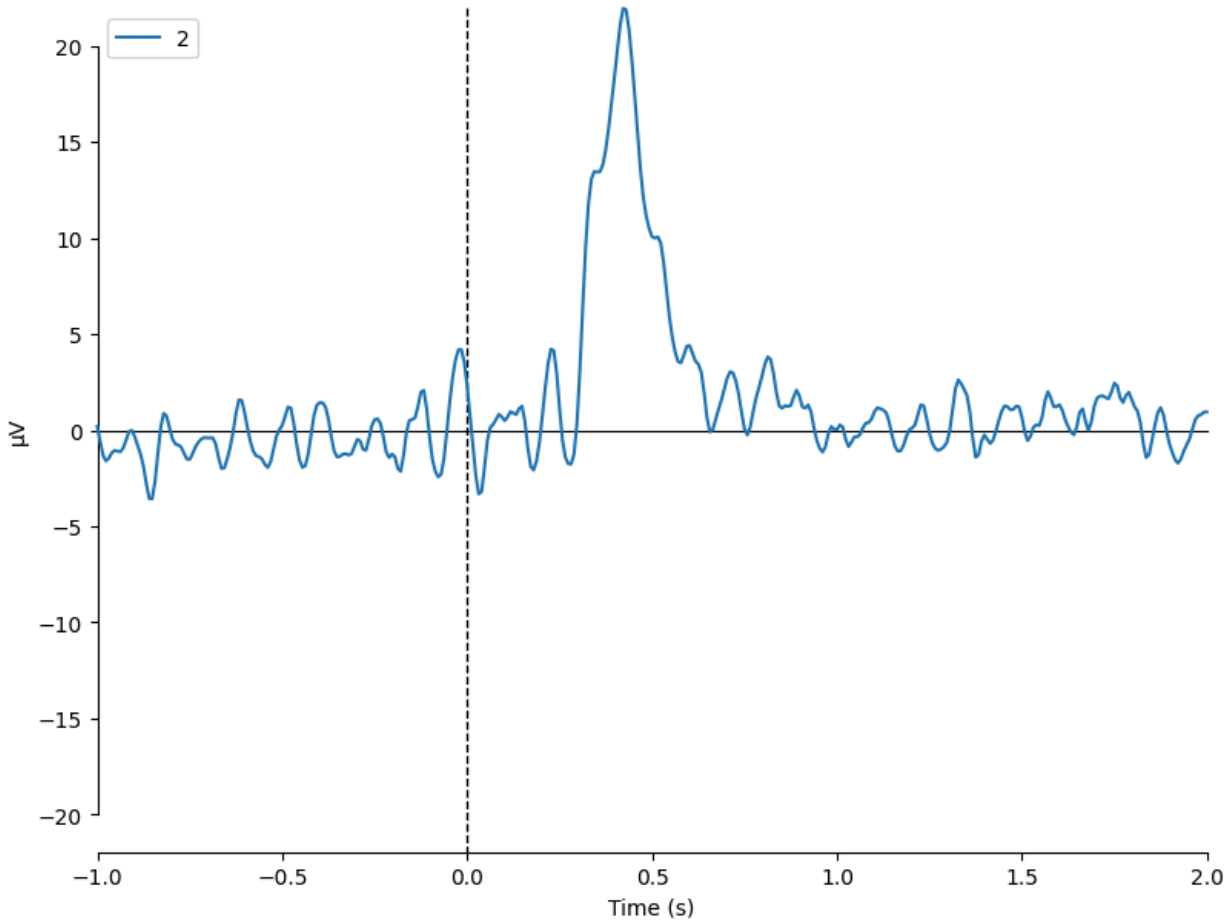
Out[41]:



Plot the averaged ERP of all channels

In [42]:
```
mne.viz.plot_compare_evokeds(evokeds=evoked, combine='mean')
```

```
combining channels using "mean"
```

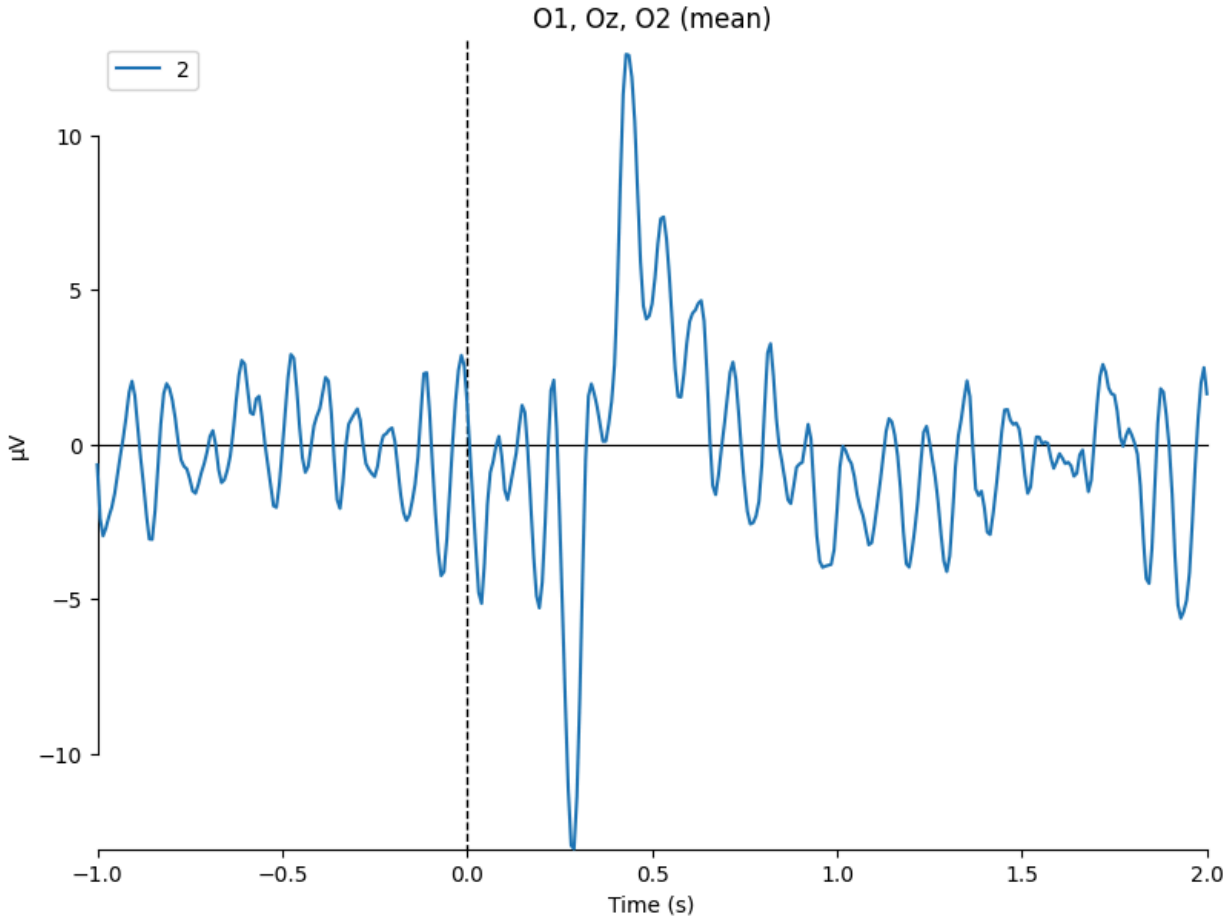


Out[42]:
```
[<Figure size 800x600 with 1 Axes>]
```

Plotting the averaged ERP of occipital channels

In [43]:
```
mne.viz.plot_compare_evokeds(evokeds=evoked, picks=['O1', 'Oz', 'O2'], combine='mean')
```

```
combining channels using "mean"
```



Out[43]: [<Figure size 800x600 with 1 Axes>]

Step 7 Time-Frequency Analysis

MNE provides three methods for time-frequency analysis, which are

- Morlet wavelets, corresponding to mne.time_frequency.tfr_morlet()
- DPSS tapers, corresponding to mne.time_frequency.tfr_multitaper()
- Stockwell Transform, corresponding to mne.time_frequency.tfr_stockwell()

Here, the first method is used as an example:

Conduct Time-Frequency Analysis

Calculate power and inter-trial coherence (ITC)

In [44]:
```
# Set some parameters for time-frequency analysis
# The frequency band is selected from 4 to 30 Hz

freqs = np.logspace(*np.log10([4, 30]), num=10)
n_cycles = freqs / 2.
power, itc = tfr_morlet(epochs, freqs=freqs, n_cycles=n_cycles, use_fft=True)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.2s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  30 out of  30 | elapsed:    1.8s finished
```

The returned power is the Power result, and itc is the inter-trial coupling result.

MNE returns the average result of all trials by default.

If you want to get the separate time-frequency analysis results of each trial, you can just set the average parameter to False.

The corresponding code can be modified as follows:

> power, itc = tfr_morlet(epochs, freqs=freqs, n_cycles=n_cycles, use_fft=True, average=False)

Plot the Time-Frequency Result

The visualization methods of time-frequency in MNE allows us to select different baseline correlation methods. The corresponding parameter is mode, which includes the following options:

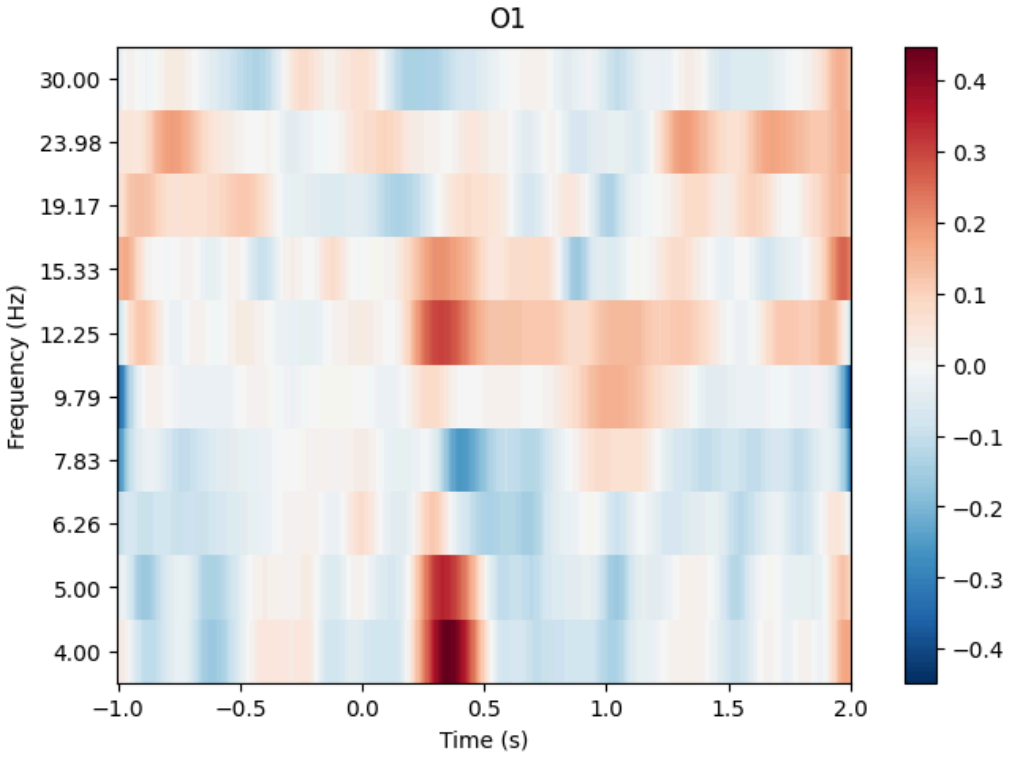
- 'mean', to subtract the baseline mean
- 'ratio', to divide by the baseline mean
- 'logratio', to divide by the baseline mean and take log
- 'percent', to subtract the baseline mean and divide by the baseline mean
- 'zscore', to subtract the baseline mean and divide by the baseline standard deviation
- 'zlogratio', to divide by the baseline mean and take the log and then divide by the standard deviation of the baseline after taking the log

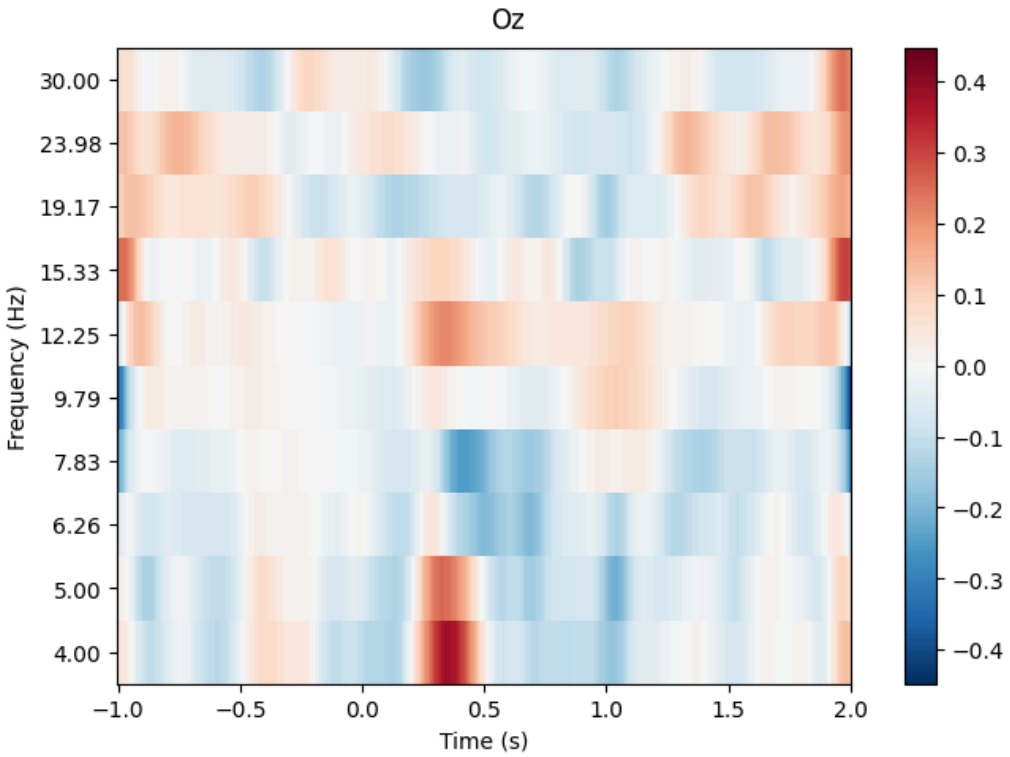
In the following example, the logratio method is chosen for baseline correction.

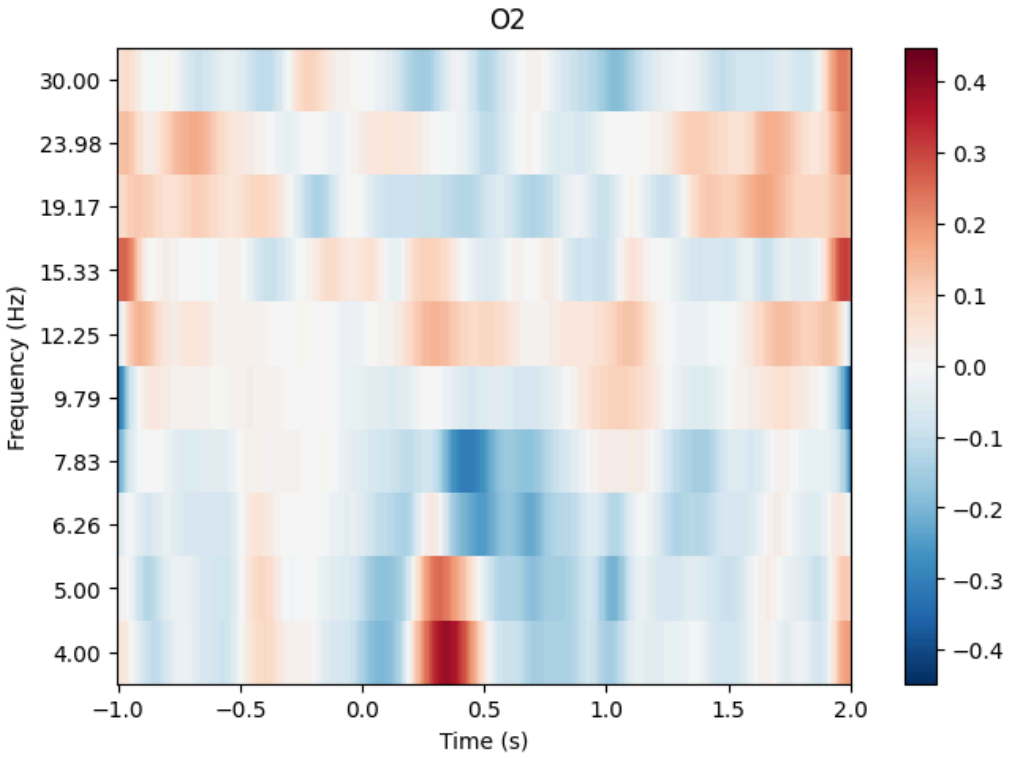
Plot power result of occipital channels

In [45]:
```
power.plot(picks=['O1', 'Oz', 'O2'], baseline=(-0.5, 0), mode='logratio', title='auto')
```

```
Applying baseline correction (mode: logratio)
```





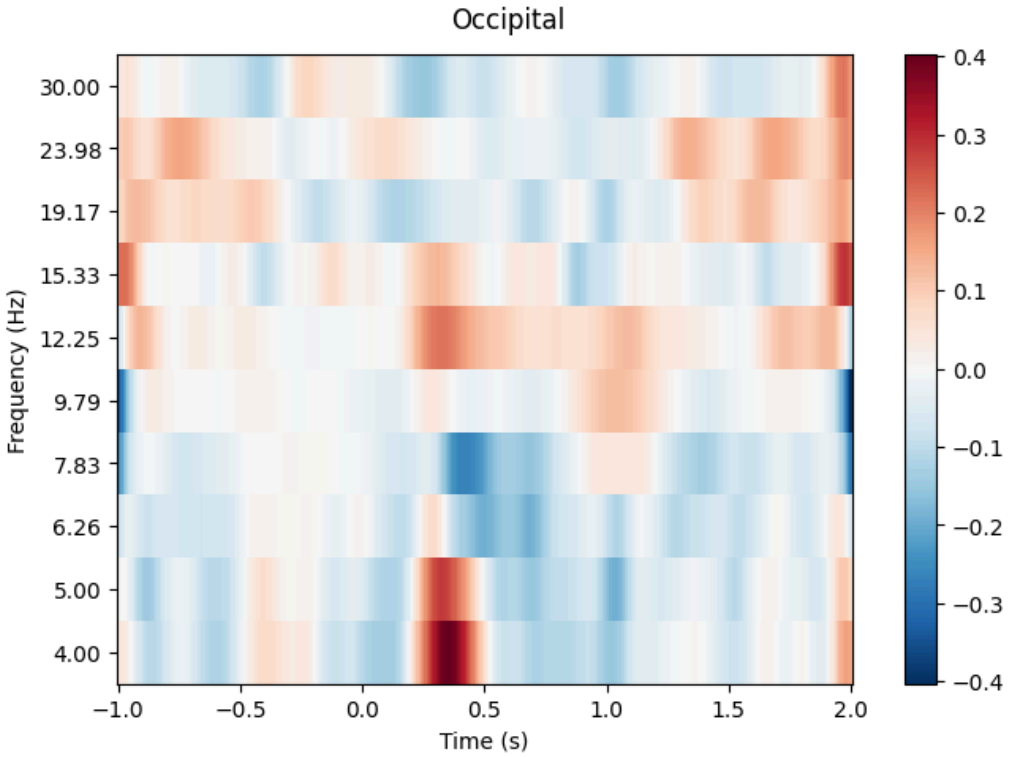


Out[45]:
```
[<Figure size 640x480 with 2 Axes>,
 <Figure size 640x480 with 2 Axes>,
 <Figure size 640x480 with 2 Axes>]
```

Plot mean power result of occipital channels

In [46]:
```
power.plot(picks=['O1', 'Oz', 'O2'], baseline=(-0.5, 0), mode='logratio',
           title='Occipital', combine='mean')
```

```
Applying baseline correction (mode: logratio)
```

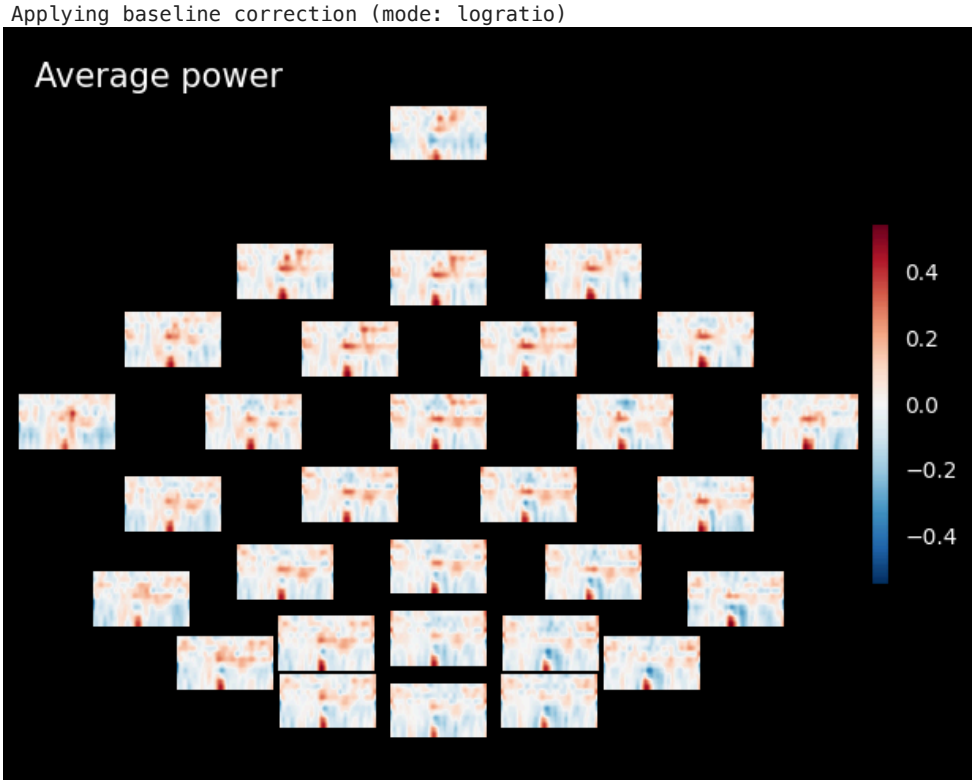


Out[46]: [<Figure size 640x480 with 2 Axes>]

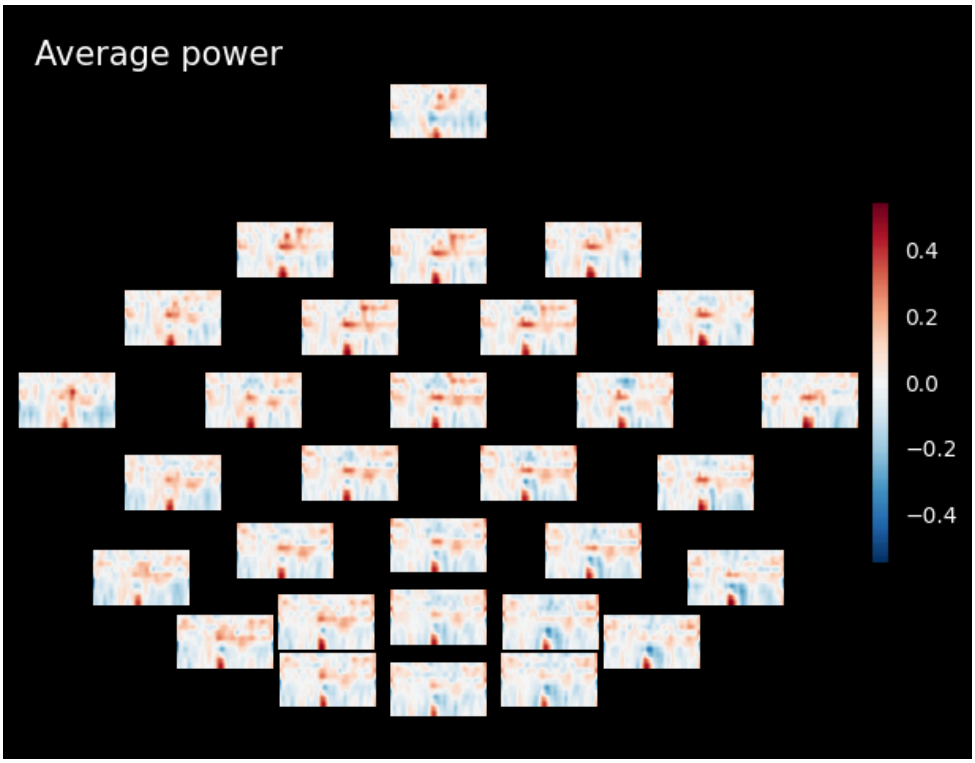
Plot 2D power topology

In [47]:
```
power.plot_topo(baseline=(-0.5, 0), mode='logratio', title='Average power')
```

```
Applying baseline correction (mode: logratio)
```



Out[47]:

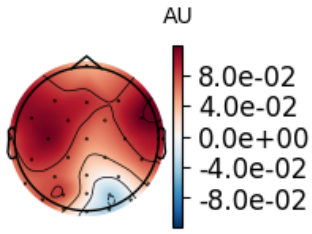


Plot power topology for different frequencies

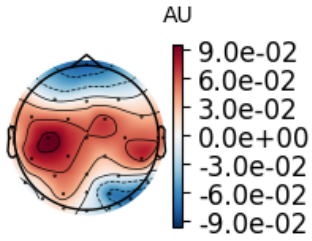
In [48]:

```
# Take theta power and alpha powers as an example
# Take the result of 0-0.5s

power.plot_topomap(tmin=0, tmax=0.5, fmin=4, fmax=8,
                   baseline=(-0.5, 0), mode='logratio')
power.plot_topomap(tmin=0, tmax=0.5, fmin=8, fmax=12,
                   baseline=(-0.5, 0), mode='logratio')
```

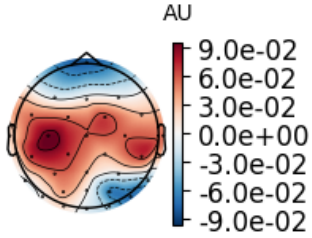
```
Applying baseline correction (mode: logratio)
```




```
Applying baseline correction (mode: logratio)
```



Out[48]:

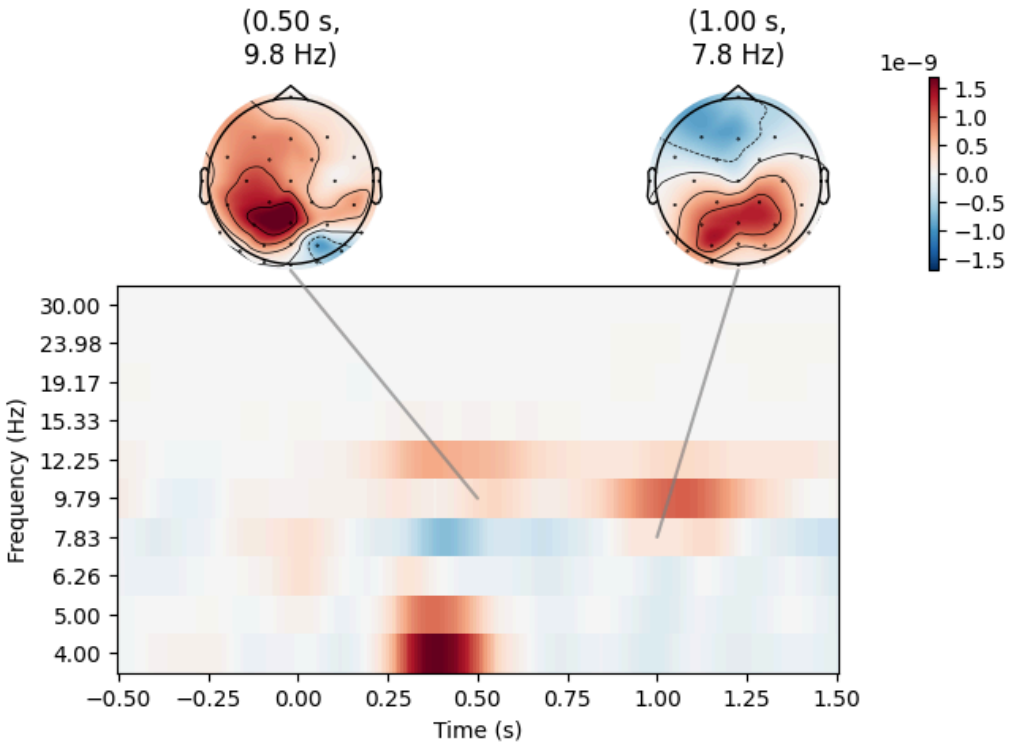


Plot joint figures

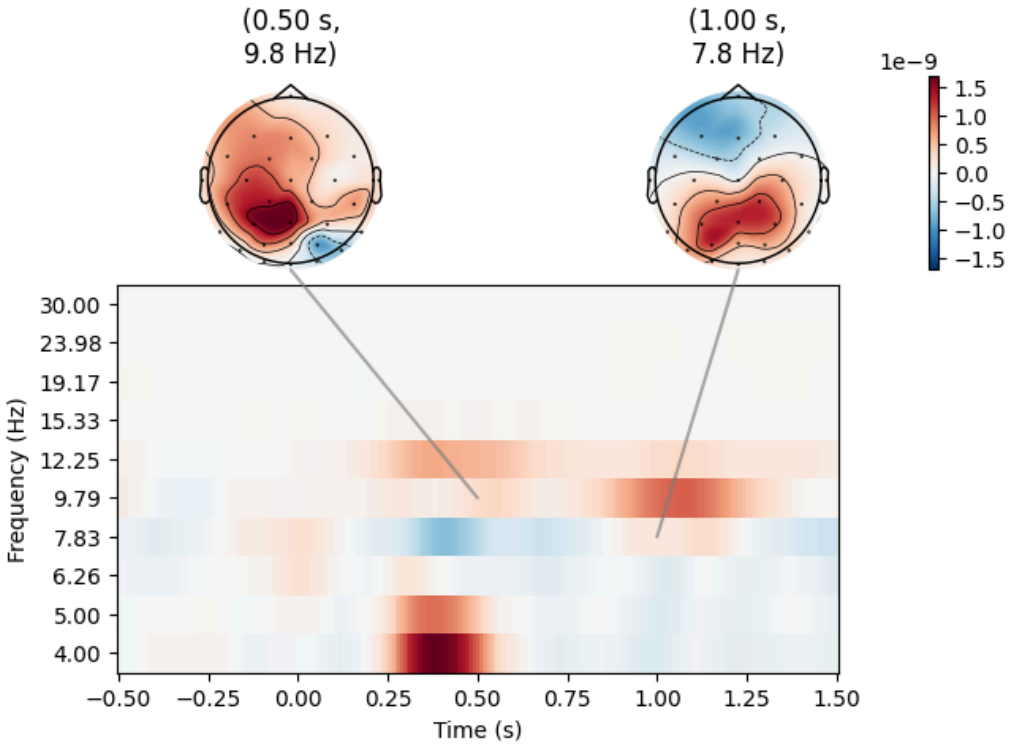
In [49]:

```
# And plot the results around 10Hz at 0.5s and around 8Hz at 1s
power.plot_joint(baseline=(-0.5, 0), mode='mean', tmin=-0.5, tmax=1.5,
                 timefreqs=[(0.5, 10), (1, 8)])
```

```
Applying baseline correction (mode: mean)
Applying baseline correction (mode: mean)
Applying baseline correction (mode: mean)
```



Out[49]:

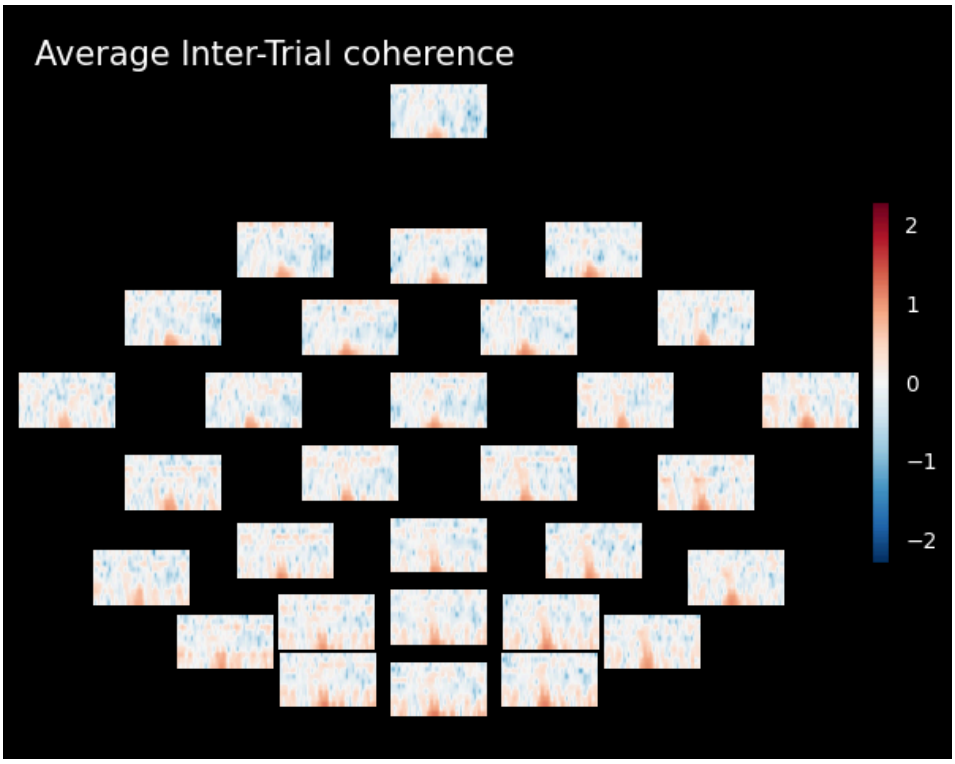


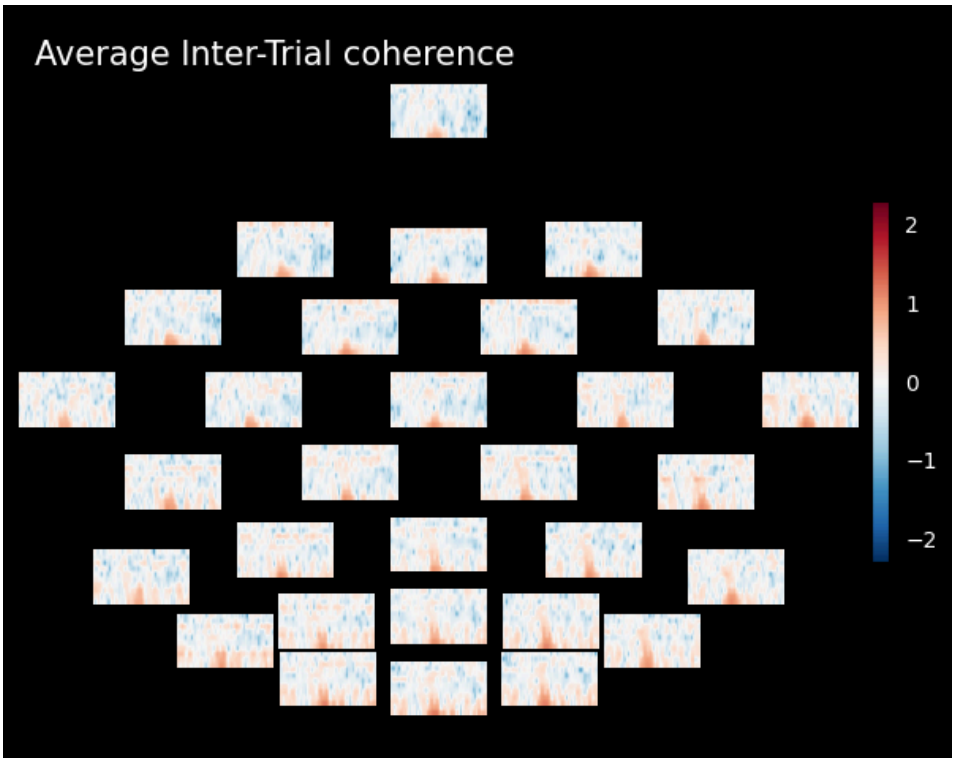
ITC results are plotted in similar ways. Below is an example of plotting ITC topology.

In [50]:
```
itc.plot_topo(baseline=(-0.5, 0), mode='logratio', title='Average Inter-Trial coherence')
```

```
Applying baseline correction (mode: logratio)
```

Out[50]:





Step 8 Data Extraction

After completing the relevant calculations, we always want to extract the raw data array, segmented data array, time-frequency result array, etc.
In MNE, Raw Class (raw data type), Epochs Class (segmented data type) and Evocked Class (averaged data type) provide get_data() method. And AverageTFR Class (time-frequency analysis data type) provides the .data attribute.

Use "get_data()"

Take epochs as an example:

In [51]:
```
epochs_array = epochs.get_data()
```

```
/var/folders/48/ln_7q6vj41sdf4mst103k9q80000gn/T/ipykernel_27170/4136209799.py:1: FutureWarning: The
current default of copy=False will change to copy=True in 1.7. Set the value of copy explicitly to a
void this warning
  epochs_array = epochs.get_data()
```

Check the data:

In [52]:
```
print(epochs_array.shape)
print(epochs_array)
```

```
(80, 32, 385)
[[[ 1.19955310e-05  2.64076199e-05  2.96414414e-05 ...  1.01356155e-05
    9.80684637e-06  5.92092751e-06]
  [ 1.90724252e-05  3.49982830e-05  1.52169891e-05 ... -6.16457047e-05
   -6.91169876e-05 -4.49072670e-05]
  [ 2.05216862e-05  3.59469862e-05  3.94453172e-05 ...  1.94767576e-05
    2.39333132e-05  2.49577850e-05]
  ...
  [-8.66725318e-07  8.10705304e-06  1.19969864e-05 ...  2.82622616e-05
    2.52288343e-05  2.71230860e-05]
  [-2.53263466e-06  7.27968446e-06  1.04266239e-05 ...  1.93294562e-05
    1.69034508e-05  2.34771912e-05]
  [ 1.33649215e-07  1.23793332e-05  1.50257433e-05 ...  1.05866334e-05
    9.54775567e-06  2.08057745e-05]]

 [[-2.86156092e-05 -2.46272912e-05 -2.49375585e-05 ... -5.66655490e-06
   -7.52341851e-06 -3.20816720e-06]
  [-1.98700747e-05 -2.32499213e-05 -2.34376177e-05 ... -5.87000498e-05
   -6.47002100e-05 -5.75247492e-05]
  [-4.77415730e-05 -4.28926395e-05 -4.62196702e-05 ... -1.70584942e-05
   -1.71572445e-05 -9.82390602e-06]
  ...
  [-1.60671591e-05 -1.70554320e-05 -1.78535513e-05 ... -2.06767937e-05
   -2.10371919e-05 -1.60745797e-05]
  [-1.27360726e-05 -1.29404692e-05 -1.56404187e-05 ... -2.57404057e-05
   -2.57716814e-05 -2.15169283e-05]
  [-3.91312740e-06 -4.33246287e-06 -8.74230064e-06 ... -2.92476907e-05
   -3.12189701e-05 -2.55777220e-05]]

 [[ 6.85227991e-06  1.32988569e-05  1.17931526e-05 ... -1.05257553e-05
   -7.29973247e-06  7.60811272e-07]
  [ 4.57868772e-05  6.10002561e-05  4.86857343e-05 ... -5.88723235e-05
   -4.53612780e-05 -5.21097750e-05]
  [-2.39041747e-06  5.51625605e-07 -3.43916043e-06 ... -2.09373615e-05
   -1.64900388e-05 -8.28851746e-06]
  ...
  [-1.33391776e-05 -1.07467635e-05 -1.24442829e-05 ... -1.64522834e-07
   -6.02052540e-06 -1.69078225e-05]
  [-2.08669303e-05 -1.78934662e-05 -1.64818362e-05 ...  3.76944599e-06
   -3.30115813e-06 -1.70946462e-05]
  [-2.83501975e-05 -2.21030099e-05 -1.98491421e-05 ...  8.76663948e-06
    1.64174717e-06 -1.52025402e-05]]

 ...

 [[ 6.80863318e-06  1.05524233e-05  6.72074726e-06 ... -4.01385062e-06
   -9.19784618e-07  1.06133759e-05]
  [ 5.76037713e-06  2.66259834e-05  8.75669214e-06 ... -3.11041534e-05
   -2.78780029e-05 -2.83221310e-05]
  [ 2.17025296e-05  3.54538639e-05  3.09487299e-05 ... -1.52174122e-05
   -1.63483974e-05  5.62132499e-06]
  ...
  [ 1.20074873e-05  2.12530233e-05  1.63303138e-05 ... -3.10716566e-06
    1.92047329e-06  6.47991816e-06]
  [ 1.46415609e-05  2.31887717e-05  1.69637313e-05 ...  7.89310003e-06
    1.46942781e-05  1.68083116e-05]
  [ 1.97671096e-05  2.89008509e-05  2.27972561e-05 ...  8.55728169e-06
    1.30367610e-05  1.41579469e-05]]

 [[ 1.74816850e-05  2.49819499e-05  2.29105866e-05 ...  3.81813476e-05
    4.09137267e-05  3.66925733e-05]
  [ 6.84401370e-06  8.58840419e-06  1.42037583e-06 ...  3.12988143e-05
    4.72432342e-05  3.72120719e-05]
  [ 2.43129604e-05  4.65274677e-05  4.61777299e-05 ...  2.78136110e-05
    3.18060014e-05  3.07669788e-05]
  ...
  [ 3.25606845e-06  8.07309023e-06  8.90675997e-06 ... -1.18110444e-05
   -4.80246809e-06 -9.39073249e-06]
  [ 1.25935768e-06  1.87061553e-06  1.89885310e-06 ... -1.64250945e-05
   -1.53387943e-05 -2.01224886e-05]
  [-3.15113541e-06 -5.09472319e-06 -6.65621545e-06 ... -2.17458072e-05
   -2.18670127e-05 -2.41018918e-05]]

 [[ 3.64681540e-05  2.98502815e-05  2.87498156e-05 ...  1.55175056e-05
    1.44826216e-05  1.30606681e-05]
  [ 2.55042060e-05  2.24582884e-05  1.90968657e-05 ...  2.20694343e-05
    1.64899586e-05  1.74567354e-05]
  [ 3.97881502e-05  3.42533154e-05  3.30624925e-05 ...  1.20407018e-05
    1.41817010e-05  2.08039910e-05]
  ...
  [-2.56909640e-05 -2.45978874e-05 -7.55036844e-06 ... -1.65130791e-05
```

```
 -1.38872442e-05 -7.89003310e-06]
[-2.81632418e-05 -2.33694463e-05 -6.84977804e-06 ... -1.02391454e-05
 -1.11644715e-05 -9.53458652e-06]
[-2.83165245e-05 -2.30232812e-05 -9.50182053e-06 ... -9.52254434e-06
 -1.06435252e-05 -1.02143019e-05]]]
```

Thus we can obtain the segmented data (NumPy Array type), and its shape is [80, 32, 385] which corresponds to 80 trials, 32 channels and 385 time points respectively.

If you want to obtain EEG data (exclude EOG or other channels' data), the above code can be modified to:

> epochs_array = epochs.get_data(picks=['eeg'])

Use ".data"

In [53]:
```
power_array = power.data
```

Check the data:

In [54]:
```
print(power_array.shape)
print(power_array)
```

```
(30, 10, 385)
[[[8.85986698e-10 9.07973706e-10 9.20058263e-10 ... 1.23420496e-09
   1.24995475e-09 1.24461341e-09]
  [7.32918844e-10 7.64367275e-10 7.87204341e-10 ... 9.84102647e-10
   9.74923498e-10 9.50824443e-10]
  [6.82156256e-10 7.29602613e-10 7.70687201e-10 ... 9.47208394e-10
   9.01253689e-10 8.46802067e-10]
  ...
  [8.68702339e-11 9.23412004e-11 9.71742416e-11 ... 1.04863686e-10
   9.93734589e-11 9.31959758e-11]
  [5.78475285e-11 6.18542974e-11 6.54692807e-11 ... 6.66801457e-11
   6.31275474e-11 5.91835037e-11]
  [3.59707583e-11 3.76646671e-11 3.89359327e-11 ... 4.17503064e-11
   4.02655814e-11 3.83566425e-11]]

 [[2.08549861e-09 2.15718563e-09 2.20604952e-09 ... 3.20895308e-09
   3.24354417e-09 3.22755190e-09]
  [1.67844510e-09 1.76829060e-09 1.84030892e-09 ... 2.54055386e-09
   2.51505487e-09 2.45304872e-09]
  [1.49721569e-09 1.61508215e-09 1.72157081e-09 ... 2.29404750e-09
   2.19507946e-09 2.07371169e-09]
  ...
  [1.95450368e-10 2.07028295e-10 2.17023523e-10 ... 2.28229601e-10
   2.18259949e-10 2.06502462e-10]
  [1.35133996e-10 1.44986198e-10 1.53958015e-10 ... 1.50674004e-10
   1.43372628e-10 1.35093542e-10]
  [8.47464664e-11 8.91013298e-11 9.25126929e-11 ... 9.65906831e-11
   9.31846381e-11 8.88114337e-11]]

 [[2.22295166e-09 2.31660017e-09 2.38604514e-09 ... 3.51045316e-09
   3.54134285e-09 3.51494742e-09]
  [1.77295781e-09 1.88450427e-09 1.97858384e-09 ... 2.74909783e-09
   2.71253358e-09 2.63627140e-09]
  [1.51076595e-09 1.63970135e-09 1.75859273e-09 ... 2.41453915e-09
   2.30404228e-09 2.17050331e-09]
  ...
  [1.96780351e-10 2.08829235e-10 2.19371880e-10 ... 2.34532464e-10
   2.23558232e-10 2.10880473e-10]
  [1.43061998e-10 1.53952322e-10 1.64030574e-10 ... 1.51420722e-10
   1.45466513e-10 1.38281328e-10]
  [8.19397181e-11 8.62056052e-11 8.96162121e-11 ... 1.08230898e-10
   1.04721206e-10 1.00021570e-10]]

 ...

 [[9.84753972e-10 1.01080219e-09 1.02145662e-09 ... 1.42596061e-09
   1.41214770e-09 1.37788149e-09]
  [7.87141108e-10 8.13893902e-10 8.27357786e-10 ... 1.15045298e-09
   1.11912311e-09 1.07333810e-09]
  [8.25047362e-10 8.78968894e-10 9.22526977e-10 ... 1.15141125e-09
   1.08797105e-09 1.01514380e-09]
  ...
  [1.42752146e-10 1.51785397e-10 1.59695581e-10 ... 1.83127076e-10
   1.76466908e-10 1.68068714e-10]
  [6.48164587e-11 6.74976999e-11 6.94854672e-11 ... 9.04567891e-11
   8.84664171e-11 8.55142632e-11]
  [4.57731069e-11 4.82475434e-11 5.02649826e-11 ... 7.27526735e-11
   7.01859193e-11 6.68705327e-11]]

 [[9.65946445e-10 9.93734977e-10 1.00735001e-09 ... 1.23711642e-09
   1.21981722e-09 1.18547472e-09]
  [7.93019581e-10 8.22586010e-10 8.39437238e-10 ... 9.94701252e-10
   9.65426266e-10 9.24016665e-10]
  [8.40246245e-10 8.95797556e-10 9.40779043e-10 ... 1.01838574e-09
   9.61625413e-10 8.96635599e-10]
  ...
  [1.28203840e-10 1.34680477e-10 1.39926354e-10 ... 1.62472689e-10
   1.57977015e-10 1.51762295e-10]
  [6.42709357e-11 6.65861212e-11 6.81186377e-11 ... 8.14601056e-11
   8.07555706e-11 7.89986809e-11]
  [4.31713285e-11 4.50276028e-11 4.63719334e-11 ... 6.97878090e-11
   6.77566077e-11 6.49078578e-11]]

 [[1.07297817e-09 1.10189041e-09 1.11549707e-09 ... 1.53222904e-09
   1.52549468e-09 1.49442694e-09]
  [8.73017183e-10 9.06061390e-10 9.25793251e-10 ... 1.18258153e-09
   1.16023297e-09 1.12059001e-09]
  [8.99004157e-10 9.58896787e-10 1.00808198e-09 ... 1.14042800e-09
   1.08719320e-09 1.02230790e-09]
  ...
  [1.25757828e-10 1.32188552e-10 1.37498259e-10 ... 1.61857580e-10
```

```
  1.58183566e-10 1.52745621e-10]
 [7.07738227e-11 7.37483707e-11 7.58665593e-11 ... 9.27531229e-11
  9.17143856e-11 8.94828847e-11]
 [4.88687480e-11 5.11797142e-11 5.29157649e-11 ... 7.73128267e-11
  7.50204552e-11 7.18113525e-11]]]
```

Thus, we can obtain the power result of time-frequency (NumPy Array type), and its shape is [30, 10, 385] which corresponds to 30 channels, 10 frequencies and 385 time points, respectively.

Chapter 2: Basic Python Data Operations

According to the analytical skills that may be used in the process of EEG data processing, this chapter aimming to provide a basic tutorial of using Python to conduct array operations and statistical analysis is divided into three parts:

- **Part 1: Basic Array Operations**
- **Part 2: Basic Data Reading and Storage Operations**
- **Part 3: Basic Statistical Analysis**

Download & Import Python Packages

In [1]:
```
!pip install h5py
!pip install neurora

import numpy as np
import h5py
import scipy.io as sio
import matplotlib.pyplot as plt
from scipy.stats import ttest_1samp, ttest_ind, ttest_rel, f_oneway
from mne.stats import fdr_correction, f_mway_rm
from neurora.stuff import clusterbased_permutation_1d_1samp_1sided, permutation_test
```

```
Requirement already satisfied: h5py in /Users/zitonglu/anaconda3/lib/python3.11/site-packages (3.9.
0)
Requirement already satisfied: numpy>=1.17.3 in /Users/zitonglu/anaconda3/lib/python3.11/site-packag
es (from h5py) (1.23.5)
Requirement already satisfied: neurora in /Users/zitonglu/anaconda3/lib/python3.11/site-packages (1.
1.6.10)
Requirement already satisfied: numpy in /Users/zitonglu/anaconda3/lib/python3.11/site-packages (from
neurora) (1.23.5)
Requirement already satisfied: scipy>=1.6.2 in /Users/zitonglu/anaconda3/lib/python3.11/site-package
s (from neurora) (1.11.1)
Requirement already satisfied: mne in /Users/zitonglu/anaconda3/lib/python3.11/site-packages (from n
eurora) (1.6.1)
Requirement already satisfied: nibabel in /Users/zitonglu/anaconda3/lib/python3.11/site-packages (fr
om neurora) (5.2.0)
Requirement already satisfied: matplotlib in /Users/zitonglu/anaconda3/lib/python3.11/site-packages
(from neurora) (3.6.3)
Requirement already satisfied: nilearn in /Users/zitonglu/anaconda3/lib/python3.11/site-packages (fr
om neurora) (0.10.3)
Requirement already satisfied: scikit-learn in /Users/zitonglu/anaconda3/lib/python3.11/site-package
s (from neurora) (1.3.0)
Requirement already satisfied: scikit-image in /Users/zitonglu/anaconda3/lib/python3.11/site-package
s (from neurora) (0.20.0)
Requirement already satisfied: contourpy>=1.0.1 in /Users/zitonglu/anaconda3/lib/python3.11/site-pac
kages (from matplotlib->neurora) (1.0.5)
Requirement already satisfied: cycler>=0.10 in /Users/zitonglu/anaconda3/lib/python3.11/site-package
s (from matplotlib->neurora) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in /Users/zitonglu/anaconda3/lib/python3.11/site-pa
ckages (from matplotlib->neurora) (4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /Users/zitonglu/anaconda3/lib/python3.11/site-pa
ckages (from matplotlib->neurora) (1.4.4)
Requirement already satisfied: packaging>=20.0 in /Users/zitonglu/anaconda3/lib/python3.11/site-pack
ages (from matplotlib->neurora) (23.1)
Requirement already satisfied: pillow>=6.2.0 in /Users/zitonglu/anaconda3/lib/python3.11/site-packag
es (from matplotlib->neurora) (9.4.0)
Requirement already satisfied: pyparsing>=2.2.1 in /Users/zitonglu/anaconda3/lib/python3.11/site-pac
kages (from matplotlib->neurora) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in /Users/zitonglu/anaconda3/lib/python3.11/site
-packages (from matplotlib->neurora) (2.8.2)
Requirement already satisfied: tqdm in /Users/zitonglu/anaconda3/lib/python3.11/site-packages (from
mne->neurora) (4.65.0)
Requirement already satisfied: pooch>=1.5 in /Users/zitonglu/anaconda3/lib/python3.11/site-packages
(from mne->neurora) (1.8.1)
Requirement already satisfied: decorator in /Users/zitonglu/anaconda3/lib/python3.11/site-packages
(from mne->neurora) (5.1.1)
Requirement already satisfied: jinja2 in /Users/zitonglu/anaconda3/lib/python3.11/site-packages (fro
m mne->neurora) (3.1.2)
Requirement already satisfied: lazy-loader>=0.3 in /Users/zitonglu/anaconda3/lib/python3.11/site-pac
kages (from mne->neurora) (0.3)
Requirement already satisfied: joblib>=1.0.0 in /Users/zitonglu/anaconda3/lib/python3.11/site-packag
es (from nilearn->neurora) (1.2.0)
Requirement already satisfied: lxml in /Users/zitonglu/anaconda3/lib/python3.11/site-packages (from
nilearn->neurora) (4.9.3)
Requirement already satisfied: pandas>=1.1.5 in /Users/zitonglu/anaconda3/lib/python3.11/site-packag
es (from nilearn->neurora) (1.5.3)
Requirement already satisfied: requests>=2.25.0 in /Users/zitonglu/anaconda3/lib/python3.11/site-pac
kages (from nilearn->neurora) (2.31.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /Users/zitonglu/anaconda3/lib/python3.11/site
-packages (from scikit-learn->neurora) (2.2.0)
Requirement already satisfied: networkx>=2.8 in /Users/zitonglu/anaconda3/lib/python3.11/site-packag
es (from scikit-image->neurora) (3.1)
Requirement already satisfied: imageio>=2.4.1 in /Users/zitonglu/anaconda3/lib/python3.11/site-packa
ges (from scikit-image->neurora) (2.31.1)
Requirement already satisfied: tifffile>=2019.7.26 in /Users/zitonglu/anaconda3/lib/python3.11/site-
packages (from scikit-image->neurora) (2023.4.12)
Requirement already satisfied: PyWavelets>=1.1.1 in /Users/zitonglu/anaconda3/lib/python3.11/site-pa
ckages (from scikit-image->neurora) (1.4.1)
Requirement already satisfied: pytz>=2020.1 in /Users/zitonglu/anaconda3/lib/python3.11/site-package
s (from pandas>=1.1.5->nilearn->neurora) (2023.3.post1)
Requirement already satisfied: platformdirs>=2.5.0 in /Users/zitonglu/anaconda3/lib/python3.11/site-
packages (from pooch>=1.5->mne->neurora) (3.10.0)
Requirement already satisfied: six>=1.5 in /Users/zitonglu/anaconda3/lib/python3.11/site-packages (f
rom python-dateutil>=2.7->matplotlib->neurora) (1.16.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /Users/zitonglu/anaconda3/lib/python3.11/
site-packages (from requests>=2.25.0->nilearn->neurora) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in /Users/zitonglu/anaconda3/lib/python3.11/site-package
s (from requests>=2.25.0->nilearn->neurora) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /Users/zitonglu/anaconda3/lib/python3.11/site-p
ackages (from requests>=2.25.0->nilearn->neurora) (1.26.16)
Requirement already satisfied: certifi>=2017.4.17 in /Users/zitonglu/anaconda3/lib/python3.11/site-p
ackages (from requests>=2.25.0->nilearn->neurora) (2023.7.22)
```

```
Requirement already satisfied: MarkupSafe>=2.0 in /Users/zitonglu/anaconda3/lib/python3.11/site-pack
ages (from jinja2->mne->neurora) (2.1.1)
```

Part 1 Basic NumPy Array Operations

When processing data with Python, NumPy arrays are often the most likely data type to be used for analysis operations.
In the first part, we will introduce some basic yet important NumPy array operations and their implementations that may be involved in the data analysis process later.

Generating Arrays

In [2]:
```
# Generate an array filled with zeros with a shape of [3, 4]
A = np.zeros([3, 4])
print(A.shape)
print(A)
```

```
(3, 4)
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

In [3]:
```
# Generate an array filled with ones with a shape of [3, 4]
A = np.ones([3, 4])
print(A.shape)
print(A)
```

```
(3, 4)
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
```

In [4]:
```
# Generate a random array with a shape of [3, 4] (random value between 0 and 1)
A = np.random.rand(3, 4)
print(A.shape)
print(A)
```

```
(3, 4)
[[0.23392924 0.22035835 0.5638377  0.48848769]
 [0.47104945 0.18138117 0.70215049 0.18883969]
 [0.43616516 0.63167613 0.50099703 0.24018979]]
```

In [5]:
```
# Generate a random array with a shape of [3, 4] (random value between 20 and 80)
A = np.random.uniform(low=20, high=80, size=[3, 4])
print(A.shape)
print(A)
```

```
(3, 4)
[[55.9454466  63.76207084 35.20959362 69.44105043]
 [47.23437158 75.80405389 74.18094595 34.43744156]
 [35.05809095 50.07752316 54.6770267  60.10960428]]
```

Flattening the Matrix into the Vector

In [6]:
```
A = np.ravel(A)
print(A.shape)
print(A)
```

```
(12,)
[55.9454466  63.76207084 35.20959362 69.44105043 47.23437158 75.80405389
 74.18094595 34.43744156 35.05809095 50.07752316 54.6770267  60.10960428]
```

Modifying Array Sizes (Reshaping the Array)

In [7]:
```
# Reshape A to have a shape of [3, 4]
A = np.reshape(A, (3, 4))
print(A.shape)
print(A)
```

```
(3, 4)
[[55.9454466  63.76207084 35.20959362 69.44105043]
 [47.23437158 75.80405389 74.18094595 34.43744156]
 [35.05809095 50.07752316 54.6770267  60.10960428]]
```

Array Transposition

Transposing the 2-dimensional array

In [8]:
```
A = A.T
print(A.shape)
print(A)
```

```
(4, 3)
[[55.9454466  47.23437158 35.05809095]
 [63.76207084 75.80405389 50.07752316]
 [35.20959362 74.18094595 54.6770267 ]
 [69.44105043 34.43744156 60.10960428]]
```

Transposing the high-dimensional array

In [9]:
```
# First, generate a 3-dimensional array with a shape of [2, 3, 4]
B = np.random.rand(2, 3, 4)
print(B.shape)
print(B)
```

```
(2, 3, 4)
[[[0.28237637 0.77526207 0.13826887 0.90446258]
  [0.83693299 0.40884927 0.86007176 0.19918559]
  [0.56159299 0.44729714 0.45152516 0.81938012]]

 [[0.28728874 0.49376632 0.82602795 0.809945  ]
  [0.81126796 0.66065919 0.36565283 0.67277449]
  [0.0574314  0.05197957 0.12344968 0.58150866]]]
```

In [10]:
```
# Transpose the array to have a shape of [2, 4, 3]
B = np.transpose(B, (0, 2, 1))
print(B.shape)
print(B)
```

```
(2, 4, 3)
[[[0.28237637 0.83693299 0.56159299]
  [0.77526207 0.40884927 0.44729714]
  [0.13826887 0.86007176 0.45152516]
  [0.90446258 0.19918559 0.81938012]]

 [[0.28728874 0.81126796 0.0574314 ]
  [0.49376632 0.66065919 0.05197957]
  [0.82602795 0.36565283 0.12344968]
  [0.809945   0.67277449 0.58150866]]]
```

In [11]:
```
# Here's another example:
# Transpose the array with a shape of [2, 4, 3] to an array with a shape of [4, 3, 2]
B = np.transpose(B, (1, 2, 0))
print(B.shape)
print(B)
```

```
(4, 3, 2)
[[[0.28237637 0.28728874]
  [0.83693299 0.81126796]
  [0.56159299 0.0574314 ]]

 [[0.77526207 0.49376632]
  [0.40884927 0.66065919]
  [0.44729714 0.05197957]]

 [[0.13826887 0.82602795]
  [0.86007176 0.36565283]
  [0.45152516 0.12344968]]

 [[0.90446258 0.809945  ]
  [0.19918559 0.67277449]
  [0.81938012 0.58150866]]]
```

Array Merging

In [12]:
```
# First, generate an array with a shape of [4, 3, 3]
C = np.random.rand(4, 3, 3)
print(C.shape)
print(C)
```

```
(4, 3, 3)
[[[0.08809861 0.95260854 0.1491686 ]
  [0.12986342 0.03184737 0.27373474]
  [0.05778086 0.18012803 0.34049261]]

 [[0.46405186 0.13893962 0.75624289]
  [0.71479078 0.49171166 0.21432559]
  [0.03929975 0.48174964 0.9563534 ]]

 [[0.88337122 0.66038061 0.47654903]
  [0.42979245 0.81191699 0.56522443]
  [0.4229371  0.4152601  0.64921869]]

 [[0.68095355 0.4697602  0.75024432]
  [0.07283738 0.94909662 0.0476452 ]
  [0.86601746 0.94838068 0.62035814]]]
```

In [13]:
```
# Merge array B (with a shape of [4, 3, 2]) with array C (with a shape of [4, 3, 3])
# That is, merge along the third dimension, resulting in an array with a shape of [4, 3, 5]
D = np.concatenate((B, C), axis=2)
print(D.shape)
print(D)
```

```
(4, 3, 5)
[[[0.28237637 0.28728874 0.08809861 0.95260854 0.1491686 ]
  [0.83693299 0.81126796 0.12986342 0.03184737 0.27373474]
  [0.56159299 0.0574314  0.05778086 0.18012803 0.34049261]]

 [[0.77526207 0.49376632 0.46405186 0.13893962 0.75624289]
  [0.40884927 0.66065919 0.71479078 0.49171166 0.21432559]
  [0.44729714 0.05197957 0.03929975 0.48174964 0.9563534 ]]

 [[0.13826887 0.82602795 0.88337122 0.66038061 0.47654903]
  [0.86007176 0.36565283 0.42979245 0.81191699 0.56522443]
  [0.45152516 0.12344968 0.4229371  0.4152601  0.64921869]]

 [[0.90446258 0.809945   0.68095355 0.4697602  0.75024432]
  [0.19918559 0.67277449 0.07283738 0.94909662 0.0476452 ]
  [0.81938012 0.58150866 0.86601746 0.94838068 0.62035814]]]
```

In [14]:
```
# Here's another example:
# Generate an array E with a shape of [1, 3, 2]
# Merge array B and array E to obtain an array F with a shape of [5, 3, 2]
E = np.random.rand(1, 3, 2)
F = np.concatenate((B, E), axis=0)
print(F.shape)
print(F)
```

```
(5, 3, 2)
[[[0.28237637 0.28728874]
  [0.83693299 0.81126796]
  [0.56159299 0.0574314 ]]

 [[0.77526207 0.49376632]
  [0.40884927 0.66065919]
  [0.44729714 0.05197957]]

 [[0.13826887 0.82602795]
  [0.86007176 0.36565283]
  [0.45152516 0.12344968]]

 [[0.90446258 0.809945  ]
  [0.19918559 0.67277449]
  [0.81938012 0.58150866]]

 [[0.99912164 0.39421159]
  [0.03843999 0.48001635]
  [0.54657219 0.11174902]]]
```

Averaging Values in an Array

In [15]:
```
# Average the values along the second dimension of array B (with a shape of [4, 3, 2])
B_mean = np.average(B, axis=1)
print(B_mean.shape)
print(B_mean)
```

```
(4, 2)
[[0.56030078 0.38532937]
 [0.54380283 0.40213503]
 [0.4832886  0.43837682]
 [0.64100943 0.68807605]]
```

Converting Non-NumPy Object into NumPy Array

In [16]:
```
G = [[1, 2], [3, 4]]
G_narry = np.array(G)
print(type(G))
print(type(G_narry))
```

```
<class 'list'>
<class 'numpy.ndarray'>
```

Part 2 Basic Data Reading and Storage Operations

This part will introduce some basic operations for data (array) reading and storage based on Python.

Data Reading based on MNE

Refer to "Chapter 1: Preprocessing Single-subject Data" - "Step 1 Loading data" and "Step 8 Data Extraction".

Storing and Reading Data with h5py

The h5py library can store data in a highly compressible HDF5 format file.
HDF5 files are similar to MATLAB's .mat files.
Both are composed of Keys and datasets.
A Key can be understood as the name of the data, and a dataset can be understood as the specific data.
However, HDF5 has stronger compression performance than .mat files.
HDF5 files have the extension .h5.

Storing data with h5py

In [17]:
```
# Generate data with a shape of [4, 5]
testdata = np.random.rand(4, 5)
# Use h5py to store the data in a file named 'test_data.h5'
f = h5py.File('test_data.h5', 'w')
# Store the above testdata matrix using the Key+Dataset method,
# Here the Key is named 'data'
f.create_dataset('data', data=testdata)
# Close the file
f.close()
```

Reading data with h5py

In [18]:
```
# Read data
testdata = np.array(h5py.File('test_data.h5', 'r')['data'])
# Print the data
print(testdata.shape)
print(testdata)
```

```
(4, 5)
[[0.3978598  0.8996023  0.54708416 0.73816992 0.02574653]
 [0.44200296 0.12362103 0.17315831 0.95591195 0.12356735]
 [0.52541798 0.48842702 0.8370089  0.00473001 0.67011308]
 [0.19024479 0.21131253 0.50230262 0.86971355 0.08896379]]
```

Storing and Reading Data with NumPy

NumPy's built-in save() function can store the array as a .npy file.
Its built-in np.load() function can read data from a .npy file.

Storing data with NumPy

In [19]:
```
# Store the testdata in NumPy Array format in a file named 'test_data.npy'
np.save('test_data.npy', testdata)
```

Reading .npy file stored as NumPy array data

In [20]:
```
# Read 'test_data.npy'
testdata = np.load('test_data.npy')
```

```
# Print data info
print(testdata.shape)
print(testdata)
```

```
(4, 5)
[[0.3978598  0.8996023  0.54708416 0.73816992 0.02574653]
 [0.44200296 0.12362103 0.17315831 0.95591195 0.12356735]
 [0.52541798 0.48842702 0.8370089  0.00473001 0.67011308]
 [0.19024479 0.21131253 0.50230262 0.86971355 0.08896379]]
```

Storing and Reading 2-Dimensional Array into text file with NumPy

NumPy's built-in savetxt() function can store a two-dimensional array as a .txt file.
Its built-in loadtxt() function can read data from a .txt file.

Storing 2-dimensional array data with NumPy

In [21]:
```
# Store the NumPy array-format testdata in a file named 'test_data.txt'
np.savetxt('test_data.txt', testdata)
```

Reading .txt file stored 2-dimensional array data

In [22]:
```
# Read 'test_data.txt'
testdata = np.loadtxt('test_data.txt')
# Print data info
print(testdata.shape)
print(testdata)
```

```
(4, 5)
[[0.3978598  0.8996023  0.54708416 0.73816992 0.02574653]
 [0.44200296 0.12362103 0.17315831 0.95591195 0.12356735]
 [0.52541798 0.48842702 0.8370089  0.00473001 0.67011308]
 [0.19024479 0.21131253 0.50230262 0.86971355 0.08896379]]
```

Since the above methods based on NumPy are only suitable for two-dimensional data,
if you really want to store multidimensional data into the text file, you can first reshape it into two-dimensional data before storing it.
The following code is an example:

In [23]:
```
# Generate a 3-dimensional data with a shape of [2, 3, 4]
testdata_3d = np.random.rand(2, 3, 4)
# Reshape to [2, 12]
testdata_3d_to_2d = np.reshape(testdata_3d, (2, 12))
# Save as a .txt file
np.savetxt('test_data_3d_to_2d.txt', testdata_3d_to_2d)
# Read data
testdata_2d = np.loadtxt('test_data_3d_to_2d.txt')
# Reshape back to [2, 3, 4]
testdata_2d_to_3d = np.reshape(testdata_2d, (2, 3, 4))
```

Reading .mat Files as NumPy Array

In Python, reading MATLAB's .mat format files is mainly done through the following two methods:

Reading with Scipy.io

The following code is just an example, with no actual data.
filename is the directory of the .mat file to be read, Key is the saved variable name of the data to be read.

> import scipy.io as sio
> data = np.array(sio.loadmat(filename)[Key])

Reading with h5py

h5py can also be used to read .mat files, following the same steps as for reading .h5 files.
The above has been introduced, the following code is just an example, with no actual data.
This method doesn't support MAT files below version 7.3.

> data = np.array(h5py.File(filename, 'r')[Key])

Part 3: Basic Statistical Analysis

In this part, we will introduce a series of basic statistical operations when we conduct group analysis on EEG data from multiple subjects.

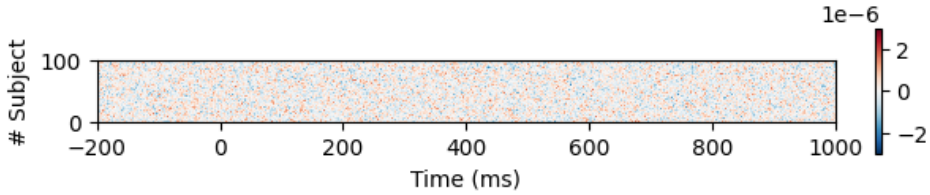
Descriptive Statistics

First, let's introduce the calculation methods for a few commonly used descriptive statistics indicators:

- Mean
- Variance
- Standard deviation
- Standard error

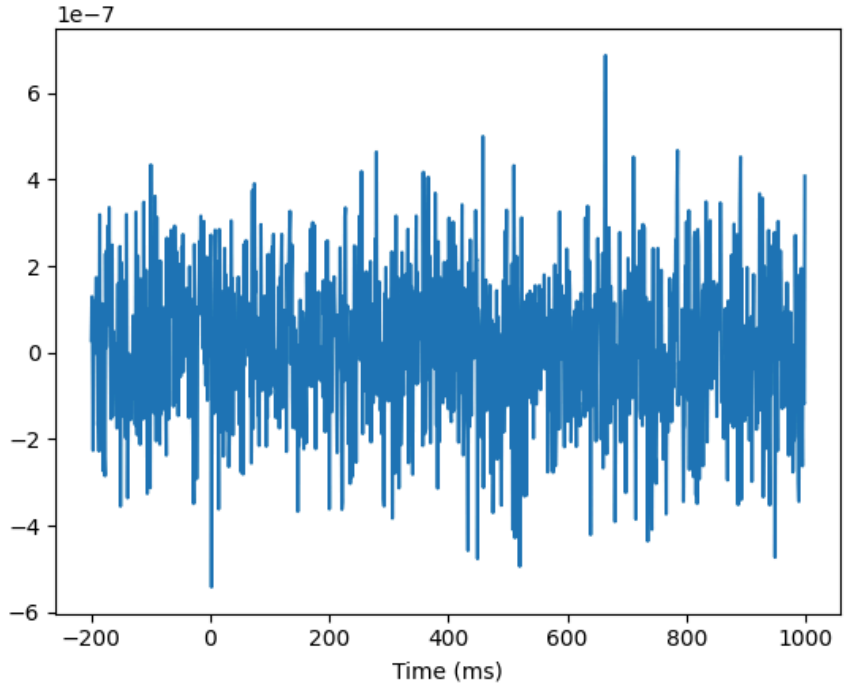
Here, we generate some fictitious EEG data for subsequent statistical analysis examples.
Assume we randomly generate the Event-Related Potentials (ERP) data of 10 subjects from -200ms to 1000ms (1200 time points, each corresponding to 1ms).

In [24]:
```
# Generate 'pseudo' EEG with a shape of [100, 1200]
data = np.random.uniform(low=-3e-6, high=3e-6, size=(100, 1200))
# Visual the data
plt.imshow(data, extent=[-200, 1000, 0, 100], cmap='RdBu_r')
plt.colorbar(fraction=0.008, ticks=[-2e-6, 0, 2e-6])
plt.xlabel('Time (ms)')
plt.ylabel('# Subject')
plt.show()
```



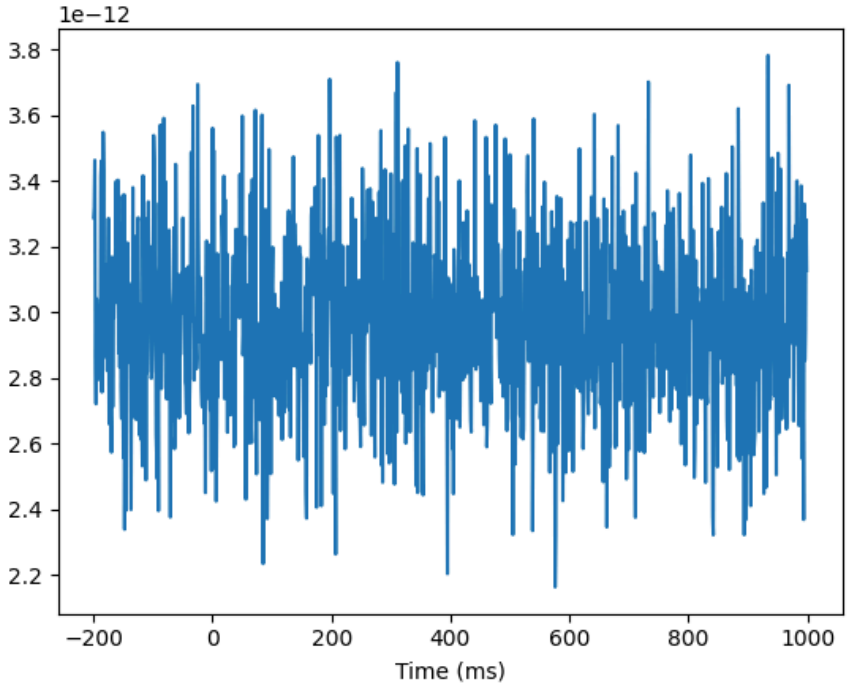
Calculate the mean ERP of 100 subjects

In [25]:
```
# Average across the dimensions of subjects
data_mean = np.mean(data, axis=0)
times = np.arange(-200, 1000)
# Visualize the mean 'pseudo' ERP
plt.plot(times, data_mean)
plt.xlabel('Time (ms)')
plt.show()
```



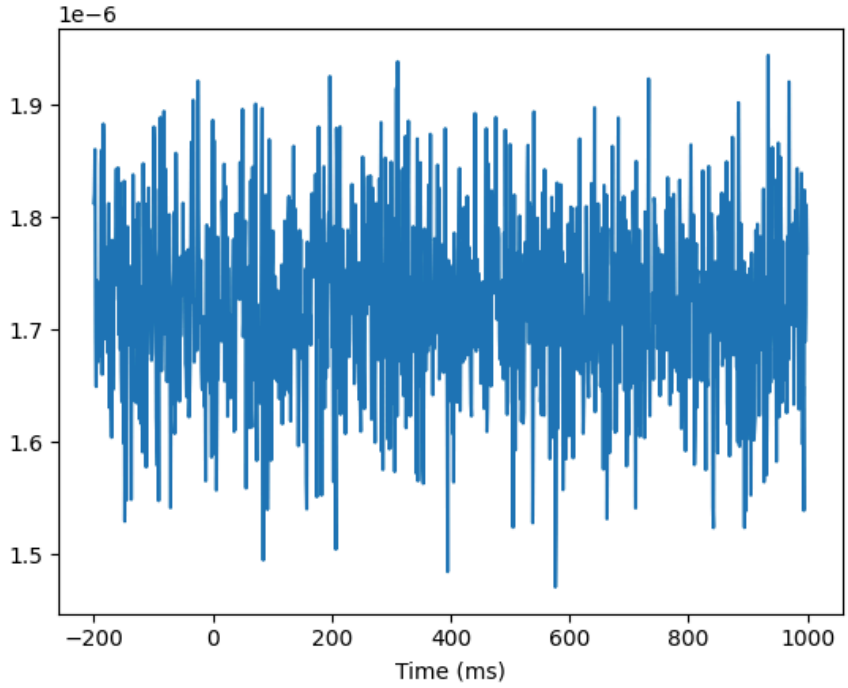
Calculate the variance (VAR) of ERP of 100 subjects

In [26]:
```
data_var = np.var(data, axis=0)
plt.plot(times, data_var)
plt.xlabel('Time (ms)')
plt.show()
```



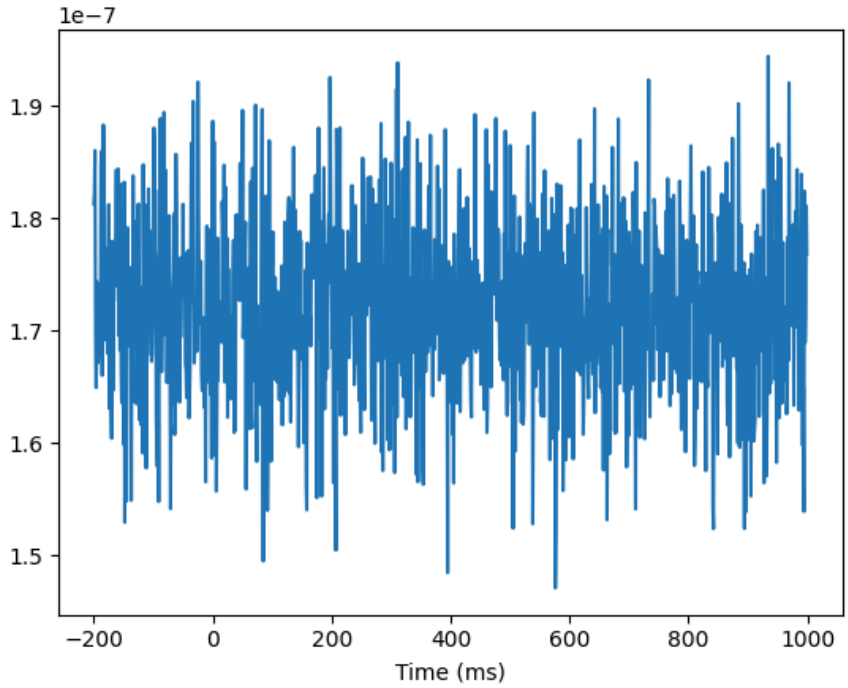
Calculate the standard deviation (STD) of ERP of 100 subjects

In [27]:
```
data_std = np.std(data, axis=0)
plt.plot(times, data_std)
plt.xlabel('Time (ms)')
plt.show()
```



Calculate the standard error (SEM) of ERP of 100 subjects

In [28]:
```
n_subjects = 100
data_sem = np.std(data, axis=0, ddof=0)/np.sqrt(n_subjects)
plt.plot(times, data_sem)
plt.xlabel('Time (ms)')
plt.show()
```



Inferential Statistics

In EEG research, besides calculating the mean and dispersion of EEG data, it often involves making some inferences about population characteristics.
Without going into basic concepts again, we only introduce commonly used significance testing methods below.

Testing on a Single Group

When it's necessary to test whether the data under a certain condition significantly differ from a specific value, tests are commonly used.

Here, like the previous example, we randomly generate data of 1200 time points for 20 subjects for subsequent examples.

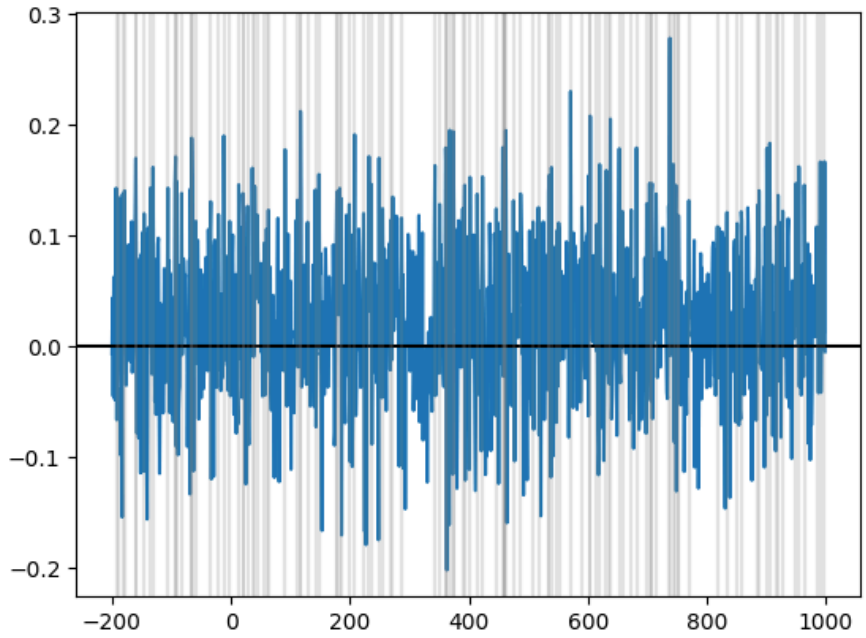
In [29]:
```
# Generate 'pseudo' EEG data with a shape of [20, 1200],
# with random values ranging from -0.1 to 0.4.
data = np.random.uniform(low=-0.5, high=0.55, size=(20, 1200))
```

Parametric testing method: One-sample t-test (uncorrected)

Conduct statistical tests assuming the data values are greater than 0 at each time point.

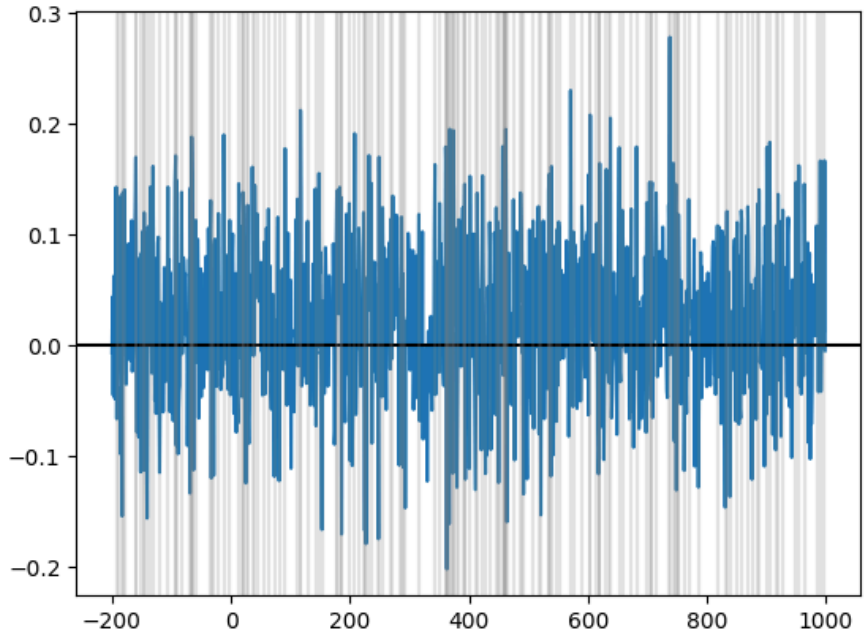
In [30]:
```
# One-sample t-test
t_vals, p_vals = ttest_1samp(data, 0, axis=0, alternative='greater')
# Print the shape of p-values: [1200] corresponding to 1200 time-points
print(np.shape(p_vals))
# Visualize the results after statistical testing
# Shadowed vertical lines represent significant time-points
plt.plot(times, np.average(data, axis=0))
plt.axhline(y=0, color='black')
for i, p_val in enumerate(p_vals):
    if p_val < 0.05:
        plt.axvline(x=times[i], color='grey', alpha=0.2)
plt.show()
```

```
(1200,)
```



Non-parametric testing method: Permutation test (uncorrected)

In [31]:
```
# The permutation_test() function in the stuff module of NeuroRA
# can perform permutation tests between two samples
# Generate a zero vector of shape [20]
zeros = np.zeros([20])
# Initialize a p_vals vector to store the calculated p-values
p_vals = np.zeros([1200])
# Calculate the p-value for each time point
for t in range(1200):
    p_vals[t] = permutation_test(data[:, t], zeros)
# Visualize the results after statistical testing
plt.plot(times, np.average(data, axis=0))
plt.axhline(y=0, color='black')
for i, p_val in enumerate(p_vals):
    if p_val < 0.05:
        plt.axvline(x=times[i], color='grey', alpha=0.2)
plt.show()
```

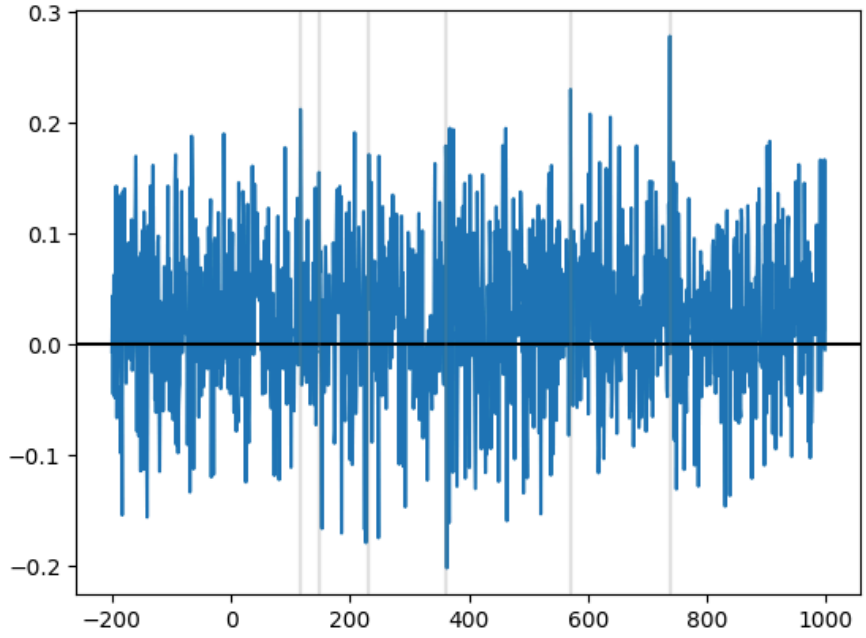


However, if conducting exploratory analysis to find significant differences at multiple time points, it's necessary to correct the results to avoid false positives due to multiple comparisons.
Common methods include:

- Bonferroni correction for controlling the familywise error rate (FWER)
- False discovery rate (FDR) correction
- Cluster-based permutation test

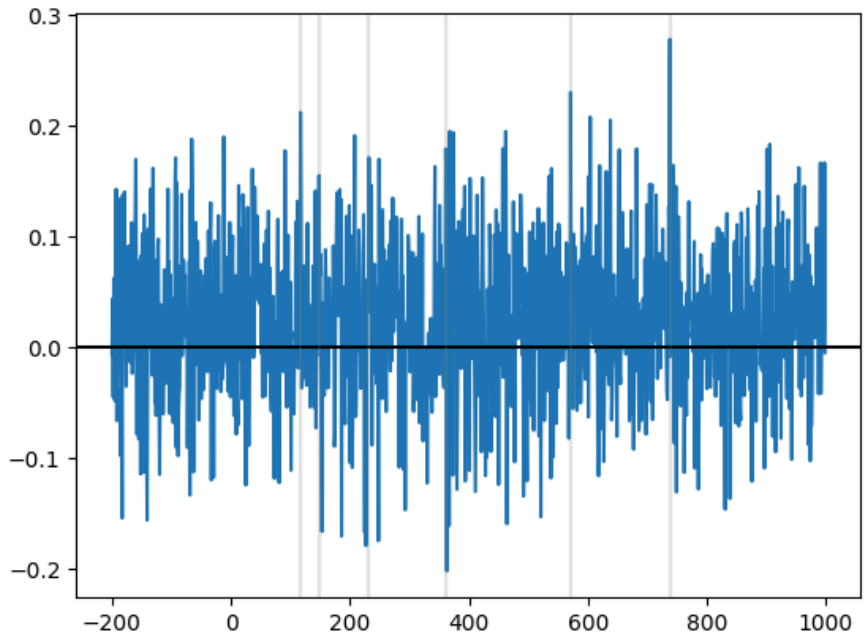
Bonferroni (FWER) correlation

In [32]:
```
# Bonferroni correction simply involves
# multiplying the p-values by the number of statistical tests conducted
# Here, this means multiplying by the number of time-points
p_bf_corrected_vals = p_vals*len(times)
# Visualize the results after statistical testing
plt.plot(times, np.average(data, axis=0))
plt.axhline(y=0, color='black')
for i, p_val in enumerate(p_bf_corrected_vals):
    if p_val < 0.05:
        plt.axvline(x=times[i], color='grey', alpha=0.2)
plt.show()
```



FDR correlation

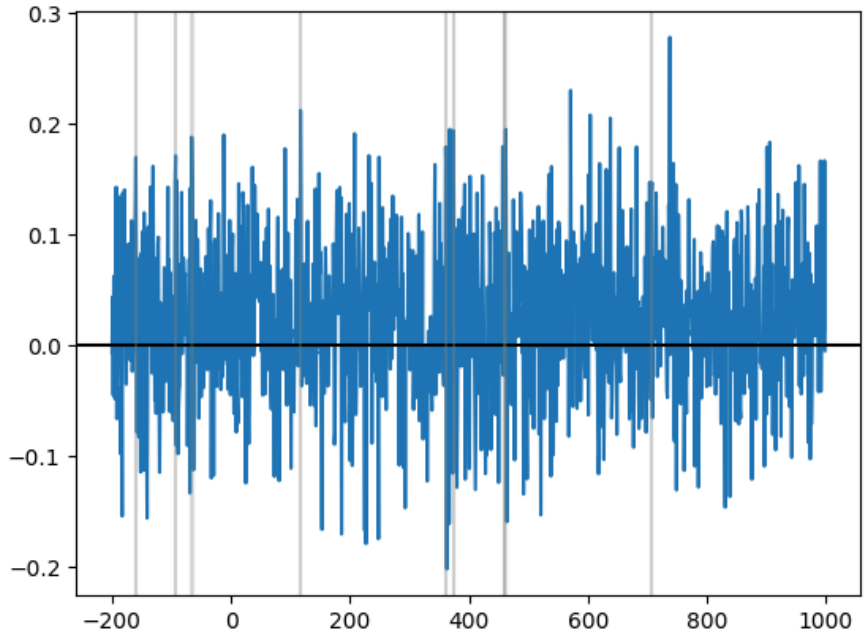
In [33]:
```
# FDR correction can be implemented using the fdr_correction() function
# in the stats module of MNE
# The first return value is a boolean array indicating whether the correction passed
# (True means it remains significant after correction),
# and the second return value is the corrected p-values
rejects, p_fdr_corrected_vals = fdr_correction(p_vals, alpha=0.05)
# Visualize the results after statistical testing
plt.plot(times, np.average(data, axis=0))
plt.axhline(y=0, color='black')
for i, p_val in enumerate(p_fdr_corrected_vals):
    if p_val < 0.05:
        plt.axvline(x=times[i], color='grey', alpha=0.2)
plt.show()
```



Cluster-based permutation test

In [34]:
```
# Here is the one-sample one-sided cluster-based permutation test for time series (1D) data
# It can be performed using the clusterbased_permutation_1d_1samp_1sided() function
# in the stuff module of NeuroRA
# This function requires the data for statistical analysis (i.e., the data here) as input,
# and outputs a matrix indicating whether the results are significant after correction
# (1 for significant points)
# Here, we first use a p<0.05 threshold to select clusters,
# with the parameter set to p_threshold=0.05
# Then, we use a p<0.05 threshold for cluster-based correction,
# with the parameter set to clusterp_threshold=0.05
rejects = clusterbased_permutation_1d_1samp_1sided(data, level=0,
                                                   p_threshold=0.05,
                                                   clusterp_threshold=0.05)
# Visualize the results after statistical testing
plt.plot(times, np.average(data, axis=0))
plt.axhline(y=0, color='black')
for i, reject in enumerate(rejects):
    if reject == 1:
        plt.axvline(x=times[i], color='grey', alpha=0.2)
plt.show()
```

```
Permutation test
Calculating: [================================================================================
==============] 100.00%
Cluster-based permutation test finished!
```



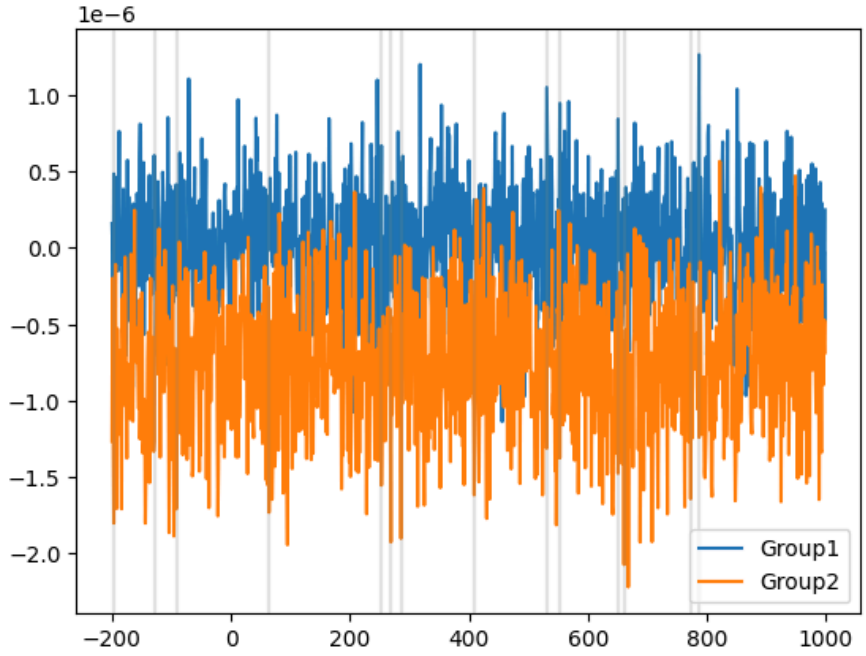
Comparing Significant Differences between Two Conditions/Groups

When it's necessary to compare whether there is a significant difference in brain activity between two conditions, independent samples t-tests (for between-subject variables), paired samples t-tests (for within-subject variables), or permutation tests can be used.

Parametric testing method: Independent samples t-test (FDR correction)

Here, we randomly generate two sets of 'pseudo' EEG data,
assuming they respectively represent data from a healthy control group (data1) and a patient group (data2).
The normal group has 20 subjects, the patient group has 18 subjects, and the epoch length remains 1200ms (1200 time points, from -200ms to 1000ms).

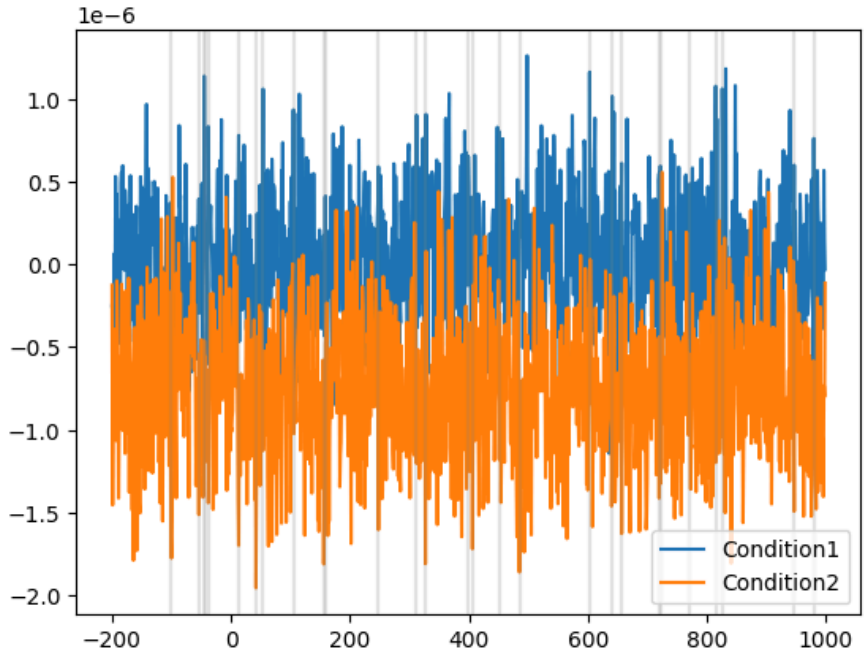
In [35]:
```
# Generate random data
data1 = np.random.uniform(low=-3e-6, high=3e-6, size=(20, 1200))
data2 = np.random.uniform(low=-4e-6, high=2.5e-6, size=(18, 1200))
# Independent samples t-test
t_vals, p_vals = ttest_ind(data1, data2, axis=0)
# FDR correction
rejects, p_fdr_corrected = fdr_correction(p_vals, alpha=0.05)
# Visualize the results after statistical testing
plt.plot(times, np.average(data1, axis=0), label='Group1')
plt.plot(times, np.average(data2, axis=0), label='Group2')
for i, reject in enumerate(rejects):
    if reject == True:
        plt.axvline(x=times[i], color='grey', alpha=0.2)
plt.legend()
plt.show()
```



Parametric testing method: Paired samples t-test (FDR correction)

Here, we randomly generate two sets of 'pseudo' EEG data,
assuming they represent the same group of subjects under two different conditions (condition 1 corresponds to data1, condition 2 corresponds to data2).
The number of subjects is 20, and the epoch length remains 1200ms (1200 time points, from -200ms to 1000ms).

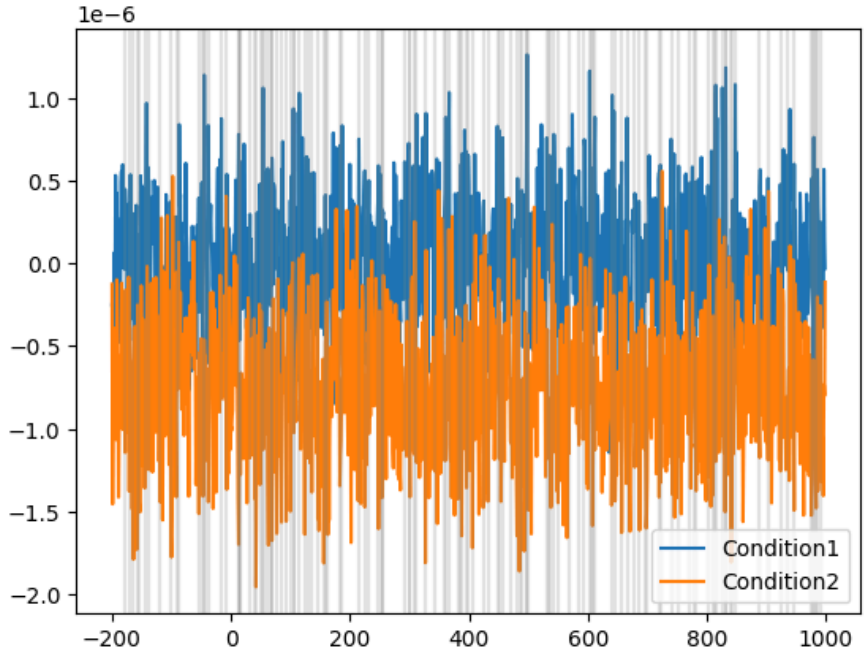
In [36]:
```
# Generate random data
data1 = np.random.uniform(low=-3e-6, high=3e-6, size=(20, 1200))
data2 = np.random.uniform(low=-4e-6, high=2.5e-6, size=(20, 1200))
# Paired samples t-test
t_vals, p_vals = ttest_rel(data1, data2, axis=0)
# FDR correction
rejects, p_fdr_corrected = fdr_correction(p_vals, alpha=0.05)
# Visualize the results after statistical testing
plt.plot(times, np.average(data1, axis=0), label='Condition1')
plt.plot(times, np.average(data2, axis=0), label='Condition2')
for i, reject in enumerate(rejects):
    if reject == True:
        plt.axvline(x=times[i], color='grey', alpha=0.2)
plt.legend()
plt.show()
```



Non-parametric testing method: Permutation test (FDR Correction)

Using the 'pseudo' data example from the paired samples t-test above.

In [37]:
```
# Initialize a p_vals vector to store the calculated p-values
p_vals = np.zeros([1200])
# Calculate p-values for each time-point
for t in range(1200):
    p_vals[t] = permutation_test(data1[:, t], data2[:, t])
# FDR correlation
rejects, p_fdr_corrected = fdr_correction(p_vals, alpha=0.05)
# Visualize the results after statistical testing
plt.plot(times, np.average(data1, axis=0), label='Condition1')
plt.plot(times, np.average(data2, axis=0), label='Condition2')
for i, reject in enumerate(rejects):
    if reject == 1:
        plt.axvline(x=times[i], color='grey', alpha=0.2)
plt.legend()
plt.show()
```

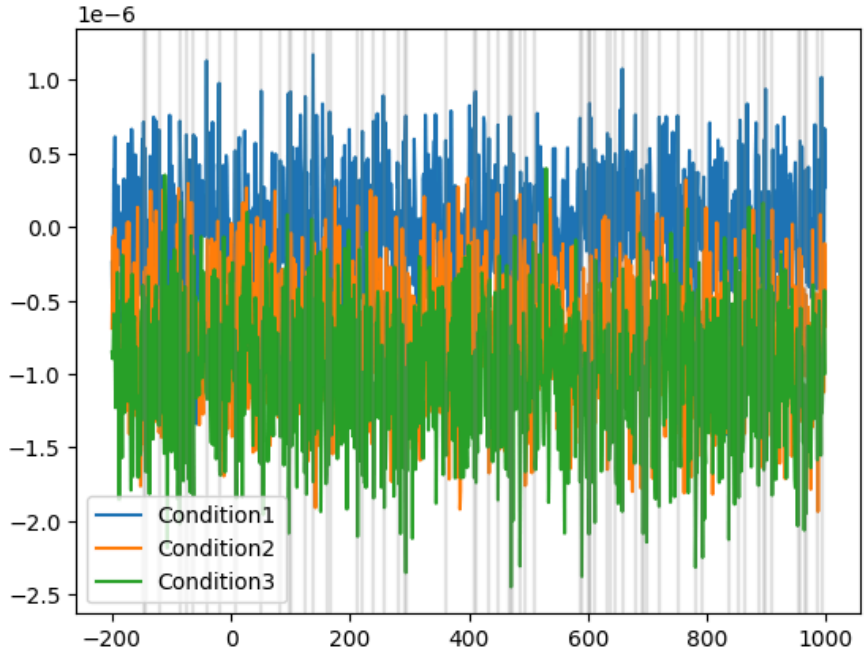


Comparing Significant Differences between Multiple Conditions/Groups

Parametric testing method: One-way ANOVA (FDR correction)

Here, similar to the previous example, we generate 'pseudo' EEG data for the same group of subjects under three different experimental conditions (corresponding to data1, data2, and data3).
The number of subjects is 20, and the epoch length is 1200ms (1200 time points, from -200ms to 1000ms).

In [38]:
```
# Generate random data
data1 = np.random.uniform(low=-3e-6, high=3e-6, size=(20, 1200))
data2 = np.random.uniform(low=-4e-6, high=2.5e-6, size=(20, 1200))
data3 = np.random.uniform(low=-4.5e-6, high=2.5e-6, size=(20, 1200))
# one-way F-test
f_vals, p_vals = f_oneway(data1, data2, data3, axis=0)
# FDR correction
rejects, p_fdr_corrected = fdr_correction(p_vals, alpha=0.05)
# Visualize the results after statistical testing
plt.plot(times, np.average(data1, axis=0), label='Condition1')
plt.plot(times, np.average(data2, axis=0), label='Condition2')
plt.plot(times, np.average(data3, axis=0), label='Condition3')
for i, reject in enumerate(rejects):
    if reject == True:
        plt.axvline(x=times[i], color='grey', alpha=0.2)
plt.legend()
plt.show()
```



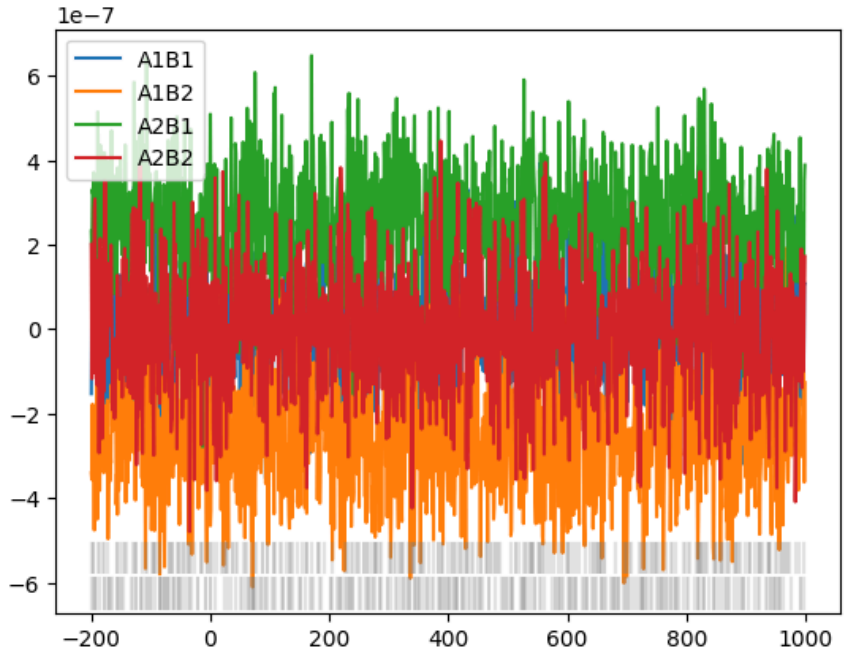
Multifactorial Design: Interaction and Main Effects of Two Factors

Parametric testing method: 2×2 repeated measures ANOVA (FDR correction)

Assuming there are two within-subject variables: A and B.
Each variable has two levels, making a total of 4 conditions.
Here, we randomly generate 'pseudo' EEG data for 20 subjects under these 4 conditions.
The epoch length is 1200ms (1200 time points, from -200ms to 1000ms).

In [39]:
```
# Generate random data
data_A1B1 = np.random.uniform(low=-3e-6, high=3e-6, size=(200, 1200))
data_A1B2 = np.random.uniform(low=-3.5e-6, high=3e-6, size=(200, 1200))
data_A2B1 = np.random.uniform(low=-3e-6, high=3.5e-6, size=(200, 1200))
data_A2B2 = np.random.uniform(low=-3.5e-6, high=3.5e-6, size=(200, 1200))
# Reshape the data
reshaped_A1B1 = data_A1B1.reshape(200, 1, 1200)
reshaped_A1B2 = data_A1B2.reshape(200, 1, 1200)
reshaped_A2B1 = data_A2B1.reshape(200, 1, 1200)
reshaped_A2B2 = data_A2B2.reshape(200, 1, 1200)
# Merge the data in the order of the two factors (A1B1、A1B2、A2B1、A2B2)
data_combine = np.concatenate((reshaped_A1B1, reshaped_A1B2,
                               reshaped_A2B1, reshaped_A2B2), axis=1)
# Set factor levels
factor_levels = [2, 2]
# Perform a 2×2 ANOVA using MNE's f_mway_rm function
# Main effect of factor A
f_main_A, p_main_A = f_mway_rm(data_combine, factor_levels, effects='A')
# Main effect of factor B
f_main_B, p_main_B = f_mway_rm(data_combine, factor_levels, effects='B')
# Interaction effect
f_inter, p_interaction = f_mway_rm(data_combine, factor_levels, effects='A:B')
# FDR correction
rejects_A, p_main_A = fdr_correction(p_main_A, alpha=0.05)
rejects_B, p_main_B = fdr_correction(p_main_B, alpha=0.05)
rejects_inter, p_interaction = fdr_correction(p_interaction, alpha=0.05)
# Visualize the results after statistical testing
# The three rows of gray vertical lines below the picture,
# from bottom to top, respectively represent the time points where the main effect of A,
# the main effect of B, and the interaction effect are significant.
plt.plot(times, np.average(data_A1B1, axis=0), label='A1B1')
plt.plot(times, np.average(data_A1B2, axis=0), label='A1B2')
plt.plot(times, np.average(data_A2B1, axis=0), label='A2B1')
plt.plot(times, np.average(data_A2B2, axis=0), label='A2B2')
for i in range(1200):
    if p_main_A[i] < 0.05:
        plt.axvline(x=times[i], ymin=0.01, ymax=0.06, color='grey', alpha=0.2)
    if p_main_B[i] < 0.05:
        plt.axvline(x=times[i], ymin=0.07, ymax=0.12, color='grey', alpha=0.2)
    if p_interaction[i] < 0.05:
        plt.axvline(x=times[i], ymin=0.13, ymax=0.18, color='grey', alpha=0.2)
```

```
plt.legend()
plt.show()
```



Chapter 3: Multiple-Subject Analysis

The section on multisubject analysis is divided into the following 3 parts：

- **Part 1: Batch Processing for Reading and Storing Demo Data**
- **Part 2: Event-Related Potential Analysis**
- **Part 3: Time-Frequency Analysis**

Download & Import Requried Python packages

In [1]:
```
! pip install gdown

import numpy as np
import os
import gdown
import zipfile
import h5py
import scipy.io as sio
import matplotlib.pyplot as plt
from scipy.stats import ttest_1samp, ttest_rel
from mne.stats import fdr_correction, f_mway_rm
from neurora.stuff import clusterbased_permutation_1d_1samp_1sided, \
                          permutation_test, \
                          clusterbased_permutation_2d_1samp_2sided, \
                          clusterbased_permutation_2d_2sided
from mne.time_frequency import tfr_array_morlet
```

```
Requirement already satisfied: gdown in /Users/zitonglu/anaconda3/lib/python3.11/site-packages (5.1.
0)
Requirement already satisfied: beautifulsoup4 in /Users/zitonglu/anaconda3/lib/python3.11/site-packa
ges (from gdown) (4.12.2)
Requirement already satisfied: filelock in /Users/zitonglu/anaconda3/lib/python3.11/site-packages (f
rom gdown) (3.9.0)
Requirement already satisfied: requests[socks] in /Users/zitonglu/anaconda3/lib/python3.11/site-pack
ages (from gdown) (2.31.0)
Requirement already satisfied: tqdm in /Users/zitonglu/anaconda3/lib/python3.11/site-packages (from
gdown) (4.65.0)
Requirement already satisfied: soupsieve>1.2 in /Users/zitonglu/anaconda3/lib/python3.11/site-packag
es (from beautifulsoup4->gdown) (2.4)
Requirement already satisfied: charset-normalizer<4,>=2 in /Users/zitonglu/anaconda3/lib/python3.11/
site-packages (from requests[socks]->gdown) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in /Users/zitonglu/anaconda3/lib/python3.11/site-package
s (from requests[socks]->gdown) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /Users/zitonglu/anaconda3/lib/python3.11/site-p
ackages (from requests[socks]->gdown) (1.26.16)
Requirement already satisfied: certifi>=2017.4.17 in /Users/zitonglu/anaconda3/lib/python3.11/site-p
ackages (from requests[socks]->gdown) (2023.7.22)
Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in /Users/zitonglu/anaconda3/lib/python3.11/si
te-packages (from requests[socks]->gdown) (1.7.1)
```

Part 1 Batch Processing for Reading and Storing Demo Data

Preprocessed Demo Data 1：

(You will use it in Part 2 and Part 3)

The original dataset is based on the article "*A multi-subject, multi-modal human neuroimaging dataset*" by Wakeman & Henson, published in Scientific Data in 2015. In this experiment, there are three types of faces: familiar faces, unfamiliar faces, and scrambled faces, with 150 images for each type, totaling 450 stimulus images. Participants wore an EEG cap to perform a simple perceptual task, which included an unpredictable stimulus phase of 800-1000ms, a delay of 1700ms, and an inter-trial interval (ITI) of 400-600ms. Each image was viewed twice, with 50% of the stimulus images being presented immediately after the first viewing, while the other 50% were presented after several other trials after the first viewing. Here, only the EEG data from the first eight subjects who viewed multiple familiar face images for the first time and then immediately viewed some of these images again in the next trial are extracted.

In [2]:
```
# Download Demo Data 1
```

```
data_dir = "data/"
if not os.path.exists(data_dir):
  os.makedirs(data_dir)

url = "https://drive.google.com/file/d/1hsPmIFod3c7ZR0YdyO8woqUfqO_3pZtR/view?usp=sharing"
filename = "demo_data1"
filepath = data_dir + filename + ".zip"

# Download the data
gdown.download(url=url, output=filepath, quiet=False, fuzzy=True)
print("Download completes!")
# unzip the data
with zipfile.ZipFile(filepath, 'r') as zip:
    zip.extractall(data_dir)
print("Unzip completes!")
```

```
Downloading...
From (original): https://drive.google.com/uc?id=1hsPmIFod3c7ZR0YdyO8woqUfqO_3pZtR
From (redirected): https://drive.google.com/uc?id=1hsPmIFod3c7ZR0YdyO8woqUfqO_3pZtR&confirm=t&uuid=6
4c004fa-0289-4fe3-9cc3-c66c63609f2e
To: /Users/zitonglu/Downloads/Python-EEG-Handbook-master/data/demo_data1.zip
100%|████████████████████████████████████████| 242M/242M [00:28<00:00, 8.43MB/s]
Download completes!
Unzip completes!
```

Taking sub1 as an example, 'sub1_first.mat' contains EEG data for the first viewing of familiar face images, and 'sub1_rep.mat' contains EEG data for the immediate second viewing of familiar faces. The number of trials in the former case is double that of the latter. The data has been preprocessed (0.1-30Hz filtering) and segmented. The data includes 74 channels (of which 70 are EEG channels, with channels 61, 62, 63, and 64 being eye movement channels) and a sampling rate of 250Hz. Each trial covers from 0.5 seconds before to 1.5 seconds after the stimulus presentation, with 500 time points per trial.

Batch Processing for Reading Demo Data 1 then Saving as a .h5 File

In [3]:

```
# Iterate over the data of 8 subjects
for sub in range(8):

    # Get the paths of .mat files for the two conditions of each subject
    subdata_first_path = 'data/demo_data1/sub' + str(sub + 1) + '_first.mat'
    subdata_rep_path = 'data/demo_data1/sub' + str(sub + 1) + '_rep.mat'

    # Extract data from the .mat files
    # subdata shape: [n_channels, n_times, n_trials]
    subdata_first = sio.loadmat(subdata_first_path)['data']
    subdata_rep = sio.loadmat(subdata_rep_path)['data']

    # Delete four EOG channels
    subdata_first = np.delete(np.array(subdata_first), [60, 61, 62, 63], axis=0)
    subdata_rep = np.delete(np.array(subdata_rep), [60, 61, 62, 63], axis=0)

    # Print the subject number and the shape of EEG matrices for two conditions:
    # first viewing of familiar faces and immediate repeated viewing
    print('sub' + str(sub + 1))
    print(subdata_first.shape)
    print(subdata_rep.shape)

    # Store the EEG data (in matrix form) for the two stimulus conditions of each subject
    # in an .h5 file with 'data' and 'data_rep' as Keys
    f = h5py.File('data/demo_data1/sub' + str(sub + 1) + '.h5', 'w')
    f.create_dataset('data_first', data=subdata_first)
    f.create_dataset('data_rep', data=subdata_rep)
    f.close()
```

```
sub1
(70, 500, 150)
(70, 500, 78)
sub2
(70, 500, 150)
(70, 500, 79)
sub3
(70, 500, 150)
(70, 500, 74)
sub4
(70, 500, 150)
(70, 500, 68)
sub5
(70, 500, 150)
(70, 500, 67)
sub6
(70, 500, 150)
(70, 500, 70)
sub7
(70, 500, 150)
(70, 500, 77)
sub8
(70, 500, 150)
(70, 500, 70)
```

Part 2 Event-Related Potential Analysis

Here, we will use Demo Data 1 as an example.

We will visualize the ERP results and conduct statistical analyses for two conditions in the experiment, the first viewing of familiar faces and the immediate repeated viewing of familiar faces, respectively.

Read Data and Average Epoches

In the example below, the ERP from the 50th channel is taken as an example, selecting the time-window from -200ms to 1000ms.

In [4]:
```python
# Initialize two variables to store the ERP for each subject under the two conditions
# 8 - n_subjects, 300 - n_times (from -200ms to 1000ms)
erp_first = np.zeros([8, 300])
erp_rep = np.zeros([8, 300])

# Iterate over each subject
for sub in range(8):

    # Read the data for this subject
    with h5py.File('data/demo_data1/sub' + str(sub + 1) + '.h5', 'r') as f:
        subdata_first = np.array(f['data_first'])
        subdata_rep = np.array(f['data_rep'])
        f.close()
    # The shape of subdata and subdata_rep is [n_channels, n_times, n_trials]

    # Select data from the 50th electrode, average across trials, and take -200ms to 1000ms
    erp_first[sub] = np.average(subdata_first[49], axis=1)[75:375]
    erp_rep[sub] = np.average(subdata_rep[49], axis=1)[75:375]

    # Baseline correction, using the baseline from -100 to 0ms
    erp_first[sub] = erp_first[sub] - np.average(erp_first[sub, 25:50])
    erp_rep[sub] = erp_rep[sub] - np.average(erp_rep[sub, 25:50])
```

Statistical Analyses and Visualization

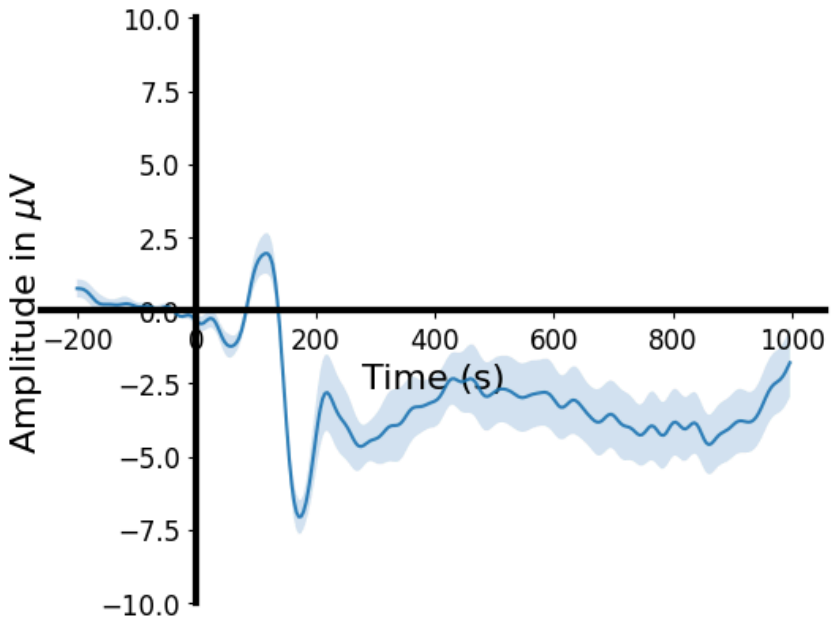
Define a function to plot ERP results - plot_erp_results()

In [5]:
```python
def plot_erp_results(erp, times, ylim=[-10, 10], labelpad=0):

    """
    parameters:
        erp: a matrix with the shape [n_subs, n_times] corresponding to all subjects' ERPs
        times: a array with the shape [n_times]
        corresponding to the time-points and the range of x-axis
        ylim: the lims of y-axis, default [-10, 10]
        labelpad : a int value, spacing in points from the axes, default 0
    """
```

```
    n_subjects = np.shape(erp)[0]

    # averaging the ERPs
    avg = np.average(erp, axis=0)
    # calcualte the SEM for each time-point
    err = np.std(erp, axis=0, ddof=0)/np.sqrt(n_subjects)

    ax = plt.gca()
    ax.spines["top"].set_visible(False)
    ax.spines["right"].set_visible(False)
    ax.spines["left"].set_linewidth(3)
    ax.spines["left"].set_position(("data", 0))
    ax.spines["bottom"].set_linewidth(3)
    ax.spines['bottom'].set_position(('data', 0))

    # plot the ERP
    plt.fill_between(times, avg+err, avg-err, alpha=0.2)
    plt.plot(times, avg, alpha=0.9)
    plt.xticks(fontsize=12)
    plt.yticks(fontsize=12)
    plt.ylabel(r'Amplitude in $\mu$V', fontsize=16, labelpad=labelpad)
    plt.xlabel('Time (s)', fontsize=16)
    plt.ylim(ylim[0], ylim[1])
    plt.show()
```

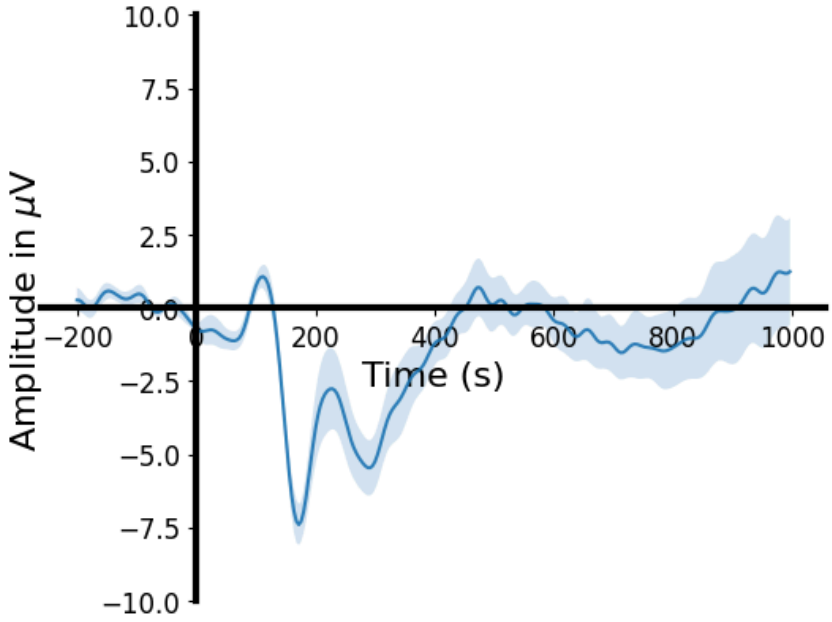
Plot the ERP under the condition of the first viewing of familiar faces

In [6]:
```
times = np.arange(-200, 1000, 4)
plot_erp_results(erp_first, times, labelpad=25)
```



Plot the ERP under the condition of the immediate repeated viewing of familiar faces

In [7]:
```
plot_erp_results(erp_rep, times, labelpad=25)
```



Define a function to visualize joint ERPs for two conditions together - plot_erp_2cons_results()

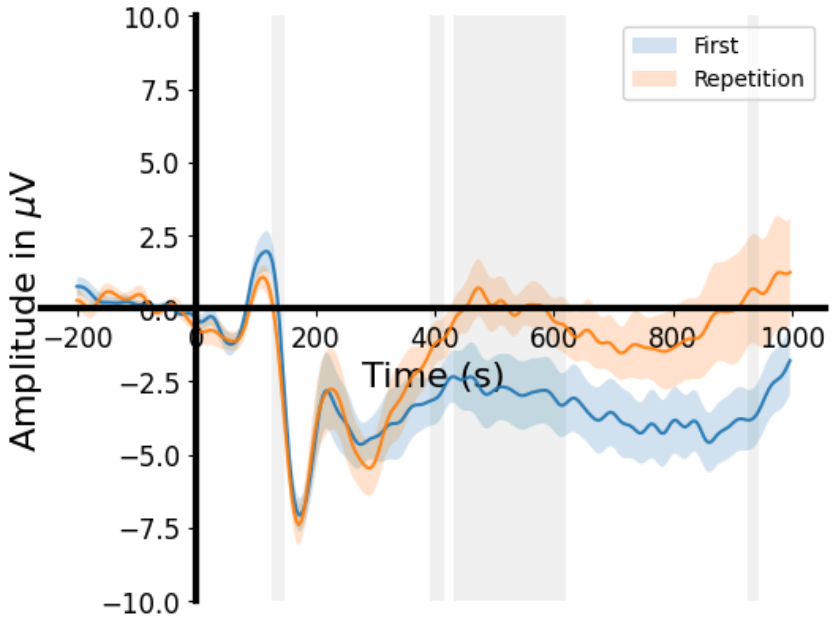
In [8]:

```
def plot_erp_2cons_results(erp1, erp2, times, con_labels=['Condition1', 'Condition2'], ylim=[-10, 10
                      p_threshold=0.05, labelpad=0):

    """
    parameters:
        erp1: a matrix with the shape [n_subs, n_times]
        corresponding to all subjects' ERPs under condition 1
        erp2: a matrix with the shape [n_subs, n_times]
        corresponding to all subjects' ERPs under condition 2
        times: an array with the shape [n_times]
        corresponding to the time-points and the range of x-axis
        con_labels: a list or array with the labels of two conditions,
        default ['Condition 1', 'Condition 2']
        ylim: the lims of y-axis, default [-10, 10]
        p_threshold : a float value, default is 0.05,
        representing the threshold of p-value
        labelpad : a int value, spacing in points from the axes, default 0
    """

    n_subjects = np.shape(erp1)[0]

    # averaging the ERPs
    avg1 = np.average(erp1, axis=0)
    avg2 = np.average(erp2, axis=0)
    # calcualte the SEM for each time-point
    err1 = np.std(erp1, axis=0, ddof=0)/np.sqrt(n_subjects)
    err2 = np.std(erp2, axis=0, ddof=0)/np.sqrt(n_subjects)

    # statistical analysis
    t_vals, p_vals = ttest_rel(erp1, erp2, axis=0)
    # FDR-correction
    # rejects, p_fdr_corrected = fdr_correction(p_vals, alpha=p_threshold)

    ax = plt.gca()
    ax.spines["top"].set_visible(False)
    ax.spines["right"].set_visible(False)
    ax.spines["left"].set_linewidth(3)
    ax.spines["left"].set_position(("data", 0))
    ax.spines["bottom"].set_linewidth(3)
    ax.spines['bottom'].set_position(('data', 0))

    # highlight the significant time-windows
    tstep = (times[-1]-times[0])/len(times)
    for i, p_val in enumerate(p_vals):
        if p_val < 0.05:
            plt.fill_between([times[i], times[i]+tstep], [ylim[1]], [ylim[0]],
                             facecolor='gray', alpha=0.1)

    # plot the ERPs with statistical results
    plt.fill_between(times, avg1+err1, avg1-err1, alpha=0.2, label=con_labels[0])
    plt.fill_between(times, avg2+err2, avg2-err2, alpha=0.2, label=con_labels[1])
    plt.plot(times, avg1, alpha=0.9)
```

```
    plt.plot(times, avg2, alpha=0.9)
    plt.xticks(fontsize=12)
    plt.yticks(fontsize=12)
    plt.ylim(ylim[0], ylim[1])
    plt.ylabel(r'Amplitude in $\mu$V', fontsize=16, labelpad=labelpad)
    plt.xlabel('Time (s)', fontsize=16)

    plt.legend()
    plt.show()
```

Plot the joint ERP results under two conditions

In [9]:
```
plot_erp_2cons_results(erp_first, erp_rep, times, con_labels=['First', 'Repetition'],
                       p_threshold=0.05, labelpad=25)
```



Part 3 Time-Frequency Analysis

We will visualize the time-frequency results and conduct statistical analyses for two conditions in the experiment, the first viewing of familiar faces and the immediate repeated viewing of familiar faces, respectively.

Read Data and Conduct the Time-Frequency Analysis

In [10]:
```
# Initialize two variables to store the time-frequency analysis results
# for each subject under two conditions
# 8 - n_subjects, 70 - n_channels, 14 - n_freqs, 500 - n_times
tfr_first = np.zeros([8, 70, 14, 500])
tfr_rep = np.zeros([8, 70, 14, 500])

# Iterate over each subject
for sub in range(8):

    # Read the data for this subject
    with h5py.File('data/demo_data1/sub' + str(sub + 1) + '.h5', 'r') as f:
        subdata_first = np.array(f['data_first'])
        subdata_rep = np.array(f['data_rep'])
        f.close()
    # Transform from [n_channels, n_times, n_trials] to [n_trials, n_channels, n_times]
    subdata_first = np.transpose(subdata_first, (2, 0, 1))
    subdata_rep = np.transpose(subdata_rep, (2, 0, 1))

    # Set some parameters for time-frequency analysis
    # Select frequency band from 4-30Hz
    freqs = np.arange(4, 32, 2)
    n_cycles = freqs / 2.
    # Time-frequency analysis
    # Use the tfr_array_morlet() function from MNE's time_frequency module
    # The input is an array of shape [n_epochs, n_channels, n_times]
    # Followed by the sampling rate of the data, the frequencies to be computed,
    # the number of cycles, and the output data type
    subtfr_first = tfr_array_morlet(subdata_first, 250, freqs, n_cycles, output='power')
```

```
    subtfr_rep = tfr_array_morlet(subdata_rep, 250, freqs, n_cycles, output='power')
    # The returned tfr shape is [n_trials, n_channels, n_freqs, n_times]
    # Average across the trials and channels dimensions
    tfr_first[sub] = np.average(subtfr_first, axis=0)
    tfr_rep[sub] = np.average(subtfr_rep, axis=0)
    # Baseline correction, using the 'logratio' method,
    # i.e., divide by the baseline mean and take the log
    # The baseline is taken from -100 to 0ms
    for chl in range(70):
        for freq in range(len(freqs)):
            tfr_first[sub, chl, freq] = 10 * np.log10(tfr_first[sub, chl, freq] /
                                    np.average(tfr_first[sub, chl, freq, 100:125]))
            tfr_rep[sub, chl, freq] = 10 * np.log10(tfr_rep[sub, chl, freq] /
                                        np.average(tfr_rep[sub, chl, freq, 100:125]))
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.2s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.3s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.4s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  70 out of  70 | elapsed:    6.4s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.2s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.2s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  70 out of  70 | elapsed:    3.3s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.2s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.3s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.4s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  70 out of  70 | elapsed:    6.4s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.2s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.3s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  70 out of  70 | elapsed:    3.6s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.2s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.3s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.4s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  70 out of  70 | elapsed:    6.3s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.2s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  70 out of  70 | elapsed:    3.1s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.2s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.3s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.4s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  70 out of  70 | elapsed:    6.2s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.2s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  70 out of  70 | elapsed:    2.8s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.2s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.3s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.4s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  70 out of  70 | elapsed:    6.6s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.2s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  70 out of  70 | elapsed:    2.8s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.2s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.3s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.4s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  70 out of  70 | elapsed:    6.5s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.2s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  70 out of  70 | elapsed:    2.9s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.2s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.3s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.4s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  70 out of  70 | elapsed:    6.2s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.1s remaining:    0.0s
```

```
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.2s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  70 out of  70 | elapsed:    3.2s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.2s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.3s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.4s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  70 out of  70 | elapsed:    6.3s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.2s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  70 out of  70 | elapsed:    2.9s finished
```

Here, the time-frequency analysis results from -200ms to 1000ms for the 50th channel under both conditions will be used as an example for subsequent steps.

In [11]:
```
tfr_first_No50 = tfr_first[:, 49, :, 75:375]
tfr_rep_No50 = tfr_rep[:, 49, :, 75:375]
```

Statistical Analyses and Visualization

Define a function to plot time-frequency analysis for a single condition - plot_tfr_results().

This section describes functionalities for statistical analysis and visualization, and the results undergo a cluster-based permutation test.

In [12]:
```
def plot_tfr_results(tfr, freqs, times, p=0.01, clusterp=0.05, clim=[-4, 4]):

    """
    Parameters:
        tfr : A matrix with shape [n_subs, n_freqs, n_times],
        representing the results of the time-frequency analysis
        freqs : An array with shape [n_freqs],
        representing the frequencies for the time-frequency analysis
        (corresponding to the frequency range and points on the y-axis)
        times : An array with shape [n_times],
        representing the time points for the time-frequency analysis
        (corresponding to the time range and points on the x-axis)
        p : A floating-point number, default is 0.01, representing the threshold for the p-value
        clusterp : A floating-point number, default is 0.05,
        representing the threshold for the cluster-level p-value
        clim : A list or array, [minimum, maximum], default is [-4, 4],
        representing the upper and lower boundaries of the color bar
    """

    n_freqs = len(freqs)
    n_times = len(times)

    # Statistical analysis
    # Note: Since a cluster-based permutation test is performed, it requires a longer runtime
    # The clusterbased_permutation_2d_1samp_2sided() function from stuff module in NeuroRA is used
    # The returned stats_results is a matrix with shape [n_freqs, n_times]
    # In this matrix, non-significant points have a value of 0,
    # significant points greater than 0 have a value of 1,
    # and significant points less than 0 have a value of -1
    # Here, iter is set to 100 for faster demonstration purposes, but 1000 is recommended
    stats_results = clusterbased_permutation_2d_1samp_2sided(tfr, 0,
                                                             p_threshold=p,
                                                             clusterp_threshold=clusterp,
                                                             iter=100)

    # Visualization of time-frequency analysis results
    fig, ax = plt.subplots(1, 1)
    # Outline the significant areas
    padsats_results = np.zeros([n_freqs + 2, n_times + 2])
    padsats_results[1:n_freqs+1, 1:n_times + 1] = stats_results
    x = np.concatenate(([times[0]-1], times, [times[-1]+1]))
    y = np.concatenate(([freqs[0]-1], freqs, [freqs[-1]+1]))
    X, Y = np.meshgrid(x, y)
    ax.contour(X, Y, padsats_results, [0.5], colors="red", alpha=0.9,
               linewidths=2, linestyles="dashed")
    ax.contour(X, Y, padsats_results, [-0.5], colors="blue", alpha=0.9,
               linewidths=2, linestyles="dashed")
    # Draw the heatmap of time-frequency results
    im = ax.imshow(np.average(tfr, axis=0), cmap='RdBu_r', origin='lower',
```

```
                    extent=[times[0], times[-1], freqs[0], freqs[-1]], clim=clim)
    ax.set_aspect('auto')
    cbar = fig.colorbar(im)
    cbar.set_label('dB', fontsize=12)
    ax.set_xlabel('Time (ms)', fontsize=16)
    ax.set_ylabel('Frequency (Hz)', fontsize=16)
    plt.show()
```

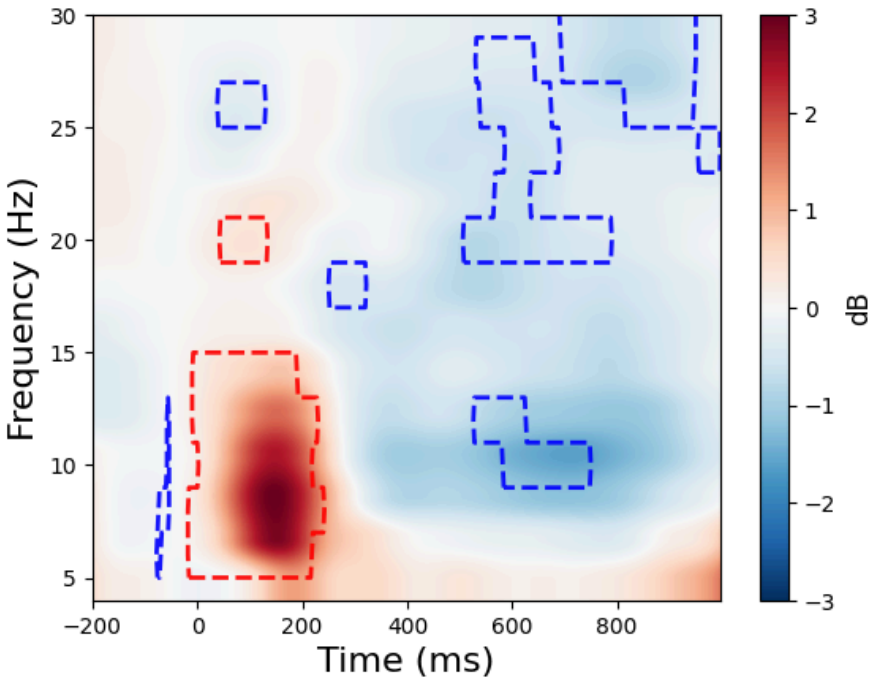
In [13]:
```
print(tfr_first.shape)
```

```
(8, 70, 14, 500)
```

Plot the time-frequency result under the condition of the first viewing of familiar faces

In [14]:
```
freqs = np.arange(4, 32, 2)
times = np.arange(-200, 1000, 4)
plot_tfr_results(tfr_first_No50, freqs, times, p=0.05, clusterp=0.05, clim=[-3, 3])
```

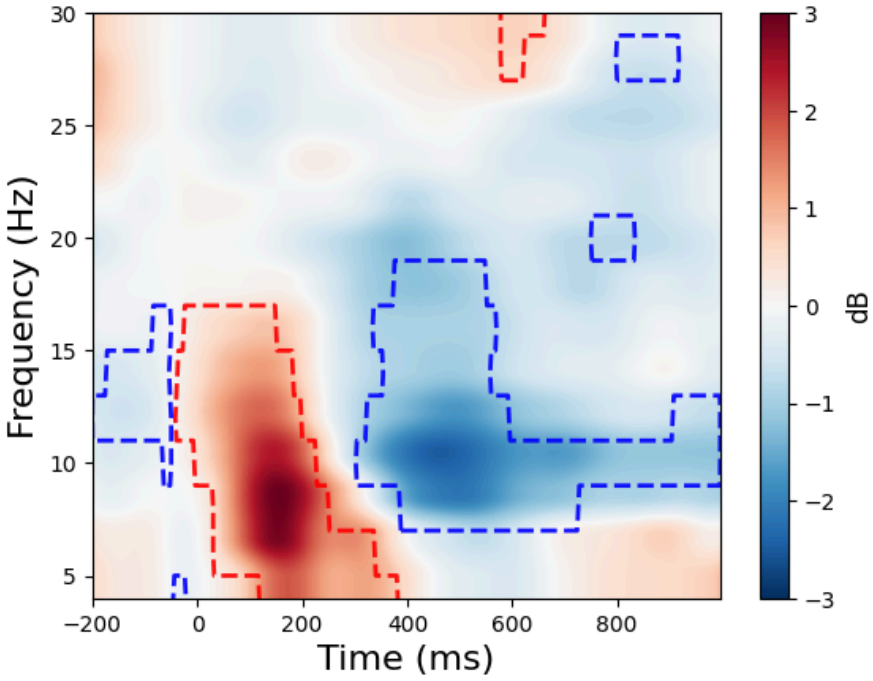
```
Permutation test

Side 1 begin:
Calculating: [============================================================================================================] 100.00%
Side 1 finished!

Side 2 begin:

Calculating: [============================================================================================================] 100.00%
Side 2 finished!

Cluster-based permutation test finished!
```



Plot the time-frequency result under the condition of the immediate repeated viewing of familiar faces

In [15]:
```
plot_tfr_results(tfr_rep_No50, freqs, times, p=0.05, clusterp=0.05, clim=[-3, 3])
```

```
Permutation test

Side 1 begin:
Calculating: [============================================================================================
==============] 100.00%
Side 1 finished!

Side 2 begin:

Calculating: [============================================================================================
==============] 100.00%
Side 2 finished!

Cluster-based permutation test finished!
```



Define a function to plot the difference of time-frequency results between two conditions - plot_tfr_diff_results()

We calculate the difference by tfr1 - tfr2

This section describes functionalities for statistical analysis and visualization, and the results undergo a cluster-based permutation test.

In [16]:

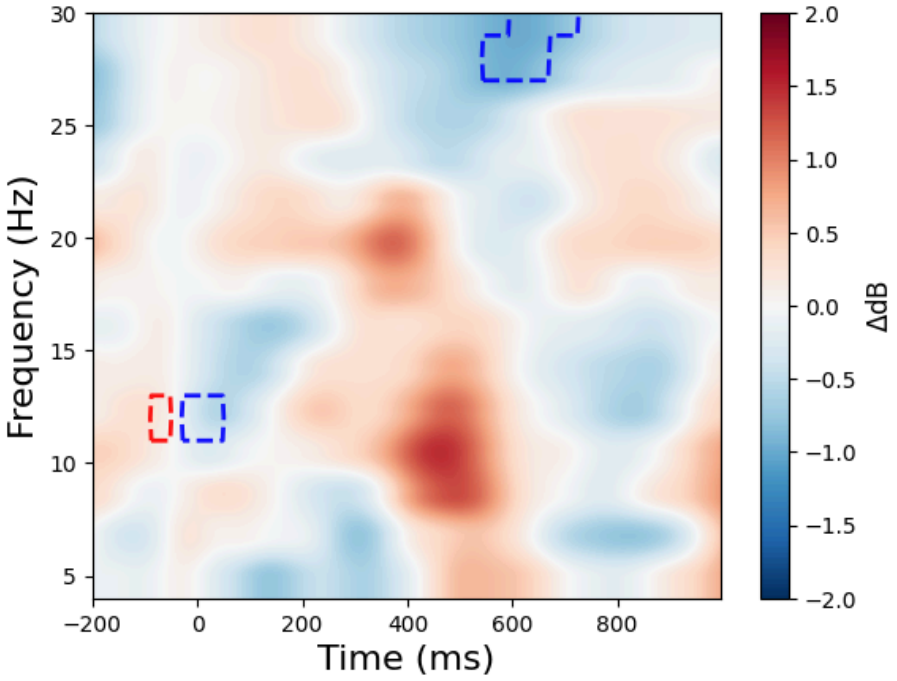
```python
def plot_tfr_diff_results(tfr1, tfr2, freqs, times, p=0.01, clusterp=0.05,
                          clim=[-2, 2]):

    """
    Parameters:
        tfr1 : A matrix with shape [n_subs, n_freqs, n_times],
        representing the time-frequency analysis results under condition 1
        tfr2 : A matrix with shape [n_subs, n_freqs, n_times],
        representing the time-frequency analysis results under condition 2
        freqs : An array with shape [n_freqs],
        representing the frequencies for the time-frequency analysis
        (corresponding to the frequency range and points on the y-axis)
        times : An array with shape [n_times],
        representing the time points for the time-frequency analysis
        (corresponding to the time range and points on the x-axis)
        p : A floating-point number, default is 0.01,
        representing the threshold for the p-value
        clusterp : A floating-point number, default is 0.05,
        representing the threshold for the cluster-level p-value
        clim : A list or array, [minimum, maximum], default is [-2, 2],
        representing the upper and lower boundaries of the color bar
    """

    n_freqs = len(freqs)
    n_times = len(times)

    # Statistical analysis
    # Note: Since a cluster-based permutation test is performed, it requires a longer runtime
    # Here, the clusterbased_permutation_2d_2sided() function from NeuroRA's stuff module is used
```

```
    # The returned stats_results is a matrix with shape [n_freqs, n_times]
    # In this matrix, non-significant points have a value of 0,
    # points where condition 1 is significantly greater than condition 2 have a value of 1,
    # and points where condition 1 is significantly less than condition 2 have a value of -1
    # Here, iter is set to 100 for faster demonstration purposes, but 1000 is recommended
    stats_results = clusterbased_permutation_2d_2sided(tfr1, tfr2,
                                                       p_threshold=p,
                                                       clusterp_threshold=clusterp,
                                                       iter=100)

    # Calculate the difference in tfr
    tfr_diff = tfr1 - tfr2

    # Visualization of time-frequency analysis results
    fig, ax = plt.subplots(1, 1)
    # Outline the significant areas
    padsats_results = np.zeros([n_freqs + 2, n_times + 2])
    padsats_results[1:n_freqs + 1, 1:n_times + 1] = stats_results
    x = np.concatenate(([times[0]-1], times, [times[-1]+1]))
    y = np.concatenate(([freqs[0]-1], freqs, [freqs[-1]+1]))
    X, Y = np.meshgrid(x, y)
    ax.contour(X, Y, padsats_results, [0.5], colors="red", alpha=0.9,
               linewidths=2, linestyles="dashed")
    ax.contour(X, Y, padsats_results, [-0.5], colors="blue", alpha=0.9,
               linewidths=2, linestyles="dashed")
    # Draw the heatmap of the time-frequency result differences
    im = ax.imshow(np.average(tfr_diff, axis=0), cmap='RdBu_r', origin='lower',
                   extent=[times[0], times[-1], freqs[0], freqs[-1]], clim=clim)

    ax.set_aspect('auto')
    cbar = fig.colorbar(im)
    cbar.set_label('$\Delta$dB', fontsize=12)
    ax.set_xlabel('Time (ms)', fontsize=16)
    ax.set_ylabel('Frequency (Hz)', fontsize=16)
    plt.show()
```

Plot the difference of time-frequency results between two conditions

In [17]:
```
freqs = np.arange(4, 32, 2)
times = np.arange(-200, 1000, 4)
plot_tfr_diff_results(tfr_first_No50, tfr_rep_No50, freqs, times,
                      p=0.05, clusterp=0.05, clim=[-2, 2])
```

```
Permutation test

Side 1 begin:
Calculating: [================================================================================
==============] 100.00%
Side 1 finished!

Side 2 begin:

Calculating: [================================================================================
==============] 100.00%
Side 2 finished!

Cluster-based permutation test finished!
```


Frequency (Hz)
30
25
20
15
10
5
−200
0
200
400
600
800
Time (ms)
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
ΔdB

Chapter 4: Advanced EEG Analysis

The section on advanced EEG analysis is divided into the following 4 parts:

- **Part 1: Batch Processing for Reading and Storing Demo Data**
- **Part 2: Classification-based Decoding**
- **Part 3: Representational Similarity Analysis**
- **Part 4: Inverted Encoding Model**

Download and Import Required Python packages

In [1]:
```
! pip install inverted_encoding

import numpy as np
import sys
import os
from six.moves import urllib
import gdown
import zipfile
import scipy.io as sio
from scipy.stats import ttest_ind
import h5py
from tqdm import tqdm
from neurora.decoding import tbyt_decoding_kfold, ct_decoding_kfold
from neurora.rsa_plot import plot_tbyt_decoding_acc, \
                             plot_ct_decoding_acc, \
                             plot_rdm, plot_tbytsim_withstats
from neurora.rdm_cal import eegRDM
from neurora.corr_cal_by_rdm import rdms_corr
from neurora.stuff import smooth_1d
from inverted_encoding import IEM, permutation, circ_diff
from decimal import Decimal
import matplotlib.pyplot as plt
```

```
Requirement already satisfied: inverted_encoding in /Users/zitonglu/anaconda3/lib/python3.11/site-pa
ckages (0.2.3)
Requirement already satisfied: numpy in /Users/zitonglu/anaconda3/lib/python3.11/site-packages (from
inverted_encoding) (1.23.5)
```

Part 1 Batch Processing for Reading and Storing Demo Data

Preprocessed Demo Data 2：

(You will use it in Part 2, Part 3, and Part 4)
The original dataset is based on the data from Experiment 2 in the article "*Dissociable Decoding of Spatial Attention and Working Memory from EEG Oscillations and Sustained Potentials*" by Bae & Luck, published in the Journal of Neuroscience in 2019. This involves a visual working memory task where participants were required to remember the orientation of a teardrop shape presented for 200 ms. After a delay of 1300 milliseconds, a teardrop shape with a random orientation was presented, and participants were required to rotate the mouse to align its orientation as closely as possible with their recollection. The stimulus could appear in 16 different orientations and 16 different locations. Here, only data from the first 5 participants are extracted, consisting of preprocessed ERP data (see the original article for preprocessing parameters) with labels for each trial's orientation and location.

In [2]:
```
# Download Demo Data 2

data_dir = "data/"
if not os.path.exists(data_dir):
  os.makedirs(data_dir)

url = "https://drive.google.com/file/d/1POBi0dckB00AKCIvpXHZFI1Oqd-jWRoj/view?usp=sharing"
filename = "demo_data2"
filepath = data_dir + filename + ".zip"

# Download the data
gdown.download(url=url, output=filepath, quiet=False, fuzzy=True)
```

```
print("Download completes!")
# unzip the data
with zipfile.ZipFile(filepath, 'r') as zip:
    zip.extractall(data_dir)
print("Unzip completes!")
```

```
Downloading...
From (original): https://drive.google.com/uc?id=1POBi0dckB00AKCIvpXHZFI1Oqd-jWRoj
From (redirected): https://drive.google.com/uc?id=1POBi0dckB00AKCIvpXHZFI1Oqd-jWRoj&confirm=t&uuid=4
30b6aca-81e1-44b5-a6f4-a2daf145dc2c
To: /Users/zitonglu/Downloads/Python-EEG-Handbook-master/data/demo_data2.zip
100%|████████████████████████████████████████| 301M/301M [00:57<00:00, 5.27MB/s]
Download completes!
Unzip completes!
```

In the 'data' folder, there are .mat files with EEG data for 5 participants, and in the 'labels' folder, there are files for each participant containing labels for the orientation and location of the stimulus items for all trials. For example, for participant number '201', the file 'ori_201.txt' contains the orientation information, and 'pos_201.txt' contains the location information for each trial's memory item. The data includes 27 electrodes, with a sampling rate of 250Hz, covering from 1.5 seconds before to 1.5 seconds after the stimulus presentation, and each trial includes 750 time points.

Batch Processing for Reading Demo Data 2 then Saving as a .h5 File

In [3]:
```
# Five subjects' ids
sub_ids = ['201', '202', '203', '204', '205']

# Iterate over the data of 5 subjects
for i, sub in enumerate(sub_ids):

    # The file paths for each participant's ERP data,
    # memory item orientation labels,
    # and memory item position labels
    subdata_path = 'data/demo_data2/data/ERP' + sub + '.mat'
    suborilabels_path = 'data/demo_data2/labels/ori_' + sub + '.txt'
    subposlabels_path = 'data/demo_data2/labels/pos_' + sub + '.txt'

    # Read ERP data
    subdata = sio.loadmat(subdata_path)['filtData']
    # Read orientation and position labels
    # In the .txt files,
    # the first column represents the specific orientation/location values,
    # and the second column represents the label values
    # corresponding to the 16 orientations/locations
    # (represented by integers from 0 to 15).
    suborilabels = np.loadtxt(suborilabels_path)[:, 1]
    subposlabels = np.loadtxt(subposlabels_path)[:, 1]

    # Print subj id, the shape of ERP data matrix, the shape of two label matrices
    print('sub' + sub)
    print(subdata.shape)
    print(suborilabels.shape)
    print(subposlabels.shape)

    # Store the EEG data and label data (in matrix form) of each subject
    # in an .h5 file with 'data', 'orilabels' and 'postlabels' as Keys
    f = h5py.File('data/demo_data2/sub' + sub + '.h5', 'w')
    f.create_dataset('data', data=subdata)
    f.create_dataset('orilabels', data=suborilabels)
    f.create_dataset('poslabels', data=subposlabels)
    f.close()
```

```
sub201
(640, 27, 750)
(640,)
(640,)
sub202
(640, 27, 750)
(640,)
(640,)
sub203
(640, 27, 750)
(640,)
(640,)
sub204
(640, 27, 750)
(640,)
(640,)
sub205
(640, 27, 750)
(640,)
(640,)
```

Part 2 Classification-based Decoding

Get EEG Data and Labels

In [4]:
```python
subs = ["201", "202", "203", "204", "205"]

# Initialize data、label_ori, and label_pos to save EEG data and labels
data = np.zeros([5, 640, 27, 500])
label_ori = np.zeros([5, 640])
label_pos = np.zeros([5, 640])

for i, sub in enumerate(subs):

    # Read the .h5 file for each subject
    subfile = h5py.File('data/demo_data2/sub' + sub + '.h5', 'r')

    # Get EEG data
    subdata = subfile['data']

    # Shape of subdata: [640, 27, 750]
    # 640 - N_trials;  27 - N_channels;  750 - N_timepoints (from -1.5s to 1.5s)

    # Only use the EEG data from -0.5s to 1.5s
    subdata = subdata[:, :, 250:]

    # Get orientation labels
    sublabel_ori = subfile['orilabels']
    # Get position labels
    sublabel_pos = subfile['poslabels']

    data[i] = subdata
    label_ori[i] = sublabel_ori
    label_pos[i] = sublabel_pos
```

Time-by-Time EEG Decoding

Time-by-time decoding for orientation information

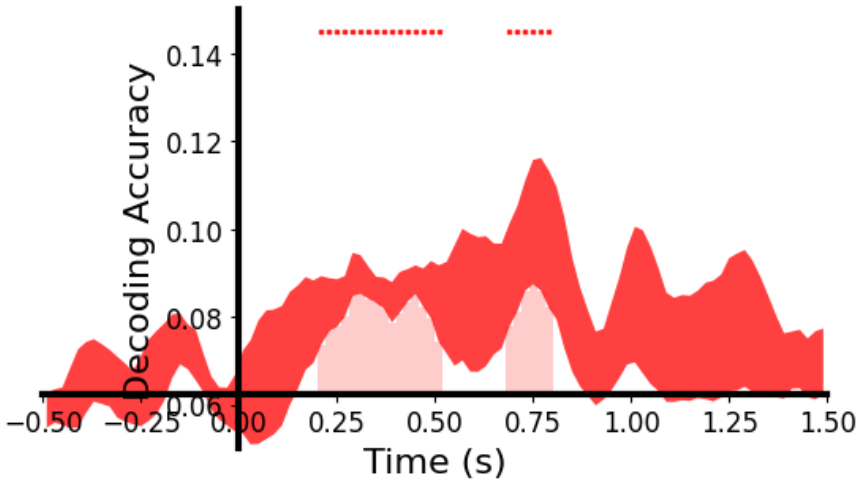
In [5]:
```python
# Here, we use the tbyt_decoding_kfold() function from the decoding module of NeuroRA
# to perform decoding at each time point.
# We input the data (with shape [n_subs, n_trials, n_channels, n_times]),
# labels corresponding to each trial, and several other parameters, including:
# - 'n=16' - The number of classification categories is 16
# (representing 16 orientations)
# - 'navg=13' - Averaging across 13 trials at a time
# - 'time_win=5' - The time window is 5
# (i.e., data from 5 time points are averaged together)
# - 'time_step=5' - The time step is 5
# (i.e., decoding is performed at every interval of 5 time points)
# - 'nfolds=3' & 'nrepeats=10' - Conducting 10 repetitions of 3-fold cross-validation
# - 'smooth=True' - Applying smoothing to the results
accs_ori = tbyt_decoding_kfold(data, label_ori, n=16, navg=13, time_win=5, time_step=5,
                               nfolds=3, nrepeats=10, smooth=True)
```

```
Calculating: [================================================================================
==============] 100.00%
Decoding finished!
```

Plot time-by-time orientation decoding results

In [6]:
```
# Here, we use the plot_tbyt_decoding_acc() function from the decoding module of NeuroRA
# for statistical analysis and visualization.
# It requires the input of decoding results (shape as [n_subs, n_results_times])
# and several other parameters:
# - 'start_time=-0.5' - The start time of the decoding results is -0.5s,
# - 'end_time=1.5' - The end time of the decoding results is 1.5s,
# - 'time_interval=0.02' - The interval between two decoding times is 0.02s
#   (since the time window is 5, and the original sampling rate is 250Hz, 5*(1/250)=0.02s),
# - 'chance=0.0625' - The chance level of classification accuracy is 0.0625
# (16 orientations, i.e., 1/16),
# - 'p=0.05' - The threshold for p-value,
# - 'cbpt=True' - Conducting cluster-based permutation test,
# - 'stats_time=[0, 1.5]' - Analyzing decoding results from 0 to 1.5s,
# - 'xlim=[-0.5, 1.5]' - The display range of the x-axis from -0.5 to 1.5,
# - 'ylim=[0.05, 0.15]' - The display range of the y-axis from 0.05 to 0.15.
plot_tbyt_decoding_acc(accs_ori, start_time=-0.5, end_time=1.5, time_interval=0.02,
                       chance=0.0625, p=0.05, cbpt=True, stats_time=[0, 1.5],
                       xlim=[-0.5, 1.5], ylim=[0.05, 0.15])
```

```
Permutation test
Calculating: [================================================================================
==============] 100.00%
Cluster-based permutation test finished!


Significant time-windows:
200ms to 520ms
679ms to 800ms
```



Out[6]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1., 1., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

Time-by-time decoding for position information

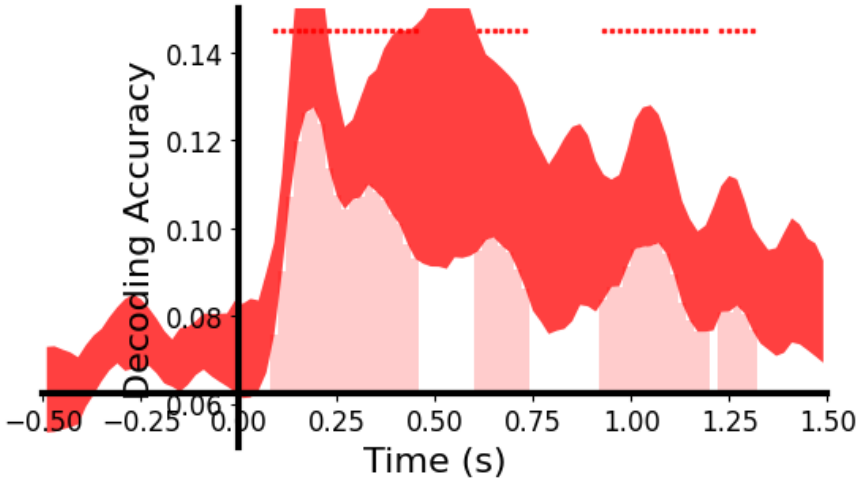
In [7]:
```
accs_pos = tbyt_decoding_kfold(data, label_pos, n=16, navg=13, time_win=5, time_step=5,
                               nfolds=3, nrepeats=10, smooth=True)
```

```
Calculating: [================================================================================
==============] 100.00%
Decoding finished!
```

Plot time-by-time position decoding results

In [8]:
```
plot_tbyt_decoding_acc(accs_pos, start_time=-0.5, end_time=1.5, time_interval=0.02,
                       chance=0.0625, p=0.05, cbpt=True, stats_time=[0, 1.5],
                       xlim=[-0.5, 1.5], ylim=[0.05, 0.15])
```

```
Permutation test
Calculating: [============================================================================================
==============] 100.00%
Cluster-based permutation test finished!


Significant time-windows:
79ms to 459ms
600ms to 740ms
919ms to 1200ms
1220ms to 1320ms
```



Out[8]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0., 0., 0.,
       0., 0., 0., 0., 1., 1., 1., 1., 1., 1., 1., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       0., 1., 1., 1., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

Cross-Temporal EEG Decoding

Cross-temporal decoding for orientation and position information

for orientation

In [9]:
```
# Here, we use the ct_decoding_kfold() function from the decoding module of NeuroRA
accs_crosstime_ori = ct_decoding_kfold(data, label_ori, n=16, navg=13,
                                       time_win=5, time_step=5,
                                       nfolds=3, nrepeats=10, smooth=True)
```

```
Decoding
Calculating: [============================================================================================
==============] 100.00%
Decoding finished!
```

for position

In [10]:
```
accs_crosstime_pos = ct_decoding_kfold(data, label_pos, n=16, navg=13,
                                       time_win=5, time_step=5,
                                       nfolds=3, nrepeats=10, smooth=True)
```

```
Decoding
Calculating: [============================================================================================
==============] 100.00%
Decoding finished!
```

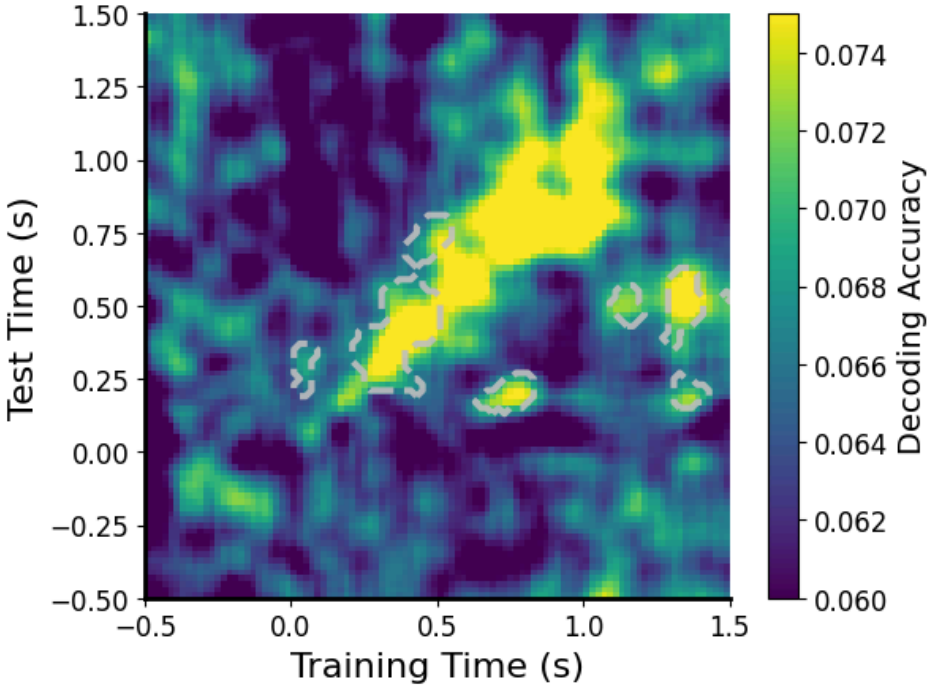
Plot cross-temporal orientation and position decoding results

for orientation

In [11]:
```
# Here, we use the plot_ct_decoding_acc() function from the decoding module of NeuroRA
plot_ct_decoding_acc(accs_crosstime_ori, start_timex=-0.5, end_timex=1.5,
                     start_timey=-0.5, end_timey=1.5,
                     time_intervalx=0.02, time_intervaly=0.02,
                     chance=0.0625, p=0.05, cbpt=True,
                     stats_timex=[0, 1.5], stats_timey=[0, 1.5],
                     xlim=[-0.5, 1.5], ylim=[-0.5, 1.5], clim=[0.06, 0.075])
```

```
Permutation test
Calculating: [================================================================================
==============] 100.00%
Cluster-based permutation test finished!
```

```
/Users/zitonglu/anaconda3/lib/python3.11/site-packages/neurora/rsa_plot.py:1050: UserWarning: No con
tour levels were found within the data range.
  plt.contour(X, Y, np.transpose(newps, (1, 0)), [0, 1], colors="silver", alpha=0.9, linewidths=3,
```

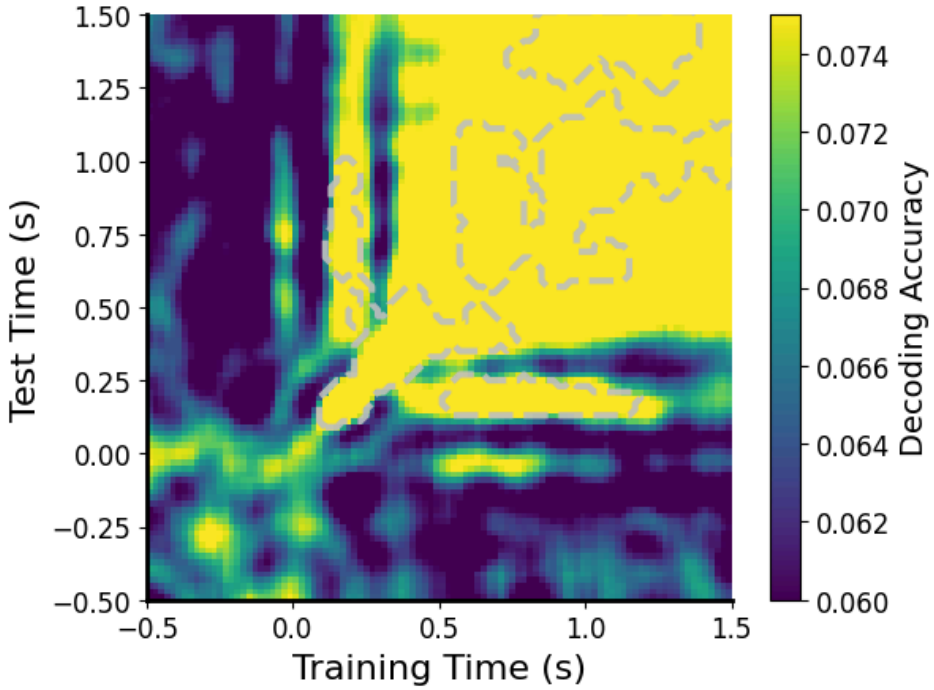


Out[11]:
```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

for position

In [12]:
```
plot_ct_decoding_acc(accs_crosstime_pos, start_timex=-0.5, end_timex=1.5,
                     start_timey=-0.5, end_timey=1.5,
                     time_intervalx=0.02, time_intervaly=0.02,
                     chance=0.0625, p=0.05, cbpt=True,
                     stats_timex=[0, 1.5], stats_timey=[0, 1.5],
                     xlim=[-0.5, 1.5], ylim=[-0.5, 1.5], clim=[0.06, 0.075])
```

```
Permutation test
Calculating: [================================================================================
==============] 100.00%
Cluster-based permutation test finished!
```



Out[12]:
```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

Part 3 Representational Similarity Analysis

Calculate EEG Representational Dissimilarity Matrices (RDMs)

Based on the 16 orientations and 16 locations, calculate the EEG RDM (a 16×16 matrix) representing orientation information and the EEG RDM representing position information at each time point, respectively.

In [13]:
```python
n_subs = 5
n_trials = 640
# Obtain EEG data under 16 orientation conditions and 16 position conditions, respectively
# Initialize data_ori and data_pos to store data for orientation and position, respectively
data_ori = np.zeros([16, n_subs, 40, 27, 500])
data_pos = np.zeros([16, n_subs, 40, 27, 500])
for sub in range(n_subs):
    index_ori = np.zeros([16], dtype=int)
    index_pos = np.zeros([16], dtype=int)
    for i in range(n_trials):
        ori = int(label_ori[sub, i])
        pos = int(label_pos[sub, i])
        data_ori[ori, sub, index_ori[ori]] = data[sub, i]
        index_ori[ori] = index_ori[ori] + 1
        data_pos[pos, sub, index_pos[pos]] = data[sub, i]
        index_pos[pos] = index_pos[pos] + 1

# Calculate EEG RDMs using eegRDM() function from the rdm_cal module in NeuroRA
# Orientation RDMs
RDMs_ori = eegRDM(data_ori, sub_opt=1, chl_opt=0, time_opt=1, time_win=5, time_step=5)
# Position RDMs
RDMs_pos = eegRDM(data_pos, sub_opt=1, chl_opt=0, time_opt=1, time_win=5, time_step=5)
# The shapes of the returned RDMs_ori and RDMs_pos are
# both [N_subs, N_results_time, 16, 16] ([5, 100, 16, 16]).
```

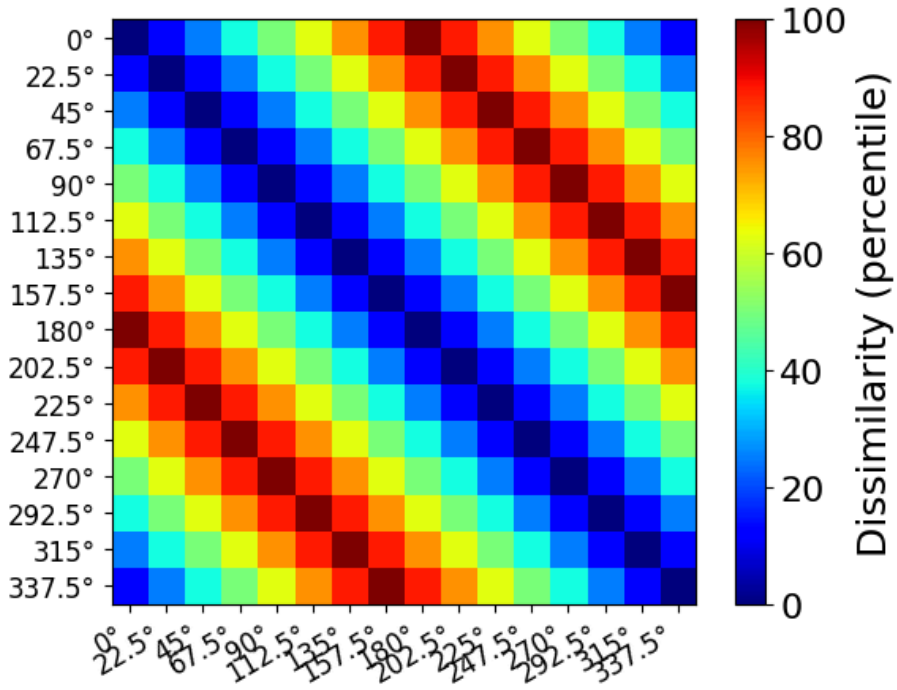
```
Computing RDMs
Calculating: [================================================================================
==============] 100.00%
RDMs computing finished!

Computing RDMs
Calculating: [================================================================================
==============] 100.00%
RDMs computing finished!
```

Construct the Hypothesis-based RDM

The hypothesis-based RDM is based on the hypothesis that the closer two angles are, the higher their similarity, and the more distant two angles are, the lower their similarity.
Here, the RDM below is applicable to both orientation and position information in our example.

In [14]:
```
model_RDM = np.zeros([16, 16])
for i in range(16):
    for j in range(16):
        diff = np.abs(i - j)
        if diff <= 8:
            model_RDM[i, j] = diff / 8
        else:
            model_RDM[i, j] = (16 - diff) / 8

# Visualize the RDM using plot_rdm() function from the rsa_plot module in NeuroRA
conditions = ["0°", "22.5°", "45°", "67.5°", "90°", "112.5°", "135°", "157.5°", "180°",
              "202.5°", "225°", "247.5°", "270°", "292.5°", "315°", "337.5°"]
plot_rdm(model_RDM, percentile=True, conditions=conditions)
```

```
1
```



Out[14]:
```
0
```

RSA

Calculate the similarity between the hypothesis-based RDM and the orientation EEG RDMs, as well as the position EEG RDMs, to temporally track when the brain conforms to the encoding of orientation and/or position.

In [15]:
```
# Use the rdms_corr() function from the corr_cal_by_rdm module in NeuroRA
# Calculate the similarity between the hyp RDM and ori EEG RDMs
similarities_ori = rdms_corr(model_RDM, RDMs_ori)
# Calculate the similarity between the hyp RDM and posi EEG RDMs
similarities_pos = rdms_corr(model_RDM, RDMs_pos)
```

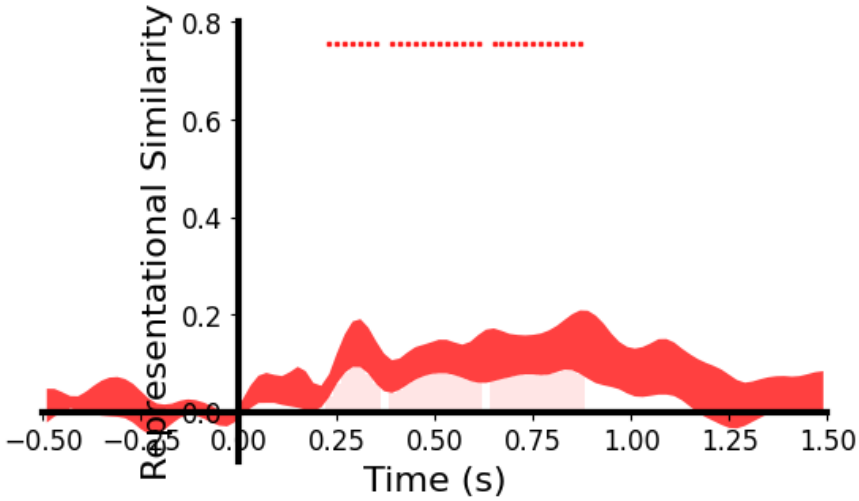
```
Computing similarities

Computing finished!

Computing similarities

Computing finished!
```

Plot the RSA results of orientation information

In [16]:
```
# Here, we use plot_tbytsim_withstats() function from the rsa_plot module in NeuroRA
# Similar to plot_tbyt_decoding_acc()

plot_tbytsim_withstats(similarities_ori, start_time=-0.5, end_time=1.5,
```

```
                        time_interval=0.02, smooth=True, p=0.05, cbpt=True,
                        stats_time=[0, 1.5], xlim=[-0.5, 1.5], ylim=[-0.1, 0.8])
```

```
Permutation test
Calculating: [================================================================================
==============] 100.00%
Cluster-based permutation test finished!


Significant time-windows:
219ms to 360ms
380ms to 620ms
640ms to 880ms
```

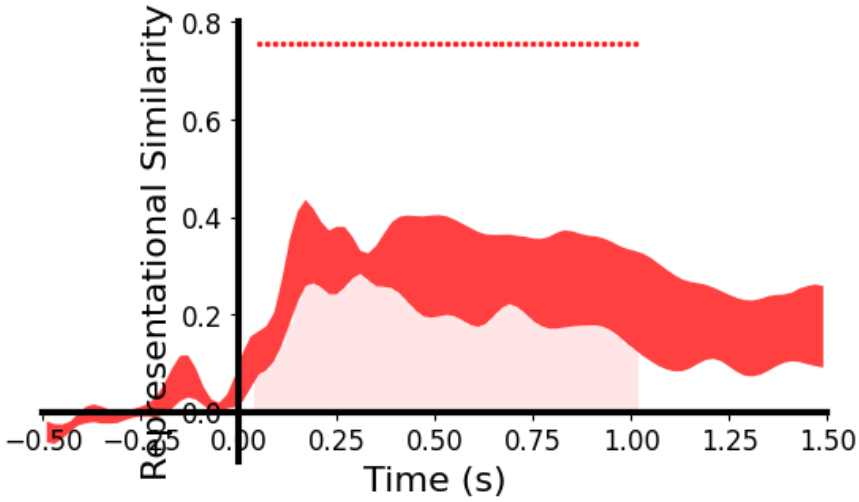


Out[16]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 1., 1., 1., 1., 1., 1., 0., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

Plot the RSA results of position information

In [17]:
```
plot_tbytsim_withstats(similarities_pos, start_time=-0.5, end_time=1.5,
                        time_interval=0.02, smooth=True, p=0.05, cbpt=True,
                        stats_time=[0, 1.5], xlim=[-0.5, 1.5], ylim=[-0.1, 0.8])
```

```
Permutation test
Calculating: [================================================================================
==============] 100.00%
Cluster-based permutation test finished!


Significant time-windows:
40ms to 1020ms
```



Out[17]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

Part 4 Inverted Encoding Model

In this part, we refer to a preprint "*Scotti, P. S., Chen, J., & Golomb, J. D. (2021). An enhanced inverted encoding model for neural reconstructions. bioRxiv*" that utilized enhanced inverted encoding model (eIEM).

Apply eIEM to decode orientation information

In [18]:
```
n_subs = 5
n_ts = 100

# Initialize mae_ori to save MAE results from IEM decoding
mae_ori = np.zeros([n_subs, n_ts])

# Calculate the MAE for each timepoint and subject using IEM function in Inverted_Encoding
for t in tqdm(range(n_ts)):
    for sub in range(n_subs):

        # Downsampling the data - average the data for every 5 timepoints
        data_t_sub = np.average(data[sub, :, :, t*5:t*5+5], axis=2)

        # Get IEM predictions
        predictions, _, _, _ = IEM(data_t_sub, label_ori[sub].astype(int),
                                   stim_max=16, nfolds=5, is_circular=True)

        # Calculate MAE
        mae_ori[sub, t] = np.mean(np.abs(circ_diff(predictions, label_ori[sub].astype(int), 16)))

# Use permutation() function in Inverted_Encoding to get null distribution of MAE
null_mae_ori = permutation(label_ori[sub].astype(int), stim_max=16,num_perm=5000, is_circular=True)
```

```
100%|██████████| 100/100 [01:28<00:00,  1.13it/s]
```

Define a function to plot eIEM results - plot_iem_results()

For the visualization, we calculate the ΔMAE (the mean of null mae minus mae) as the final IEM results.

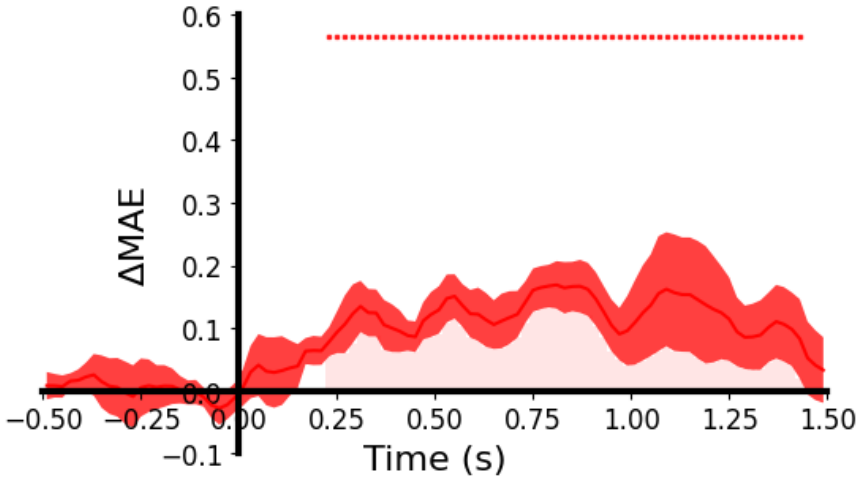
In [19]:
```
def plot_iem_results(mae_ori, null_mae, start_time=0, end_time=1, time_interval=0.01,
                     stats_time=[0, 1], xlim=[0, 1], ylim=[-0.1, 0.8]):

    """
    Plot the time-by-time eIEM results

    Parameters
    ----------
    mae : array
        The mae results.
    null_mae : array
        The null distribution of mae.
    start_time : int or float. Default is 0.
        The start time.
    end_time : int or float. Default is 1.
        The end time.
    time_interval : float. Default is 0.01.
        The time interval between two time samples.
    xlim : array or list [xmin, xmax]. Default is [0, 1].
        The x-axis (time) view lims.
    ylim : array or list [ymin, ymax]. Default is [0.4, 0.8].
        The y-axis (decoding accuracy) view lims.
    """

    if len(np.shape(mae_ori)) != 2:

        return "Invalid input!"

    n = len(np.shape(mae_ori))

    yminlim = ylim[0]
    ymaxlim = ylim[1]

    nsubs, nts = np.shape(mae_ori)
    tstep = float(Decimal((end_time - start_time) / nts).quantize(Decimal(str(time_interval))))

    if tstep != time_interval:
        return "Invalid input!"

    delta1 = (stats_time[0] - start_time) / tstep - int((stats_time[0] - start_time) / tstep)
```

```
    delta2 = (stats_time[1] - start_time) / tstep - int((stats_time[1] - start_time) / tstep)

    mae_ori = smooth_1d(mae_ori)

    avg = np.average(np.average(null_mae)-mae_ori, axis=0)
    err = np.zeros([nts])

    for t in range(nts):
        err[t] = np.std(mae_ori[:, t], ddof=1)/np.sqrt(nsubs)

    ps = np.zeros([nts])
    for t in range(nts):
        ps[t] = ttest_ind(mae_ori[:, t], null_mae, alternative='less')[1]
        if ps[t] < 0.05:
            ps[t] = 1
        else:
            ps[t] = 0

    print('\nSignificant time-windows:')
    for t in range(nts):
        if t == 0 and ps[t] == 1:
            print(str(int(start_time * 1000)) + 'ms to ', end='')
        if t > 0 and ps[t] == 1 and ps[t - 1] == 0:
            print(str(int((start_time + t * tstep) * 1000)) + 'ms to ', end='')
        if t < nts - 1 and ps[t] == 1 and ps[t + 1] == 0:
            print(str(int((start_time + (t + 1) * tstep) * 1000)) + 'ms')
        if t == nts - 1 and ps[t] == 1:
            print(str(int(end_time * 1000)) + 'ms')

    for t in range(nts):
        if ps[t] == 1:
            plt.plot(t*tstep+start_time+0.5*tstep, (ymaxlim-yminlim)*0.95+yminlim, 's',
                     color='r', alpha=0.8, markersize=2)
            xi = [t*tstep+start_time, t*tstep+tstep+start_time]
            ymin = [0]
            ymax = [avg[t]-err[t]]
            plt.fill_between(xi, ymax, ymin, facecolor='r', alpha=0.1)

    fig = plt.gcf()
    fig.set_size_inches(6.4, 3.6)

    ax = plt.gca()
    ax.spines["top"].set_visible(False)
    ax.spines["right"].set_visible(False)
    ax.spines["left"].set_linewidth(3)
    ax.spines["left"].set_position(("data", 0))
    ax.spines["bottom"].set_linewidth(3)
    ax.spines['bottom'].set_position(('data', 0))
    x = np.arange(start_time+0.5*tstep, end_time+0.5*tstep, tstep)
    plt.plot(x, avg, color='r', alpha=0.95)
    plt.fill_between(x, avg + err, avg - err, facecolor='r', alpha=0.75)
    plt.ylim(yminlim, ymaxlim)
    plt.xlim(xlim[0], xlim[1])
    plt.tick_params(labelsize=12)
    plt.xlabel('Time (s)', fontsize=16)
    plt.ylabel(r'$\Delta$MAE', fontsize=16)

    plt.show()

    return ps
```

Plot the IEM results of orientation information

In [20]:
```
plot_iem_results(mae_ori, null_mae_ori, start_time=-0.5, end_time=1.5, time_interval=0.02,
                 xlim=[-0.5, 1.5], ylim=[-0.1, 0.6])
```

```
Significant time-windows:
219ms to 1440ms
```



Out[20]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0., 0., 0.])
```

Apply eIEM to decode position information

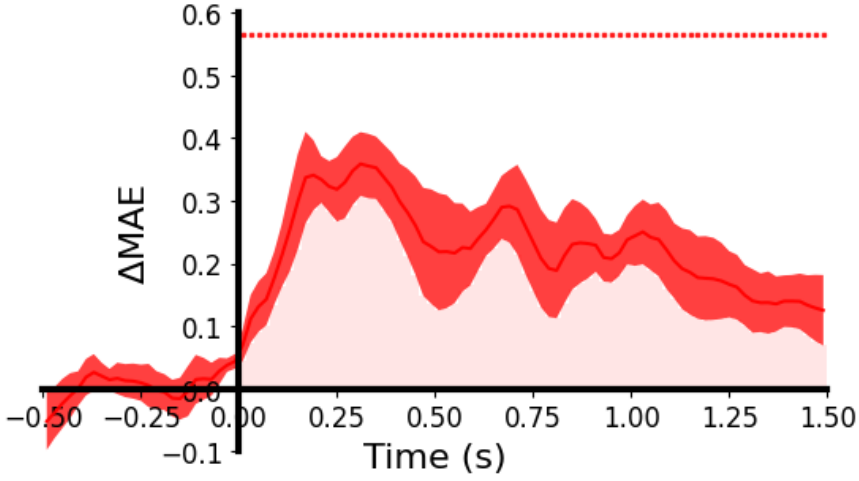
In [21]:
```
# Initialize mae_pos to save MAE results from IEM decoding
mae_pos = np.zeros([n_subs, n_ts])

# Calculate the MAE for each timepoint and subject using IEM function in Inverted_Encoding
for t in tqdm(range(n_ts)):
    for sub in range(n_subs):

        # Downsampling the data - average the data for every 5 timepoints
        data_t_sub = np.average(data[sub, :, :, t*5:t*5+5], axis=2)

        # Get IEM predictions
        predictions, _, _, _ = IEM(data_t_sub, label_pos[sub].astype(int), stim_max=16, nfolds=5, is

        # Calculate MAE
        mae_pos[sub, t] = np.mean(np.abs(circ_diff(predictions, label_pos[sub].astype(int), 16)))

# Use permutation() function in Inverted_Encoding to get null distribution of MAE
null_mae_pos = permutation(label_pos[sub].astype(int), stim_max=16, num_perm=5000, is_circular=True)
```

```
100%|██████████████████████████████████████| 100/100 [01:28<00:00,  1.13it/s]
```

Plot the IEM results of position information

In [22]:
```
plot_iem_results(mae_pos, null_mae_pos, start_time=-0.5, end_time=1.5, time_interval=0.02,
                 xlim=[-0.5, 1.5], ylim=[-0.1, 0.6])
```

```
Significant time-windows:
0ms to 1500ms
```



Out[22]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

Acknowledgement

We would like to express our sincere gratitude to the active support and feedback from the members of the WeChat groups "MNE-Python Study" and "NeuroRA Q&A". We are particularly thankful for the valuable reference provided by Steven Luck's book "An introduction to the Event-Related Potential Technique", as well as the official documentation and tutorials from EEGLab and MNE-Python. Special thanks go to Mengxin Ran, an undergraduate from OSU Vision & Cognitive Neuroscience Lab, for her meticulous review of the content. We are grateful to the authors of the public datasets used in this tutorial, including Daniel Wakeman, Richard Henson, Gi-Yeul Bae, and Steven Luck, who have made significant contributions to the open science community. These datasets form the foundation that allows our handbook to have detailed examples. We thank all the readers who helped promote our handbook and provided support and feedback after the release of the first Chinese version three years ago. We would also like to extend our thanks to our beloved group pet cat "Fanfan"  , who has been a source of joy and comfort in the WeChat group for all of the authors.

References

1. Delorme, A., and Makeig, S. (2004). EEGLAB: an open source toolbox for analysis of single-trial EEG dynamics including independent component analysis. J. Neurosci. Methods *134*, 9–21. 10.1016/J.JNEUMETH.2003.10.009.
2. Gramfort, A., Luessi, M., Larson, E., Engemann, D.A., Strohmeier, D., Brodbeck, C., Goj, R., Jas, M., Brooks, T., Parkkonen, L., et al. (2013). MEG and EEG data analysis with MNE-Python. Front. Neurosci. *7*, 70133. 10.3389/FNINS.2013.00267.
3. Nilearn 10.5281/zenodo.8397156.
4. NiBabel 10.5281/zenodo.591597.
5. Lu, Z., and Ku, Y. (2020). NeuroRA: A Python Toolbox of Representational Analysis From Multi-Modal Neural Data. Front. Neuroinform. *14*, 61. 10.3389/FNINF.2020.563669.
6. Wakeman, D.G., and Henson, R.N. (2015). A multi-subject, multi-modal human neuroimaging dataset. Sci. Data *2*, 1–10. 10.1038/sdata.2015.1.
7. Norman, K.A., Polyn, S.M., Detre, G.J., and Haxby, J. V. (2006). Beyond mind-reading: multi-voxel pattern analysis of fMRI data. Trends Cogn. Sci. *10*, 424–430. 10.1016/J.TICS.2006.07.005.
8. Grootswagers, T., Wardle, S.G., and Carlson, T.A. (2017). Decoding Dynamic Brain Patterns from Evoked Responses: A Tutorial on Multivariate Pattern Analysis Applied to Time Series Neuroimaging Data. J. Cogn. Neurosci. *29*, 677–697. 10.1162/JOCN_A_01068.
9. Kriegeskorte, N., Mur, M., and Bandettini, P. (2008). Representational similarity analysis - connecting the branches of systems neuroscience. Front. Syst. Neurosci. *4*. 10.3389/NEURO.06.004.2008.
10. Kriegeskorte, N., Mur, M., Ruff, D.A., Kiani, R., Bodurka, J., Esteky, H., Tanaka, K., and Bandettini, P.A. (2008). Matching Categorical Object Representations in Inferior Temporal Cortex of Man and Monkey. Neuron *60*, 1126–1141. 10.1016/J.NEURON.2008.10.043.
11. Scotti, P.S., Chen, J., and Golomb, J.D. (2022). An enhanced inverted encoding model for neural reconstructions. bioRxiv. 10.1101/2021.05.22.445245.
12. Brouwer, G.J., and Heeger, D.J. (2009). Decoding and Reconstructing Color from

Responses in Human Visual Cortex. J. Neurosci. *29*, 13992–14003. 10.1523/JNEUROSCI.3577-09.2009.
13. Brouwer, G.J., and Heeger, D.J. (2011). Cross-orientation suppression in human visual cortex. J. Neurophysiol. *106*, 2108–2119. 10.1152/jn.00540.2011.
14. Sprague, T.C., Saproo, S., and Serences, J.T. (2015). Visual attention mitigates information loss in small- and large-scale neural codes. Trends Cogn. Sci. *19*, 215–226. 10.1016/J.TICS.2015.02.005.
15. Bae, G.Y., and Luck, S.J. (2018). Dissociable Decoding of Spatial Attention and Working Memory from EEG Oscillations and Sustained Potentials. J. Neurosci. *38*, 409–422. 10.1523/JNEUROSCI.2860-17.2017.