# 实验 1：EEG、MEG 数据处理基础

2024 年 3 月 25 日

实验 1：EEG/MEG 数据处理基础

## 1 实验简介

本实验主要介绍神经电生理学数据的处理方法。

通过本实验，你将学习 EEG/MEG 数据的查看和分析方法，掌握以下技能：

- 通过开放获取数据，了解 EEG/MEG 数据的组织方式；
- 利用 `MNE-Python` 开源工具包，查看并初步处理 EEG/MEG 数据；
- 了解 EEG/MEG 数据的常用预处理方法。

## 2 实验准备

为了顺利完成本实验的操作内容，你需要首先完成 `MNE-Python` 工具包及其依赖的安装。为了加快安装速度，请确保 `pip` 包管理器已切换到国内源。本实验的依赖包主要包括：

- `MNE-Python`：开源的脑电图和脑磁图数据处理工具；
- `numpy`：Python 科学计算的基础软件包；
- `pyvistaqt`：基于 QT 的三维可视化工具，主要用于辅助查看 `MNE-Python` 分析结果。

执行以下命令，检查并安装相关依赖。

[1]:
```
import pkgutil

# 检查 MNE-Python 是否已安装
if pkgutil.find_loader('mne'):
    print('MNE-Python is available.')
```

```
else:
    !pip install mne --quiet

# 检查 pyvistaqt 是否已安装
if pkgutil.find_loader('pyvistaqt'):
    print('pyvistaqt is available.')
else:
    !pip install pyvistaqt --quiet
```

```
MNE-Python is available.
pyvistaqt is available.
```

[2]:
```
# 忽略 Warning 输出
import warnings
warnings.filterwarnings("ignore")
```

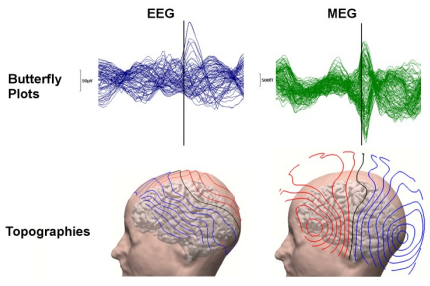
# 3 下载和查看数据

## 3.1 神经电生理学方法

神经电生理学方法（Electrophysiology method）是指基于电学原理，采用电生理仪器及微电极测量或记录神经元生理活动的信号变化的技术。常用的神经电生理学测量方法包括：

- 脑电图（Electroencephalography，EEG）：神经元在兴奋时通过膜内外离子通道的开闭产生随时间变化的电流，EEG 通过将电极放置在人的头皮上来检测大量脑神经元放电的叠加，进而用于研究大脑神经活动；
- 脑磁图（Magnetoencephalography，MEG）：神经元产生的生物电流产生磁场，脑磁图通过测量这种脑磁场信号，对脑功能区进行定位及评价，其信号来源与 EEG 相同，但相较于 EEG 空间分辨率更高，能够与 MRI、CT 等解剖影象信息叠加整合。

  思考/扩展：尝试用表格的形式比较 EEG、MEG 技术的异同。



## 3.2 数据来源

本实验使用 MNE-Python 工具包的样例数据集进行处理、分析。该样例数据集是通过麻省总医院 Athinoula A. Martino 生物医学成像中心的 Neuromag Vectorview 系统获得的，同时采集了 MEG 数据和 60 通道的 EEG 数据。

实验中，被试将在左右视野中看到棋盘格的纹样，在左耳或右耳播放声音。被试将随机看到微笑的人脸图样，此时他们需要尽可能快地按下按钮作为回应。

## 3.3 下载数据并查看

默认情况下，样例数据集没有随 MNE-Python 包一同安装。首先通过 mne.datasets.sample.data_path 下载数据，数据规模约为 1.5G，一般需要数分钟时间完成下载。数据将存放在当前目录下的 mne_data 子目录中。

如果下载速度太慢，可按以下步骤进行数据的获取和配置：

1. 从以下链接下载数据 [Link]，文件提取码：5q1g；
2. 将下载的文件直接解压到该 Notebook 所在的目录（解压得到的 mne_data 目录应该与.ipynb 文件在同一目录下）；
3. 在 Windows 资源管理器地址栏中输入%homepath% 并回车，进入当前 Windows 系统的 Home 目录，在该目录下创建子目录.mne，然后在.mne 目录下创建 JSON 文件 mne-python.json；
4. 打开 mne-python.json 文件，写入以下内容，将路径改为 mne_data 实际的存放位置即可：

```
{
"MNE_DATASETS_SAMPLE_PATH": "PATH\\TO\\YOUR\\mne_data\\DIRECTORY"
}
```

[3]:
```
import numpy as np
import mne
mne.viz.set_3d_backend("pyvistaqt")
```

```
Using pyvistaqt 3d backend.
```

[5]:
```
sample_data_folder = mne.datasets.sample.data_path()
```

在本实验中，我们将使用一个较小规模的子数据集 sample_audvis_filt-0-40_raw.fif 进行分析。

> 思考/扩展：如果你想详细了解 MEG 数据的组织形式，参见[Link]。

[6]:
```
sample_data_raw_file = (
    sample_data_folder / "MEG" / "sample" / "sample_audvis_filt-0-40_raw.fif"
)
raw = mne.io.read_raw_fif(sample_data_raw_file)
```

```
Opening raw data file D:\Code\cogsci\ex1\mne_data\MNE-sample-
data\MEG\sample\sample_audvis_filt-0-40_raw.fif…
    Read a total of 4 projection items:
        PCA-v1 (1 x 102)  idle
        PCA-v2 (1 x 102)  idle
        PCA-v3 (1 x 102)  idle
        Average EEG reference (1 x 60)  idle
    Range : 6450 … 48149 =     42.956 …   320.665 secs
Ready.
```

Raw 是 MNE-Python 中的一个非常有用的数据结构。该数据结构主要用来存储连续型数据，其维度为 n_channels × n_times。我们可通过以下方法查看数据的基本信息。从中我们可以看到，数据中包含了 204 个通道的梯度计信号，102 个通道的磁力计信号，9 个通道的刺激信号，60 个通道的 EEG 信号和 1 个通道的 EOG（眼电位）信号。

[7]:
```
print(raw)
print(raw.info)
```

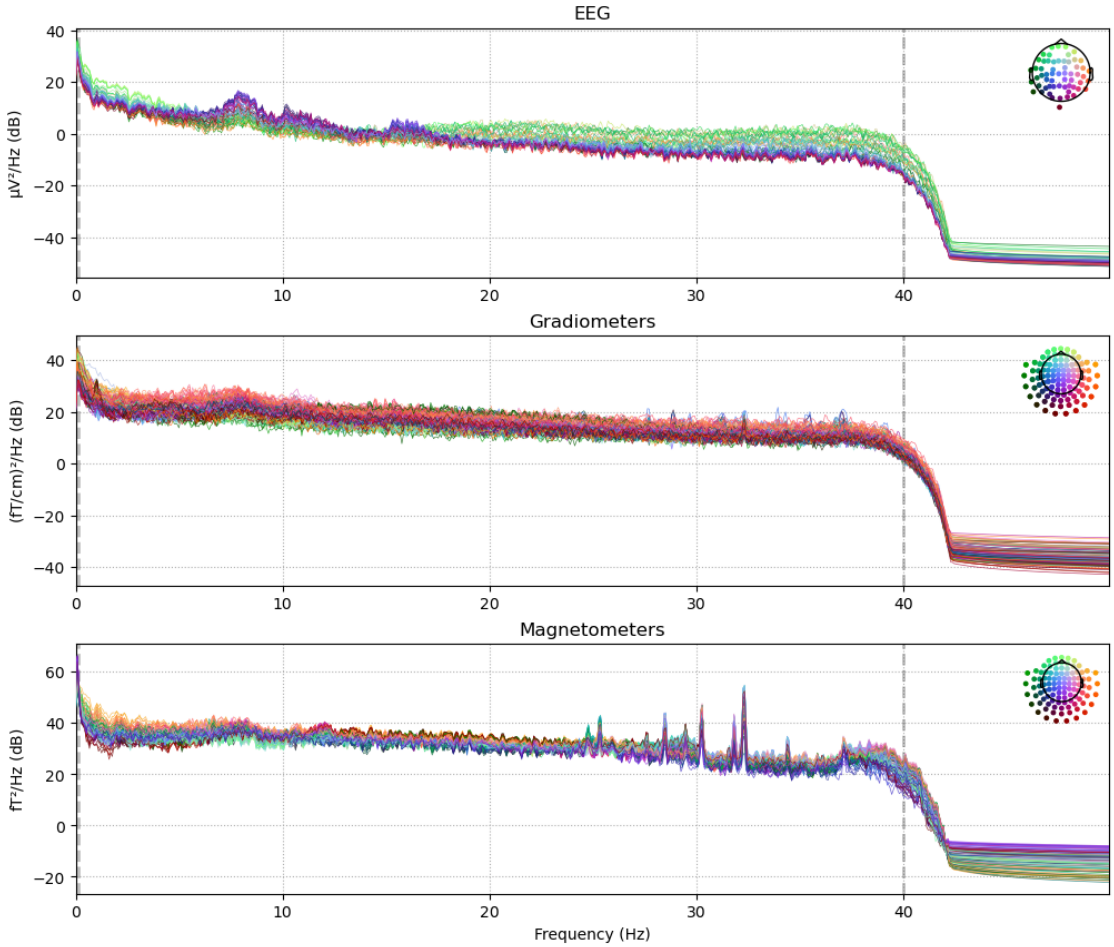
```
<Raw | sample_audvis_filt-0-40_raw.fif, 376 x 41700 (277.7 s), ~3.3 MB, data not
loaded>
<Info | 15 non-empty values
 bads: 2 items (MEG 2443, EEG 053)
 ch_names: MEG 0113, MEG 0112, MEG 0111, MEG 0122, MEG 0123, MEG 0121, MEG …
 chs: 204 Gradiometers, 102 Magnetometers, 9 Stimulus, 60 EEG, 1 EOG
 custom_ref_applied: False
 dev_head_t: MEG device -> head transform
 dig: 146 items (3 Cardinal, 4 HPI, 61 EEG, 78 Extra)
 file_id: 4 items (dict)
 highpass: 0.1 Hz
 hpi_meas: 1 item (list)
 hpi_results: 1 item (list)
 lowpass: 40.0 Hz
 meas_date: 2002-12-03 19:01:10 UTC
```

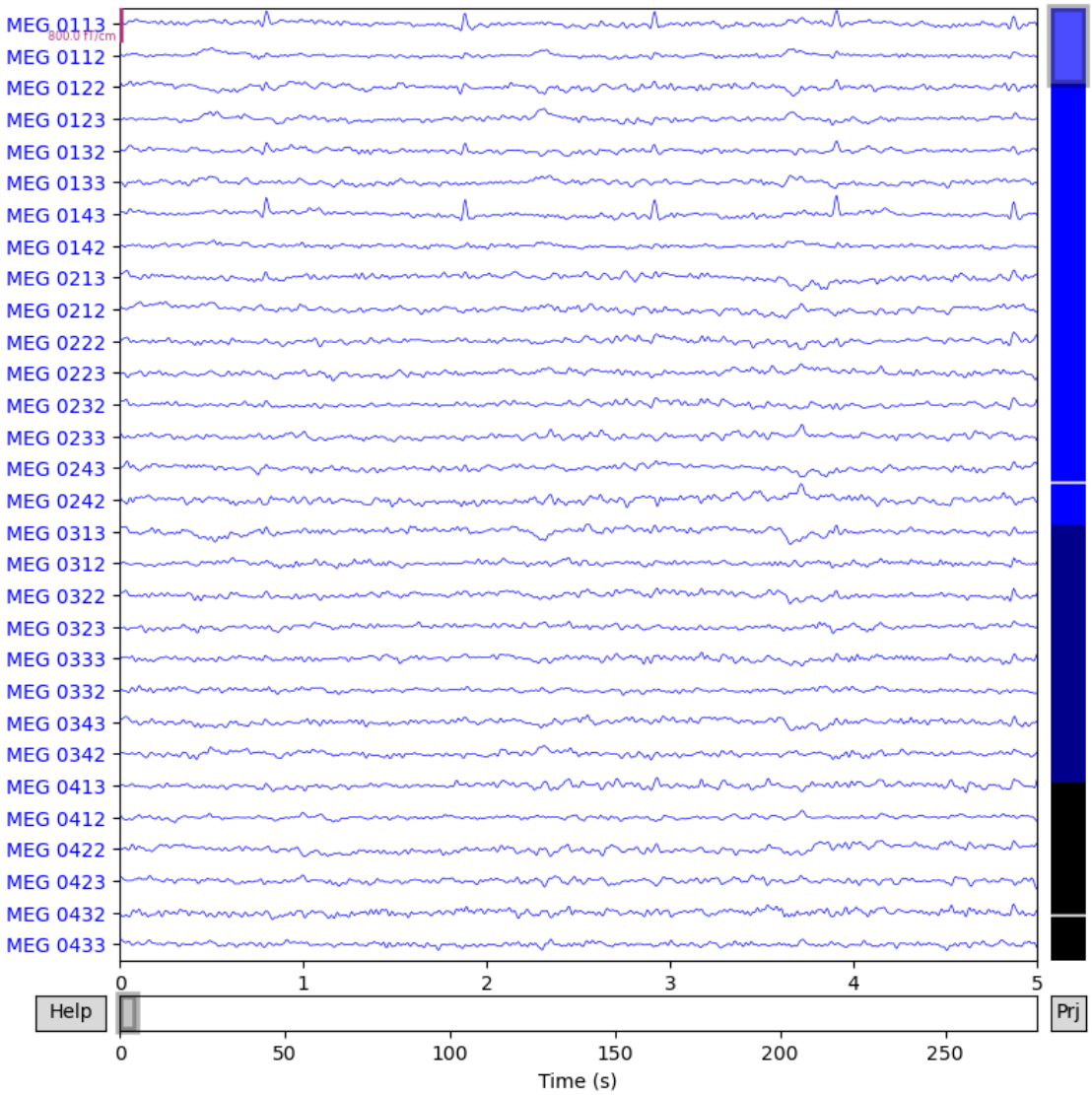
```
 meas_id: 4 items (dict)
 nchan: 376
 projs: PCA-v1: off, PCA-v2: off, PCA-v3: off, Average EEG reference: off
 sfreq: 150.2 Hz
>
```

我们首先使用 Raw 对象的 compute_psd 方法绘制各传感器的功率谱密度图（PSD），这里我们将图标中的频率上限设置为 50Hz（数据已经过 40Hz 的低通滤波处理）。同时，使用 plot 方法绘制波形图。

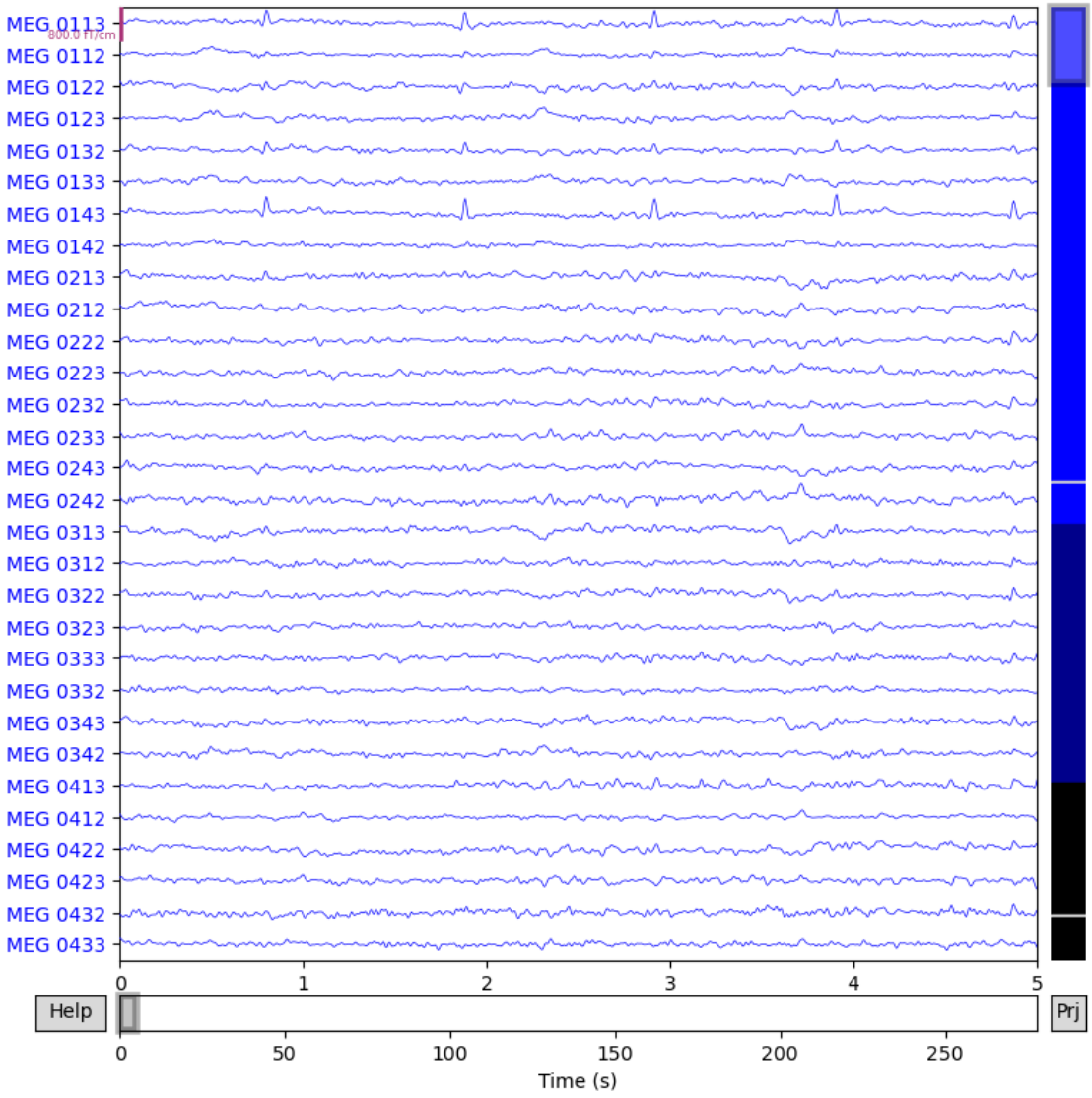
[8]:
```
raw.compute_psd(fmax=50).plot(picks="data", exclude="bads")
raw.plot(duration=5, n_channels=30)
```

```
Effective window size : 13.639 (s)
Using matplotlib as 2D backend.
```




MEG 0113
MEG 0112
MEG 0122
MEG 0123
MEG 0132
MEG 0133
MEG 0143
MEG 0142
MEG 0213
MEG 0212
MEG 0222
MEG 0223
MEG 0232
MEG 0233
MEG 0243
MEG 0242
MEG 0313
MEG 0312
MEG 0322
MEG 0323
MEG 0333
MEG 0332
MEG 0343
MEG 0342
MEG 0413
MEG 0412
MEG 0422
MEG 0423
MEG 0432
MEG 0433
0
1
2
3
4
5
Help
Prj
0
50
100
150
200
250
Time (s)

[8]:




# 4 数据预处理

通过 ICA（Independent components analysis）方法可以从 EEG/MEG 数据中提取出若干独立成分（例如眨眼、眼动、视觉刺激等等）。

思考/扩展：简述 ICA 方法的原理和应用场景。

[9]:
```
# 初始化一个 ICA 处理器
ica = mne.preprocessing.ICA(n_components=20, random_state=97, max_iter=800)

# 运行 ICA 分解
ica.fit(raw)
```

```
Fitting ICA to data using 364 channels (please be patient, this may take a
while)
Selecting by number: 20 components
Fitting ICA took 2.1s.
```

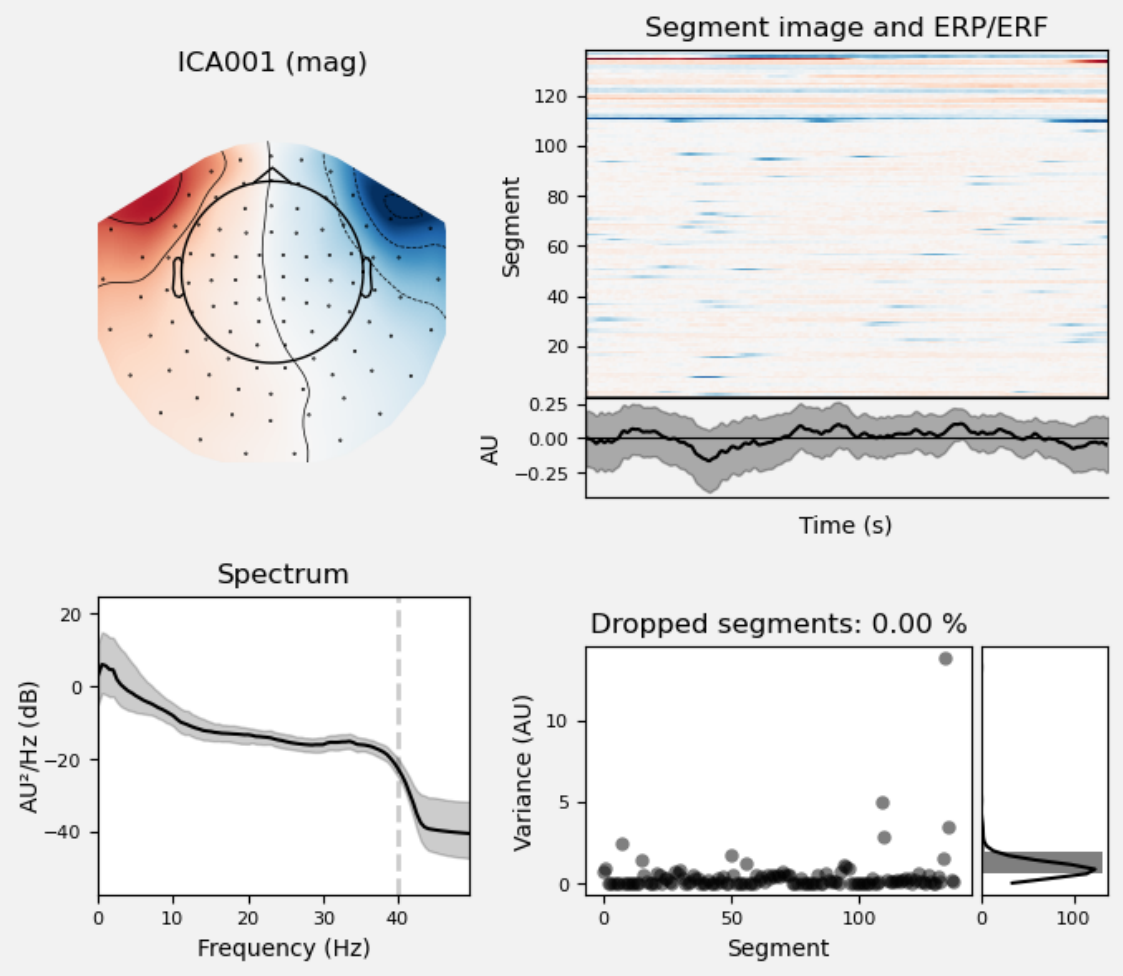
[9]:
```
<ICA | raw data decomposition, method: fastica (fit in 27 iterations on 41700
samples), 20 ICA components (364 PCA components available), channel types: mag,
grad, eeg, no sources marked for exclusion>
```

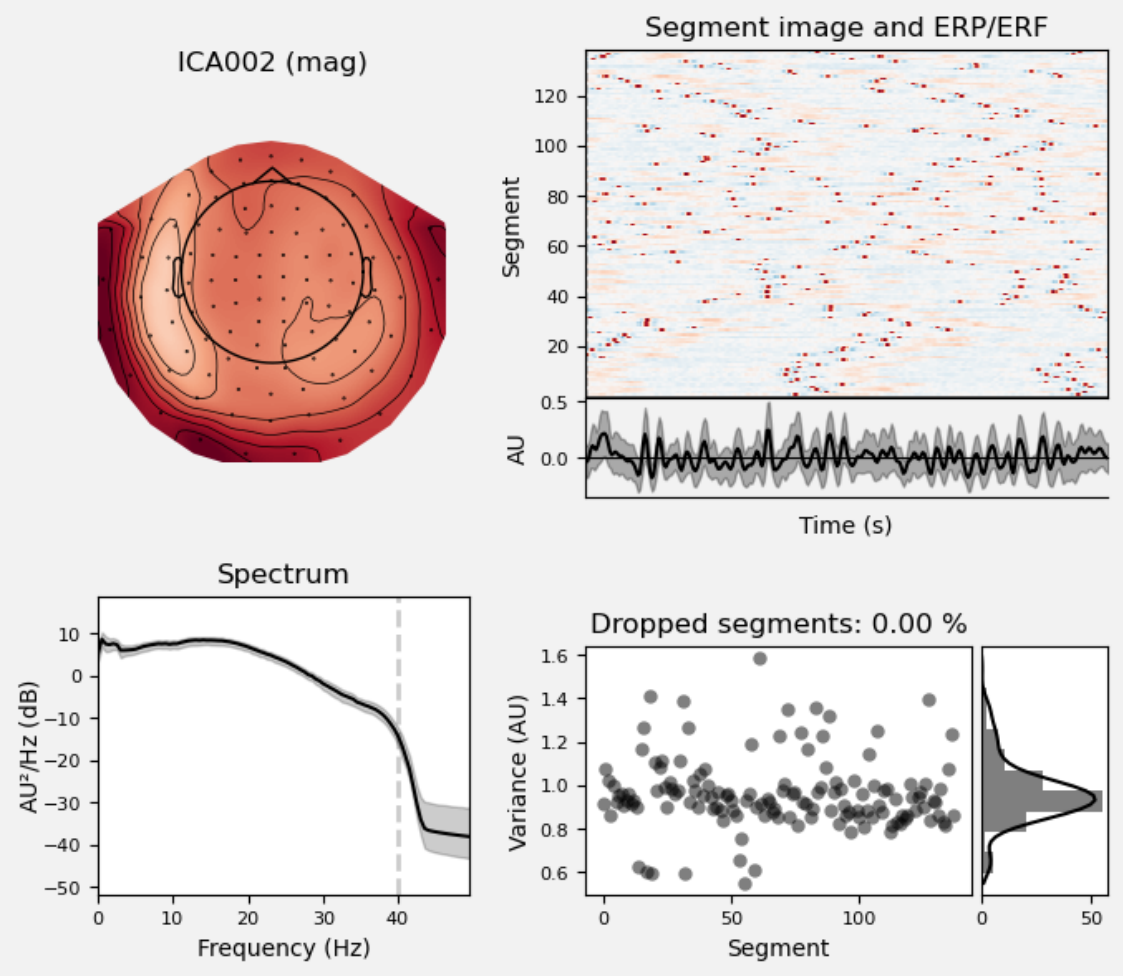
然而，由于在采集过程中，数据很容易混入很多类型的噪声，如复杂的运动伪影、电极坏道等，会对 ICA 分解会产生不良影响，因此需要剔除一些不良成分。

这里，我们剔除 1、2 号成分作为示例。

[10]:
```
# 剔除一些不良通道
ica.exclude = [1, 2]
ica.plot_properties(raw, picks=ica.exclude)
```

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


ICA001 (mag)
Segment image and ERP/ERF
Segment
120
100
80
60
40
20
AU
0.25
0.00
−0.25
Time (s)
Spectrum
AU²/Hz (dB)
20
0
−20
−40
0
10
20
30
40
Frequency (Hz)
Dropped segments: 0.00 %
Variance (AU)
10
5
0
0
50
100
Segment
0
100




[10]: [<Figure size 700x600 with 6 Axes>, <Figure size 700x600 with 6 Axes>]

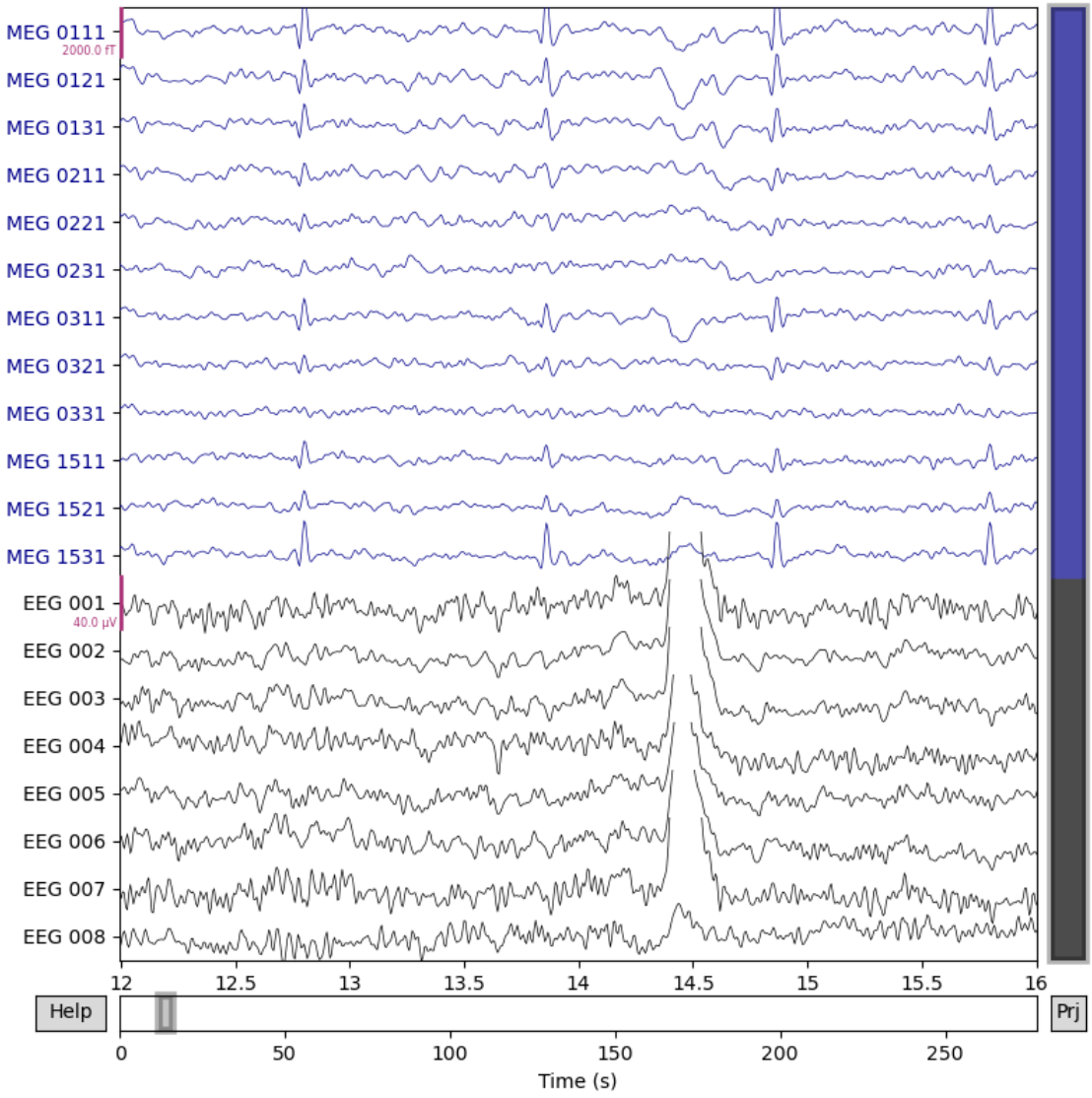
我们刚才已经指定了要剔除（ica.exclude）的成分，此时可以通过 apply 方法应用到数据集中。这里，我们将原始数据复制一份，以便比较剔除前后的差异。

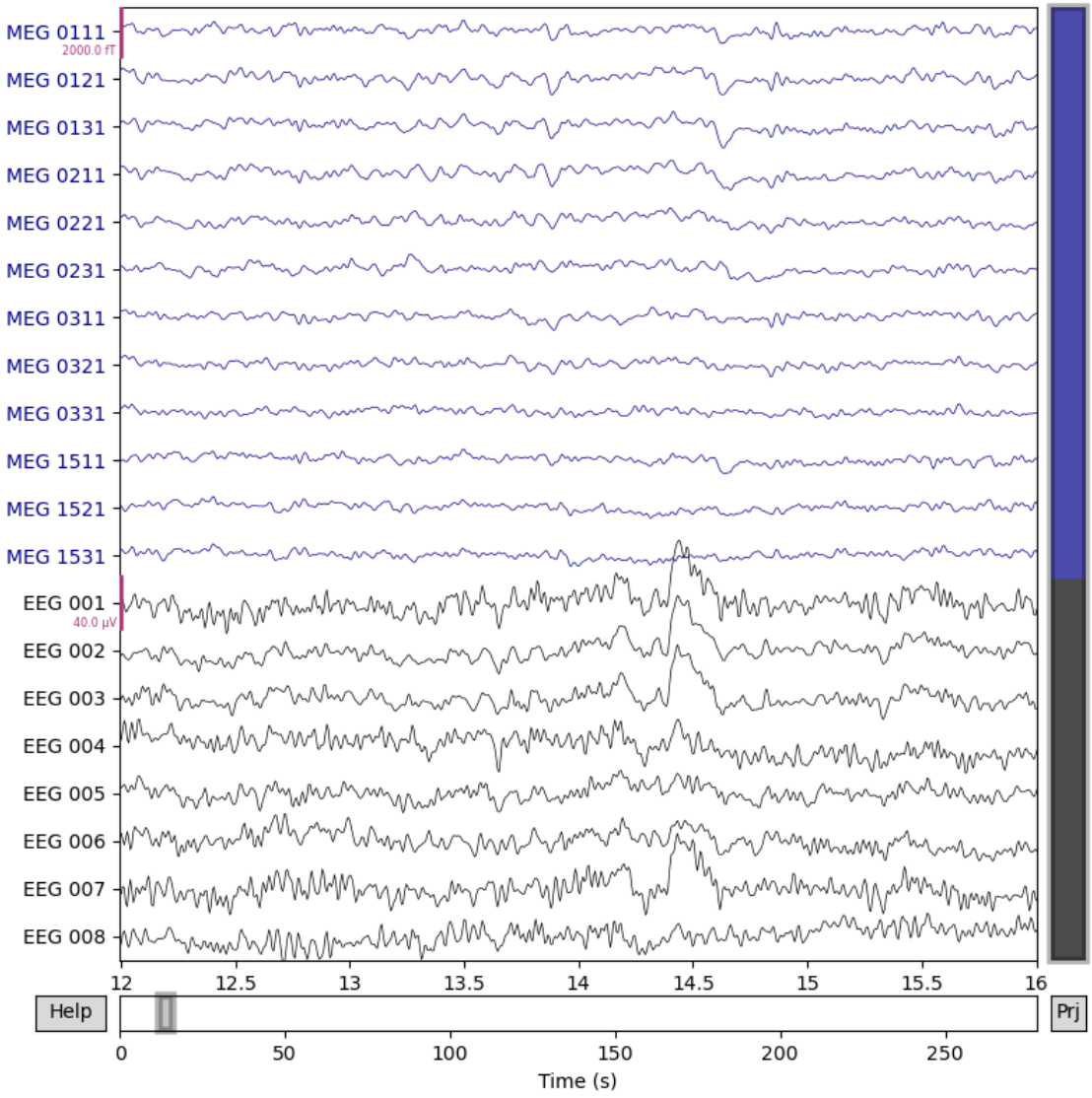
[11]:
```
orig_raw = raw.copy()
raw.load_data()
ica.apply(raw)

# 指定一些通道，用于观察剔除前后的差异
chs = [
    "MEG 0111",
```

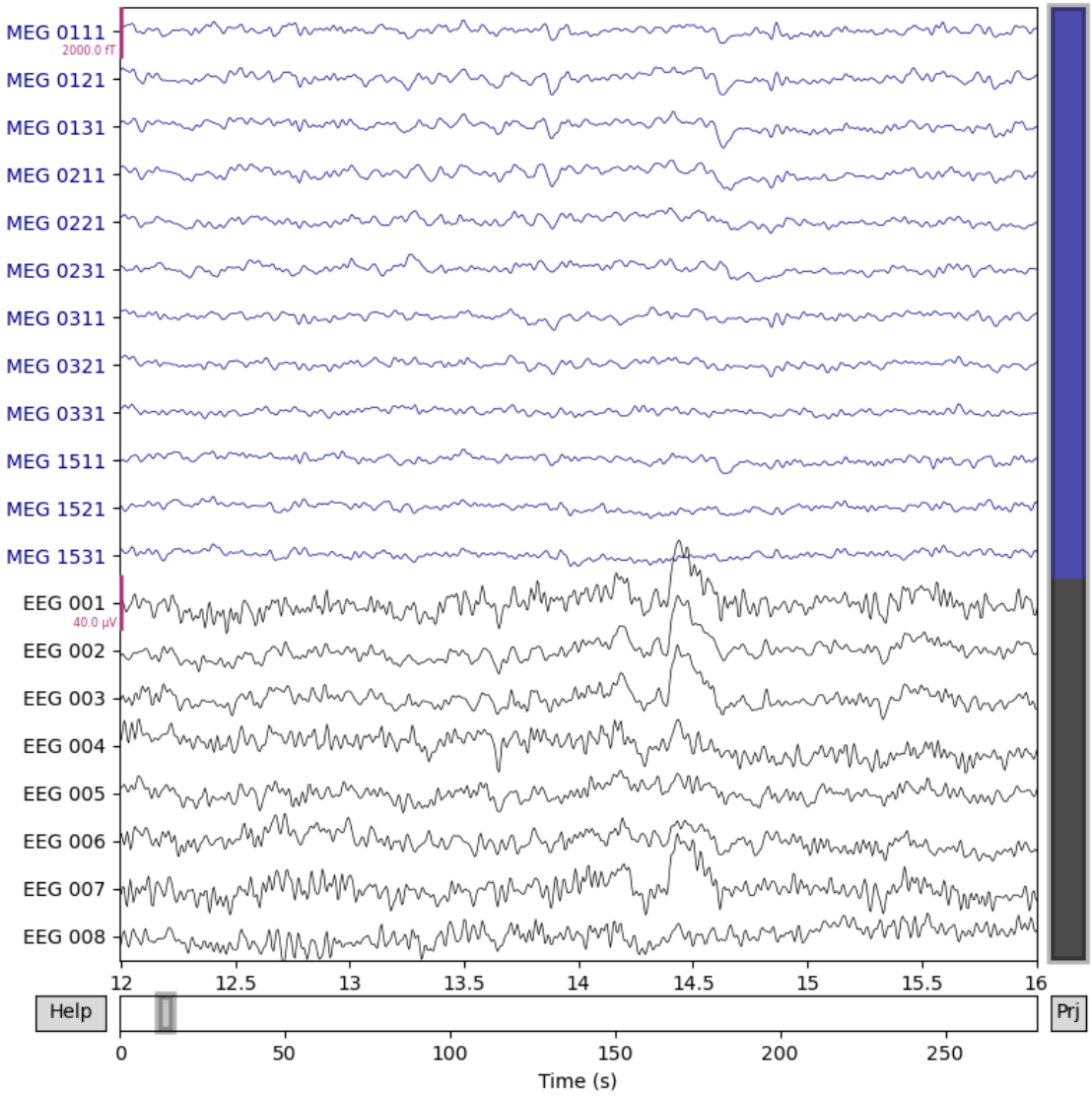
```
    "MEG 0121",
    "MEG 0131",
    "MEG 0211",
    "MEG 0221",
    "MEG 0231",
    "MEG 0311",
    "MEG 0321",
    "MEG 0331",
    "MEG 1511",
    "MEG 1521",
    "MEG 1531",
    "EEG 001",
    "EEG 002",
    "EEG 003",
    "EEG 004",
    "EEG 005",
    "EEG 006",
    "EEG 007",
    "EEG 008",
]
chan_idxs = [raw.ch_names.index(ch) for ch in chs]
orig_raw.plot(order=chan_idxs, start=12, duration=4)
raw.plot(order=chan_idxs, start=12, duration=4)
```

```
Reading 0 … 41699  =      0.000 …   277.709 secs…
Applying ICA to Raw instance
    Transforming to ICA space (20 components)
    Zeroing out 2 ICA components
    Projecting back using 364 PCA components
```


MEG 0111
2000.0 fT
MEG 0121
MEG 0131
MEG 0211
MEG 0221
MEG 0231
MEG 0311
MEG 0321
MEG 0331
MEG 1511
MEG 1521
MEG 1531
EEG 001
40.0 µV
EEG 002
EEG 003
EEG 004
EEG 005
EEG 006
EEG 007
EEG 008
12
12.5
13
13.5
14
14.5
15
15.5
16
Help
Prj
0
50
100
150
200
250
Time (s)


MEG 0111
2000.0 fT
MEG 0121
MEG 0131
MEG 0211
MEG 0221
MEG 0231
MEG 0311
MEG 0321
MEG 0331
MEG 1511
MEG 1521
MEG 1531
EEG 001
40.0 µV
EEG 002
EEG 003
EEG 004
EEG 005
EEG 006
EEG 007
EEG 008
12
12.5
13
13.5
14
14.5
15
15.5
16
Help
Prj
0
50
100
150
200
250
Time (s)

[11]:




# 5 实验事件

在不同类型的刺激（stimulus）发出时，计算机对这些刺激进行了记录。这些刺激被整合到了样例数据的 STI 014 通道中。

[12]:
```
events = mne.find_events(raw, stim_channel="STI 014")
print(events[:5])  # 查看前 5 个事件
```

```
319 events found on stim channel STI 014
Event IDs: [ 1  2  3  4  5 32]
[[6994    0    2]
 [7086    0    3]
 [7192    0    1]
 [7304    0    4]
 [7413    0    2]]
```

从上面的输出结果中可以看到，事件数组是一个 3 列的 NumPy array，第 1 列为样本编号，第 3 列为事件 ID，而第 2 列通常可以忽略。

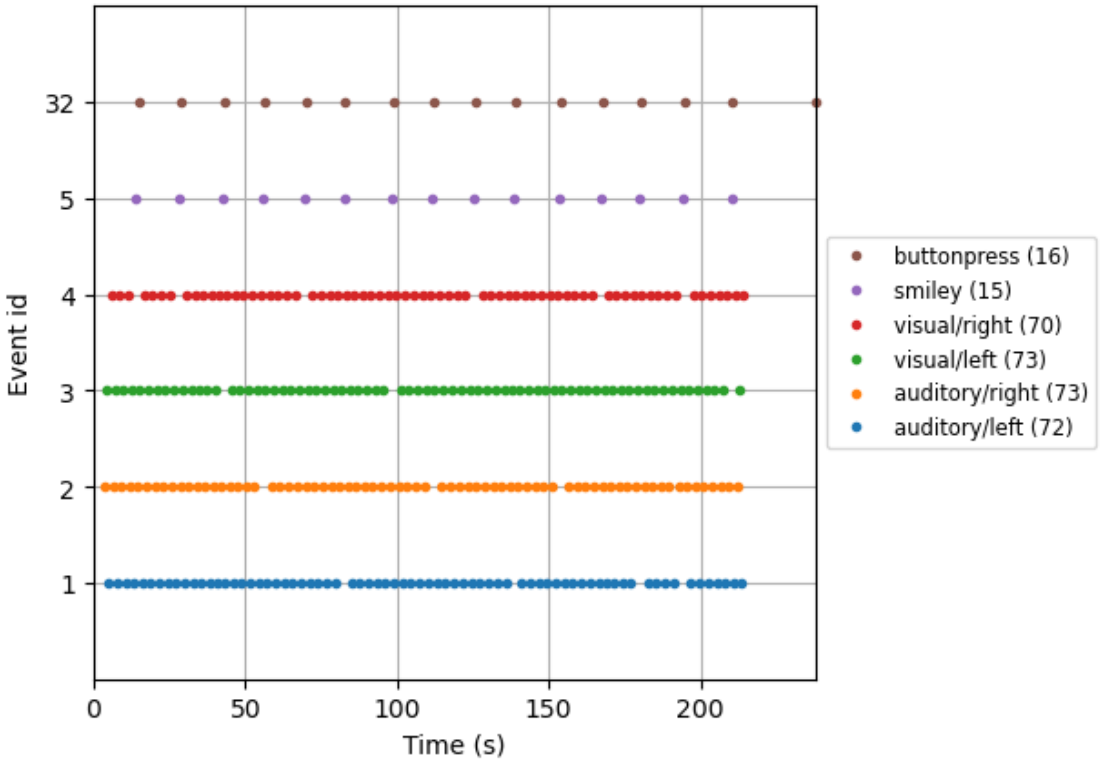
本实验中，事件与事件 ID 的对应关系如以下 dict 所示：

- 1：左耳听觉刺激；
- 2：右耳听觉刺激；
- 3：左视野视觉刺激；
- 4：右视野视觉刺激；
- 5：视野出现微笑面孔图样；
- 6：按动按钮。

[13]:
```
event_dict = {
    "auditory/left": 1,
    "auditory/right": 2,
    "visual/left": 3,
    "visual/right": 4,
    "smiley": 5,
    "buttonpress": 32,
}
```

通过以下方法，我们可以查看不同事件发生的时间及频率。

[14]:
```
fig = mne.viz.plot_events(
    events, event_id=event_dict, sfreq=raw.info["sfreq"], first_samp=raw.
 ↪first_samp
)
```




# 6 Epoch 数据的创建

为了更加方便地进行数据分析，MNE-Python 包提供了 Epochs 类型。该类型的数据可以基于 Raw 对象进行创建。

首先，我们定义一个数据过滤条件，以保证数据质量。这里指定了数据的最大振幅，我们期望一旦超出该范围，数据将被剔除。

[15]:
```
reject_criteria = dict(
    mag=4000e-15,  # 4000 fT
    grad=4000e-13,  # 4000 fT/cm
    eeg=150e-6,  # 150 µV
    eog=250e-6,  # 250 µV
)
```

接下来，通过 mne.Epochs 指定事件字典、过滤条件、相对时间等，创建 Epochs 对象。

[16]:
```
epochs = mne.Epochs(
    raw,
    events,
    event_id=event_dict,
    tmin=-0.2,
    tmax=0.5,
    reject=reject_criteria,
    preload=True,
)
```

```
Not setting metadata
319 matching events found
Setting baseline interval to [-0.19979521315838786, 0.0] s
Applying baseline correction (mode: mean)
Created an SSP operator (subspace dimension = 4)
4 projection items activated
Using data from preloaded Raw for 319 events and 106 original time points …
    Rejecting  epoch based on EOG : ['EOG 061']
    Rejecting  epoch based on EOG : ['EOG 061']
    Rejecting  epoch based on MAG : ['MEG 1711']
    Rejecting  epoch based on EOG : ['EOG 061']
    Rejecting  epoch based on EOG : ['EOG 061']
    Rejecting  epoch based on MAG : ['MEG 1711']
    Rejecting  epoch based on EEG : ['EEG 008']
    Rejecting  epoch based on EOG : ['EOG 061']
    Rejecting  epoch based on EOG : ['EOG 061']
    Rejecting  epoch based on EOG : ['EOG 061']
10 bad epochs dropped
```

指定我们关注的事件类型，同时通过 `equalize_event_counts` 进行事件时间对齐（例如，两类事件发生的时间分别为 [1,2,3,4,120,121]、[3.5,4.5,120,121]，那么第一类事件中的 1、2 将被删除）。

[17]:
```
conds_we_care_about = ["auditory/left", "auditory/right", "visual/left",␣
 ↪"visual/right"]
epochs.equalize_event_counts(conds_we_care_about)
aud_epochs = epochs["auditory"]
vis_epochs = epochs["visual"]
```

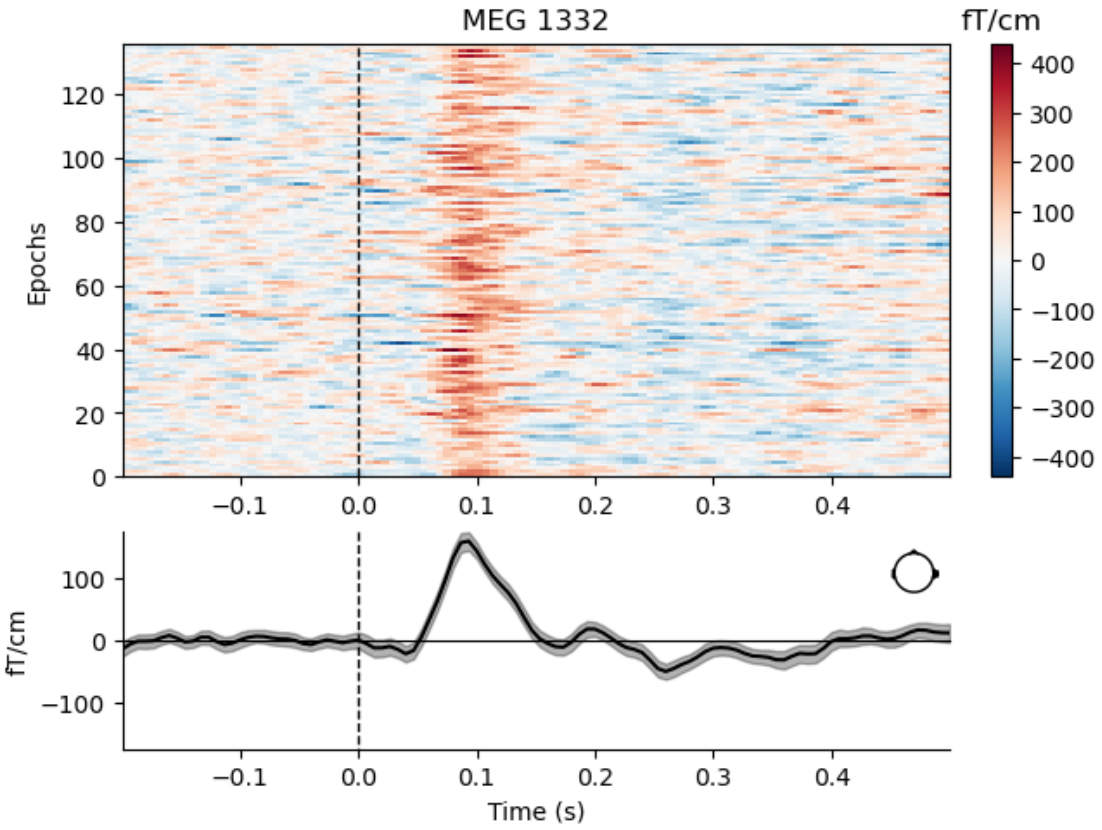
```
del raw, epochs  # 清理内存，防止内存不足
```

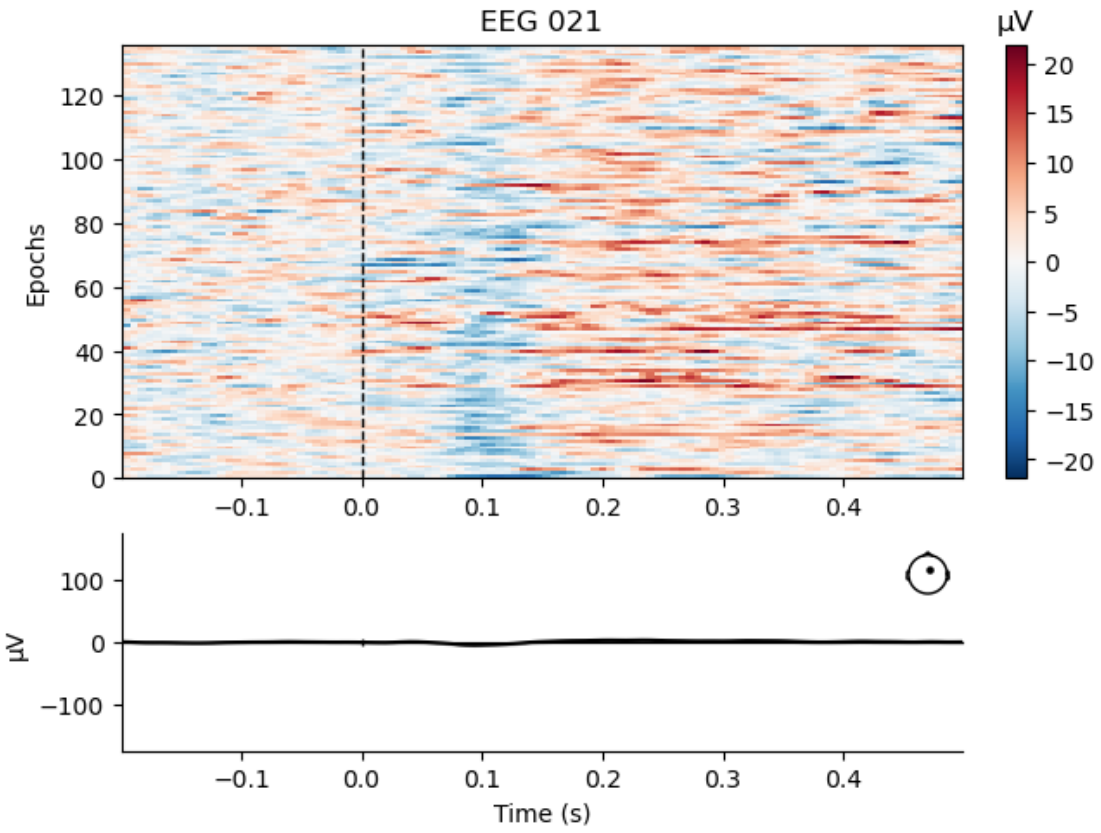
```
Dropped 7 epochs: 121, 195, 258, 271, 273, 274, 275
```

对于 Epochs 类型，我们可以使用 plot_image 方法进行数据查看。

[18]:
```
aud_epochs.plot_image(picks=["MEG 1332", "EEG 021"])
```

```
Not setting metadata
136 matching events found
No baseline correction applied
0 projection items activated
Not setting metadata
136 matching events found
No baseline correction applied
0 projection items activated
```






[18]: [<Figure size 640x480 with 4 Axes>, <Figure size 640x480 with 4 Axes>]

## 7 时频分析

在 mne.time_frequency 子模块中，提供了很多用于时频信号处理的方法。这里我们使用 Morlet 小波方法对听觉刺激的 Epochs 数据进行分析。

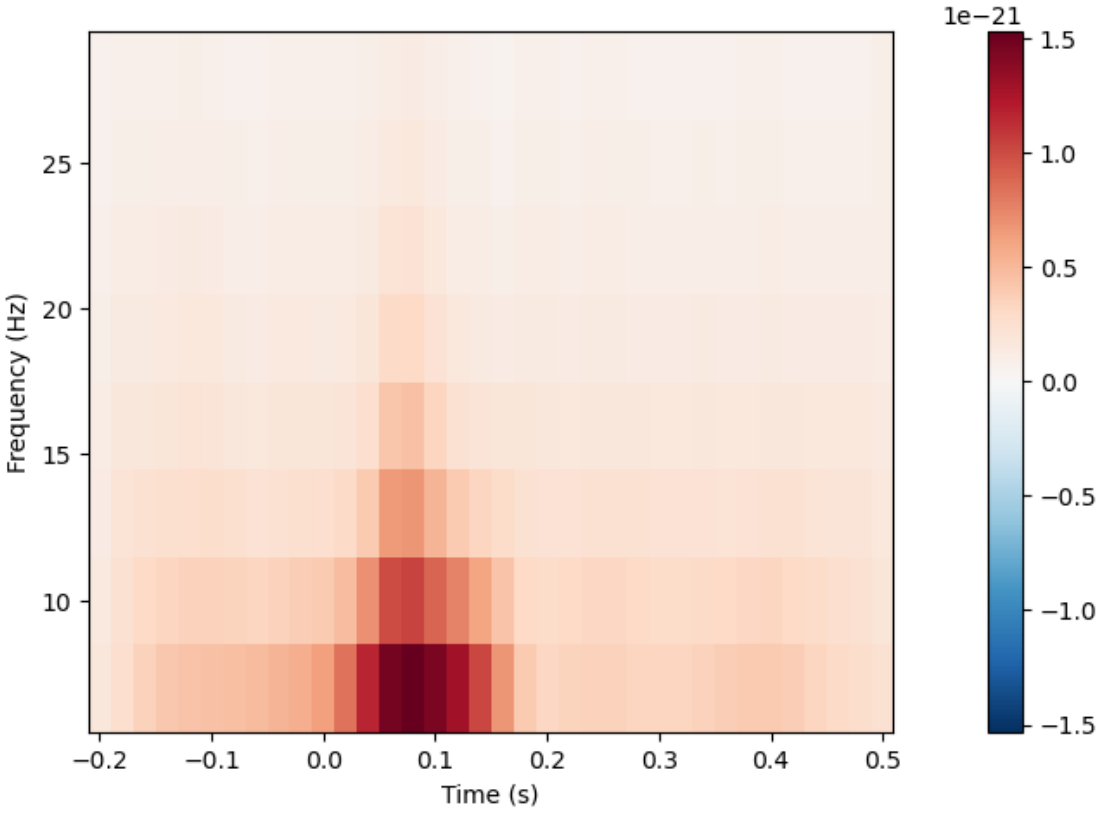
从下面的分析结果中，可以明显看到”auditory N100” 类型的刺激反应。

[19]:
```
frequencies = np.arange(7, 30, 3)
power = mne.time_frequency.tfr_morlet(
    aud_epochs, n_cycles=2, return_itc=False, freqs=frequencies, decim=3
)
power.plot(["MEG 1332"])
```

```
[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

Removing projector <Projection | PCA-v1, active : True, n_channels : 102>
Removing projector <Projection | PCA-v2, active : True, n_channels : 102>
Removing projector <Projection | PCA-v3, active : True, n_channels : 102>
Removing projector <Projection | Average EEG reference, active : True,
n_channels : 60>
Removing projector <Projection | PCA-v1, active : True, n_channels : 102>
Removing projector <Projection | PCA-v2, active : True, n_channels : 102>
Removing projector <Projection | PCA-v3, active : True, n_channels : 102>
Removing projector <Projection | Average EEG reference, active : True,
n_channels : 60>
No baseline correction applied

[Parallel(n_jobs=1)]: Done 364 out of 364 | elapsed:    5.3s finished
```



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

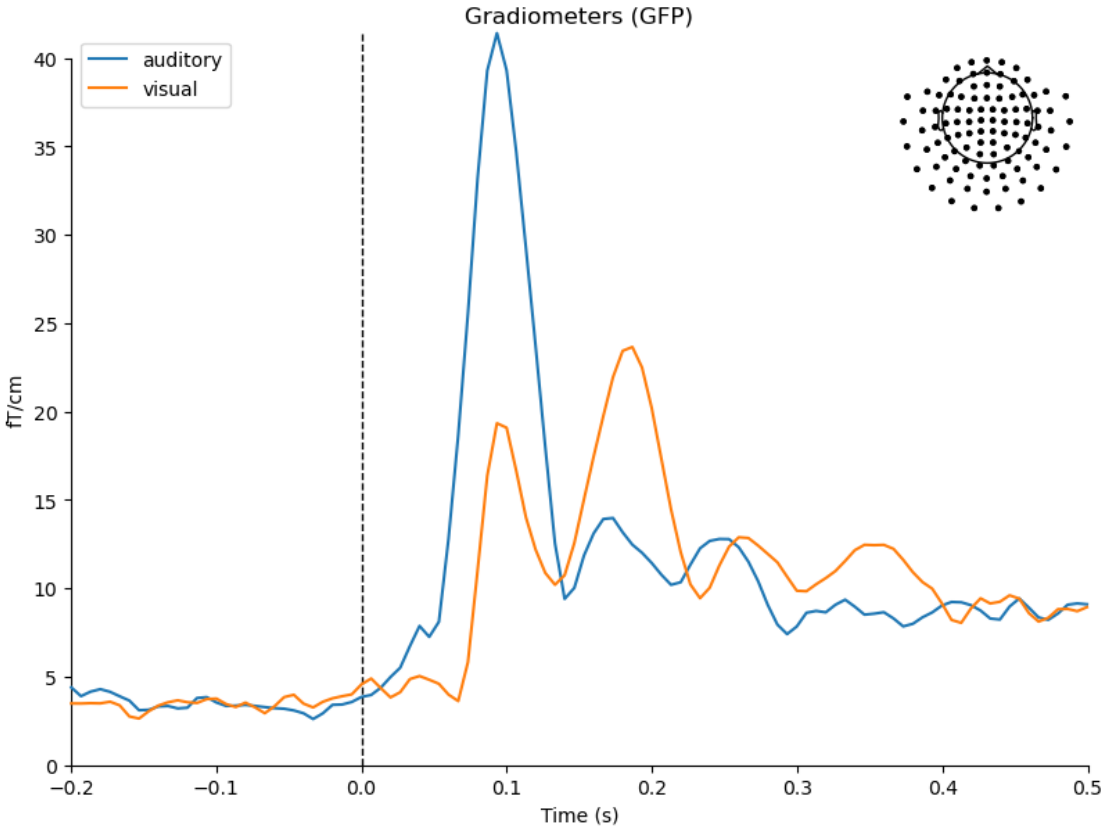
# 8 估计诱发反应

在本实验类型中，存在视觉、听觉等多个类型的刺激。我们可以使用 `Evoked` 对象对不同类型刺激的诱发反应（Evoked responses）进行分析评估。首先通过 `average` 方法计算各种传感器的总体场功率（Global field power，GFP），之后通过 `mne.viz.plot_compare_evokeds` 方法进行可视化。

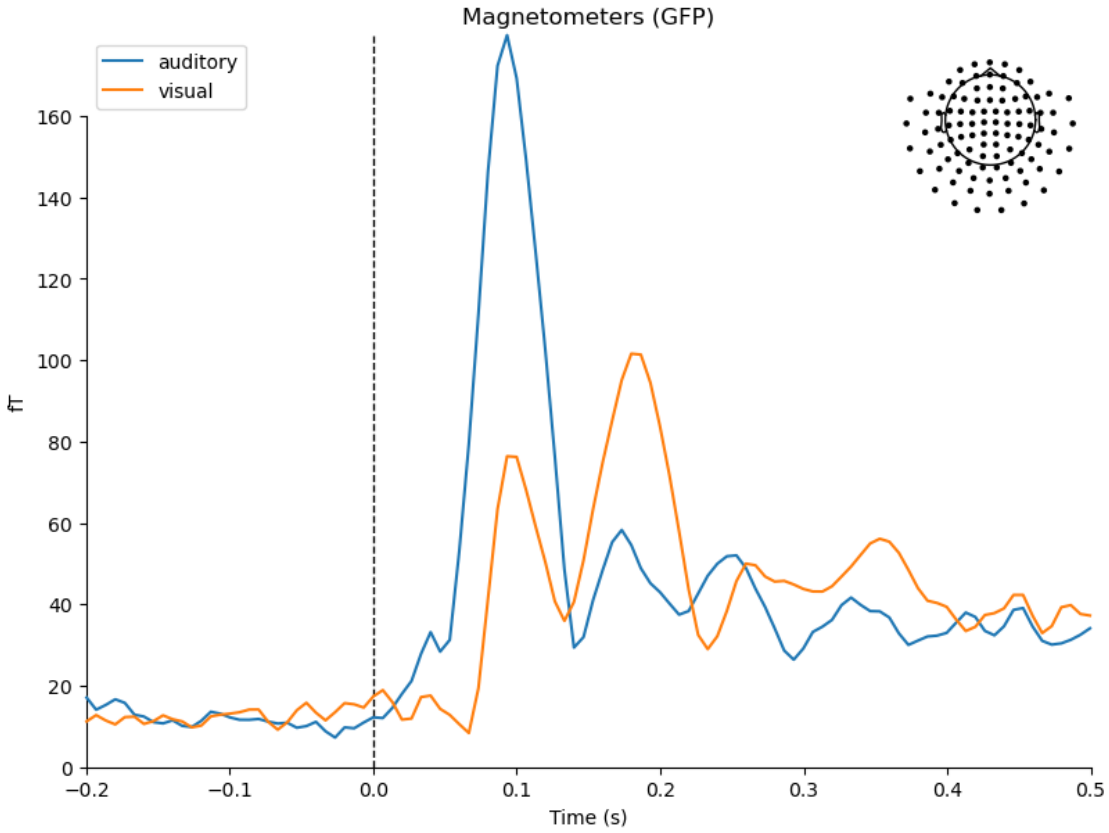
[20]:
```
aud_evoked = aud_epochs.average()
vis_evoked = vis_epochs.average()

mne.viz.plot_compare_evokeds(
    dict(auditory=aud_evoked, visual=vis_evoked),
    legend="upper left",
    show_sensors="upper right",
)
```

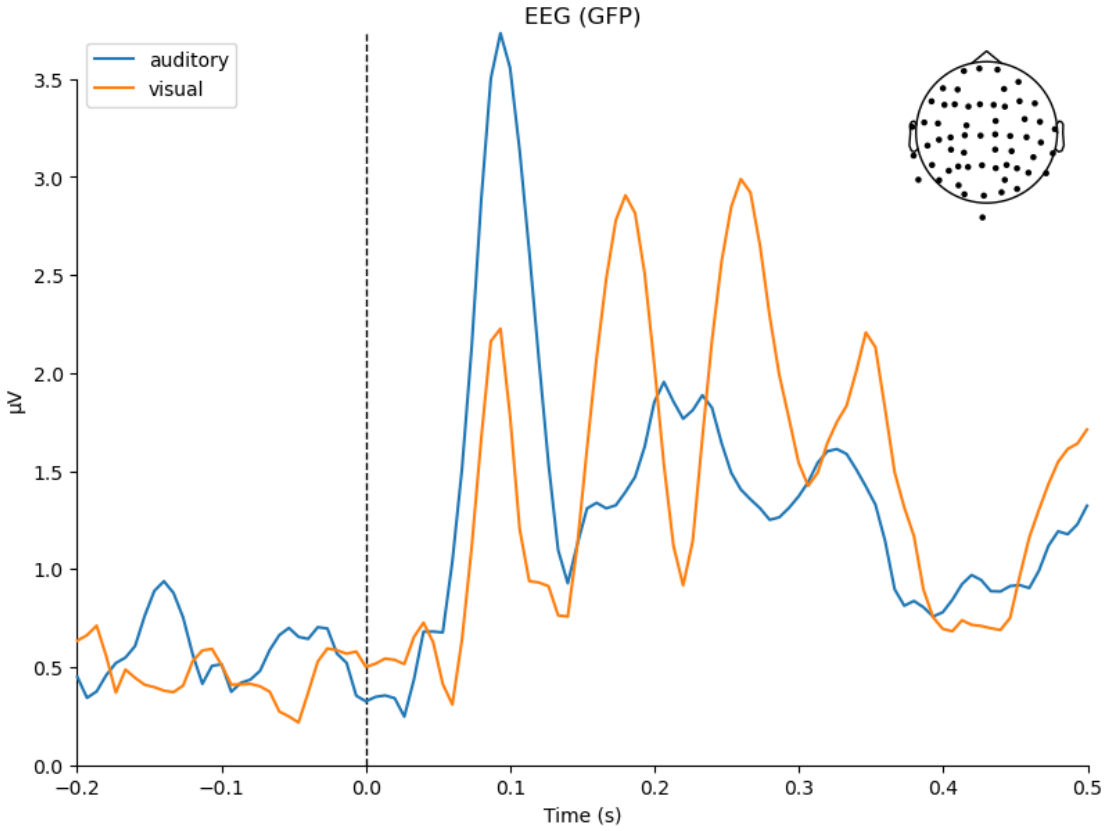
```
Multiple channel types selected, returning one figure per type.
combining channels using "gfp"
combining channels using "gfp"
```




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




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



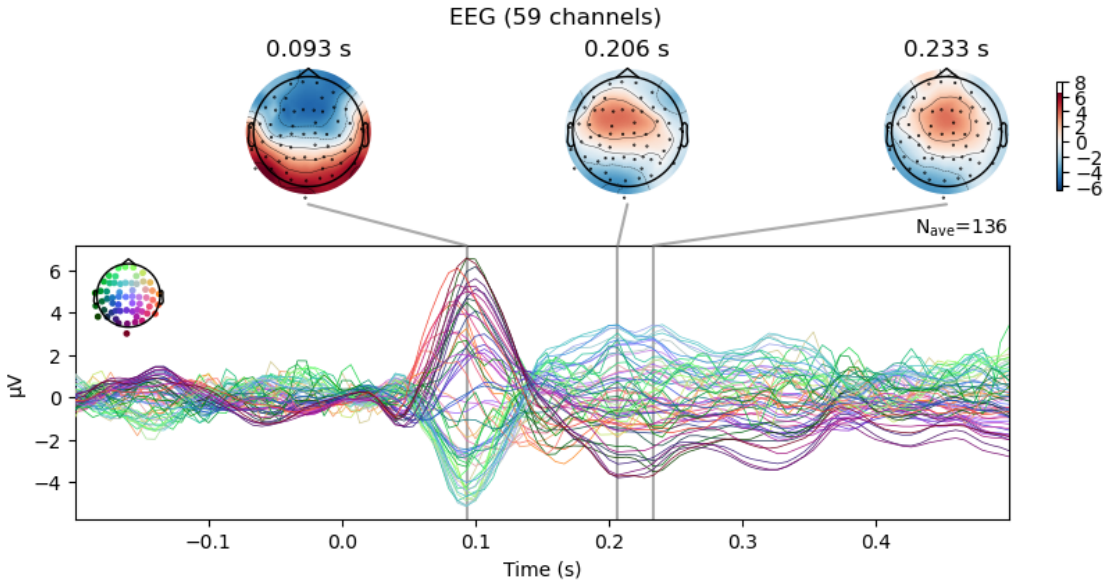

[20]:
```
[<Figure size 800x600 with 2 Axes>,
 <Figure size 800x600 with 2 Axes>,
 <Figure size 800x600 with 2 Axes>]
```

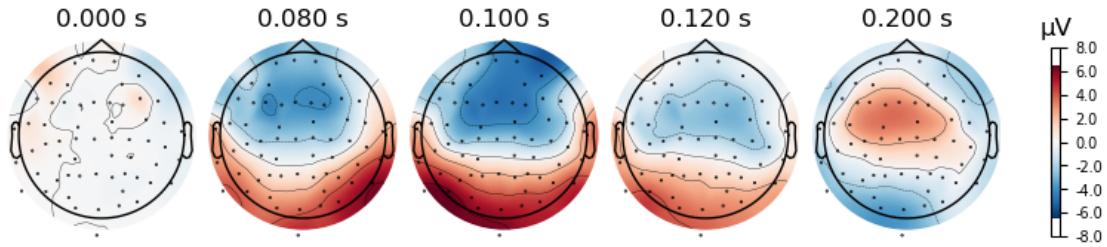
使用 `plot_joint` 或 `plot_topomap` 可以进行更加详细的可视化分析。此处我们仅使用 EEG 数据。

从图中可以观察到，在背外侧额叶存在明显的听觉诱发 N100-P200 成分。

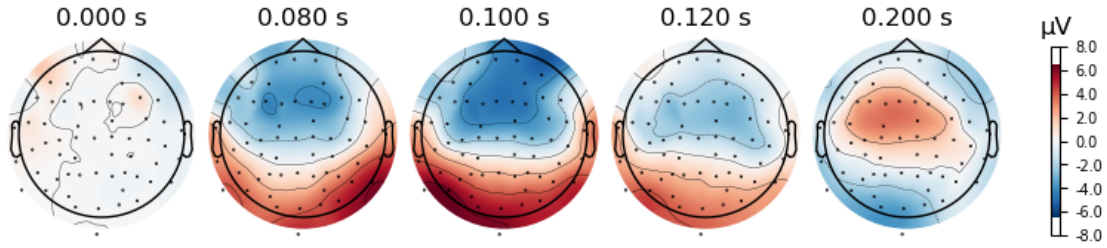
[21]:
```
aud_evoked.plot_joint(picks="eeg")
aud_evoked.plot_topomap(times=[0.0, 0.08, 0.1, 0.12, 0.2], ch_type="eeg")
```

```
Projections have already been applied. Setting proj attribute to True.
Removing projector <Projection | PCA-v1, active : True, n_channels : 102>
Removing projector <Projection | PCA-v2, active : True, n_channels : 102>
Removing projector <Projection | PCA-v3, active : True, n_channels : 102>
```



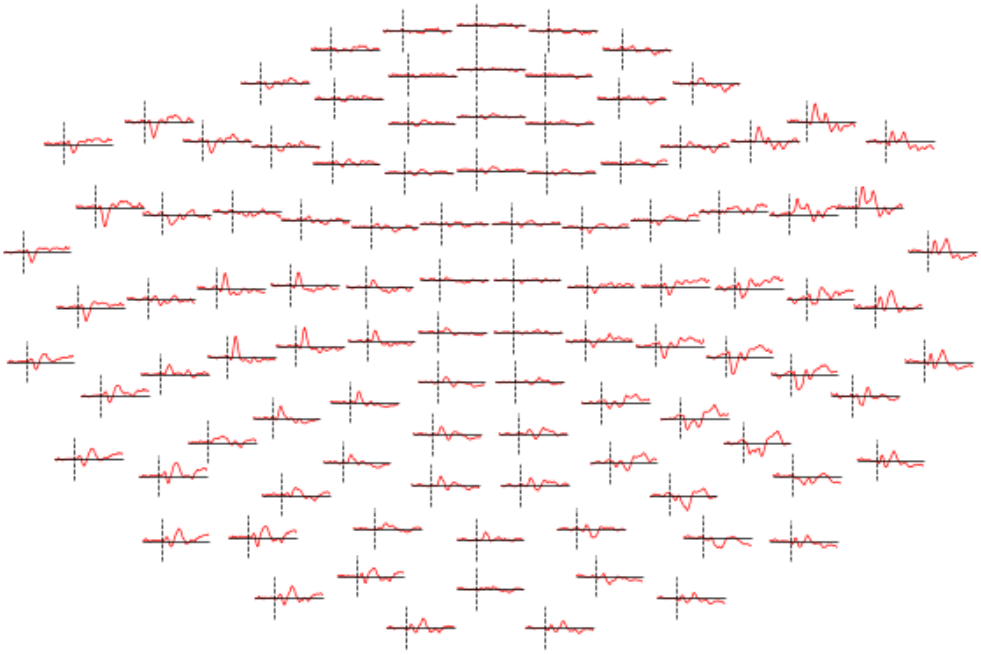

[21]:



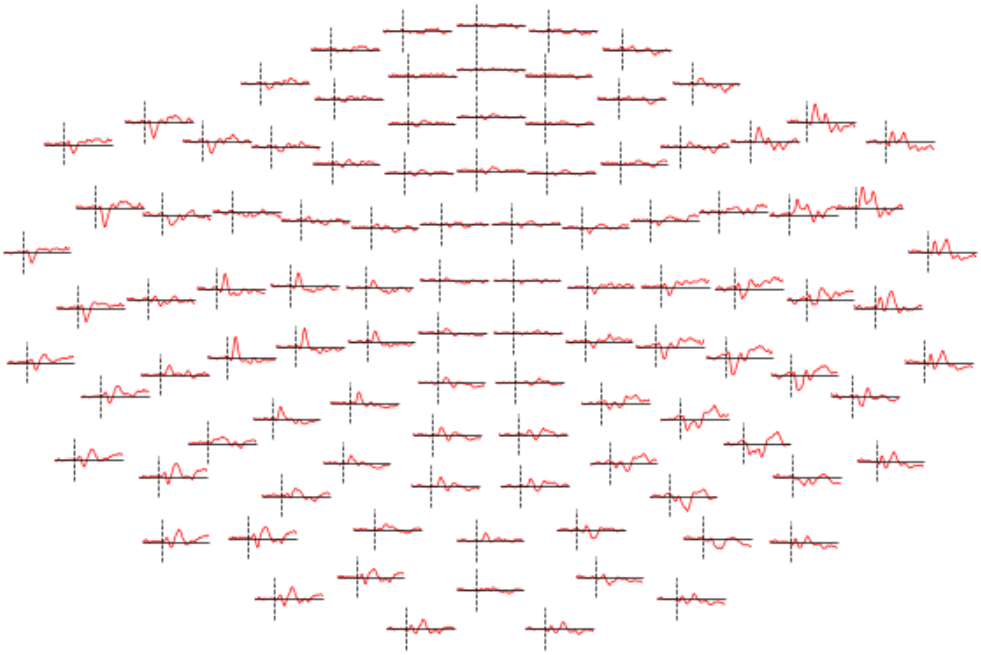

对于 `aud_evoked` 和 `vis_evoked`，我们可以传入 `[1, -1]` 权重，得到两者的绝对差异，并通过下面的方法可视化。

[22]:
```
evoked_diff = mne.combine_evoked([aud_evoked, vis_evoked], weights=[1, -1])
evoked_diff.pick(picks="mag").plot_topo(color="r", legend=False)
```

```
Removing projector <Projection | Average EEG reference, active : True,
n_channels : 60>
```



[22]:



# 9 源分析（Source Analysis）

一般认为，EEG/MEG 的信号来源是脑神经元上的突触后电位，这些突触电流的同步活动导致了脑内的电流流动。由于头部是导电介质，容积传导允许这些电流传播到头皮表面，在那里它们会在放置在头皮不同位置的电极之间产生电位差。通过使用电极阵列记录这些电位，可以构建地形图，显示活跃神经元群在任何给定时刻产生的头皮电位分布。如果只有一个脑区活动，则电位分布的评估是简单直观的。然而，如果大脑的几个区域同时活跃，就会出现头皮电位的复杂模式，而推断潜在的来源就成为一项艰巨的任务。

将 EEG/MEG 测量信号映射到源空间（Source space）（即大脑皮层的活动，可通过 MRI 进行测量），从而估计脑活动，实现**高时间分辨率**的电生理成像，是源分析要达到的目标。这个过程需要一个逆向算子（Inverse operator），本实验直接使用一个已构建好的算子；之后，使用最小范数估计（MNE）方法，进行源分析并得到分析结果。

[23]:
```
# 加载逆向算子
inverse_operator_file = (
    sample_data_folder / "MEG" / "sample" / "sample_audvis-meg-oct-6-meg-inv.
 ↪fif"
)
inv_operator = mne.minimum_norm.read_inverse_operator(inverse_operator_file)

# 设置信噪比 (SNR) 用于计算正则化参数 (λ²)
snr = 3.0
lambda2 = 1.0 / snr**2

# 生成源时程 (STC)
stc = mne.minimum_norm.apply_inverse(
    vis_evoked, inv_operator, lambda2=lambda2, method="MNE"
)
```

```
Reading inverse operator decomposition from D:\Code\cogsci\ex1\mne_data\MNE-
sample-data\MEG\sample\sample_audvis-meg-oct-6-meg-inv.fif…
    Reading inverse operator info…
    [done]
    Reading inverse operator decomposition…
    [done]
    305 x 305 full covariance (kind = 1) found.
    Read a total of 4 projection items:
        PCA-v1 (1 x 102) active
        PCA-v2 (1 x 102) active
        PCA-v3 (1 x 102) active
        Average EEG reference (1 x 60) active
    Noise covariance matrix read.
    22494 x 22494 diagonal covariance (kind = 2) found.
    Source covariance matrix read.
    22494 x 22494 diagonal covariance (kind = 6) found.
    Orientation priors read.
    22494 x 22494 diagonal covariance (kind = 5) found.
    Depth priors read.
    Did not find the desired covariance matrix (kind = 3)
    Reading a source space…
```

```
    Computing patch statistics…
    Patch information added…
    Distance information added…
    [done]
    Reading a source space…
    Computing patch statistics…
    Patch information added…
    Distance information added…
    [done]
    2 source spaces read
    Read a total of 4 projection items:
        PCA-v1 (1 x 102) active
        PCA-v2 (1 x 102) active
        PCA-v3 (1 x 102) active
        Average EEG reference (1 x 60) active
    Source spaces transformed to the inverse solution coordinate frame
Preparing the inverse operator for use…
    Scaled noise and source covariance from nave = 1 to nave = 136
    Created the regularized inverter
    Created an SSP operator (subspace dimension = 3)
    Created the whitener using a noise covariance matrix with rank 302 (3 small
eigenvalues omitted)
Applying inverse operator to "0.50 × visual/left + 0.50 × visual/right"…
    Picked 305 channels from the data
    Computing inverse…
    Eigenleads need to be weighted …
    Computing residual…
    Explained  70.2% variance
    Combining the current components…
[done]
```

[24]:
```
# 被试 MRI 测量结果
subjects_dir = sample_data_folder / "subjects"

# 对源时程 (STC) 进行可视化
stc.plot(
```

```
    initial_time=0.1, hemi="split", views=["lat", "med"],␣
 ↪subjects_dir=subjects_dir
)
```

```
Using control points [8.61922423e-11 1.06837855e-10 4.49139511e-10]
For automatic theme detection, "darkdetect" has to be installed! You can install
it with `pip install darkdetect`
For automatic theme detection, "darkdetect" has to be installed! You can install
it with `pip install darkdetect`
For automatic theme detection, "darkdetect" has to be installed! You can install
it with `pip install darkdetect`
For automatic theme detection, "darkdetect" has to be installed! You can install
it with `pip install darkdetect`
```

[24]: <mne.viz._brain._brain.Brain at 0x243a83e0a60>