
Simultaneous EEG-fMRI

Contents

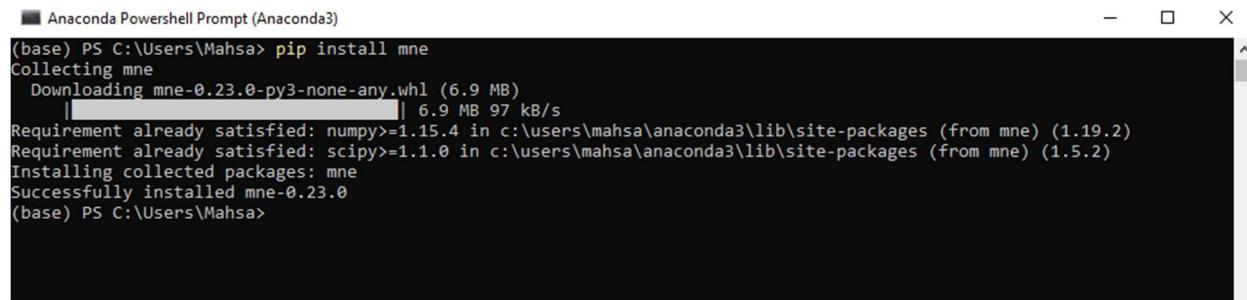
Step 2: denoise the EEG dataset	2
a) Load EEG dataset into python using mne	2
b) Remove gradient and ballistocardiogram artifacts (figure 3).....	5
a. Use average artifact subtraction for both (Allen et al. 2000 see moodle)	5
c) Run ICA (figure 5) on EEG signal to isolate the alpha band components (figure 6).....	6
a. can use the mne implementation of fastICA	6
d) Find the alpha band ICA components by visualizing the topography and power spectrum (figure 7).7	
e) Apply 8-13 Hz bandpass filter to alpha-band component(s) (to isolate alpha), rectify, and smooth..	9
a. There could be multiple components with good alpha, in that case, average them (after rectifying)	9
Step 3: denoise the fMRI dataset. The fMRI dataset must be motion corrected and bandpass filtered:..	10
a) Load fMRI dataset into python using nibabel.....	10
b) Motion correct by registering every volume to the first volume	12
a. find some python library that can do this or create your own, can also do this using FSL or AFNI or some other external library (then load the corrected image in python).....	12
c) Apply 0.01 – 0.1 Hz bandpass filter to the time series in each voxel.....	13
a. This will remove all frequencies outside 0.01 – 0.1 Hz	13
Step 4: combine datasets. After completing the pre-processing for both datasets (step 2,3) it is time to combine the datasets and get our final result:.....	13
a) Resample the smooth, rectified 8-13 Hz bandpass filtered component (step 2e) to 1 Hz (to match fMRI).	13
b) Correlate EEG alpha power (8-13 Hz) with fMRI in each voxel, yielding a 106x106x16 image where the value in each voxel is a correlation coefficient.....	14
a. It might help to also bandpass filter the rectified, resampled EEG alpha between 0.01 – 0.1 Hz..	14
c) Show resulting correlation map overlayed on the T1 image (figure 8).	14
d) Perform cluster-based multiple comparison testing to eliminate spurious clusters	15
Dataset (from openneuro, if you choose to do the bonus)	16

Step 2: denoise the EEG dataset

a) Load EEG dataset into python using mne

In order to load EEG, I firstly did “pip install mne”.

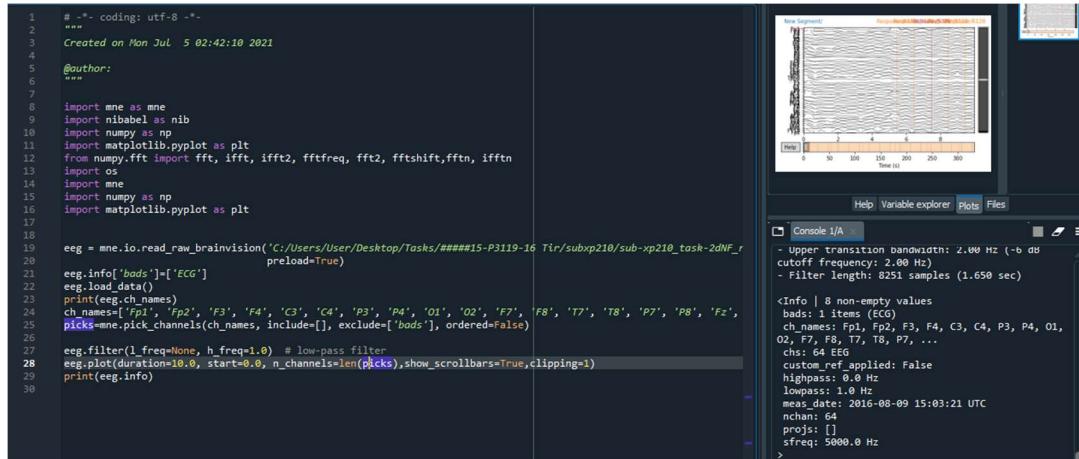
MNE is minimum-norm estimate or minimum-norm estimation. MNE generate a distributed map of activation on a source space, usually on a cortical surface. It also uses a linear inverse operator to project sensor measurements into the source space.



```
Anaconda Powershell Prompt (Anaconda3)
(base) PS C:\Users\Mahsa> pip install mne
Collecting mne
  Downloading mne-0.23.0-py3-none-any.whl (6.9 MB)
    |████████| 6.9 MB 97 kB/s
Requirement already satisfied: numpy>=1.15.4 in c:\users\mahsa\anaconda3\lib\site-packages (from mne) (1.19.2)
Requirement already satisfied: scipy>=1.1.0 in c:\users\mahsa\anaconda3\lib\site-packages (from mne) (1.5.2)
Installing collected packages: mne
Successfully installed mne-0.23.0
(base) PS C:\Users\Mahsa>
```

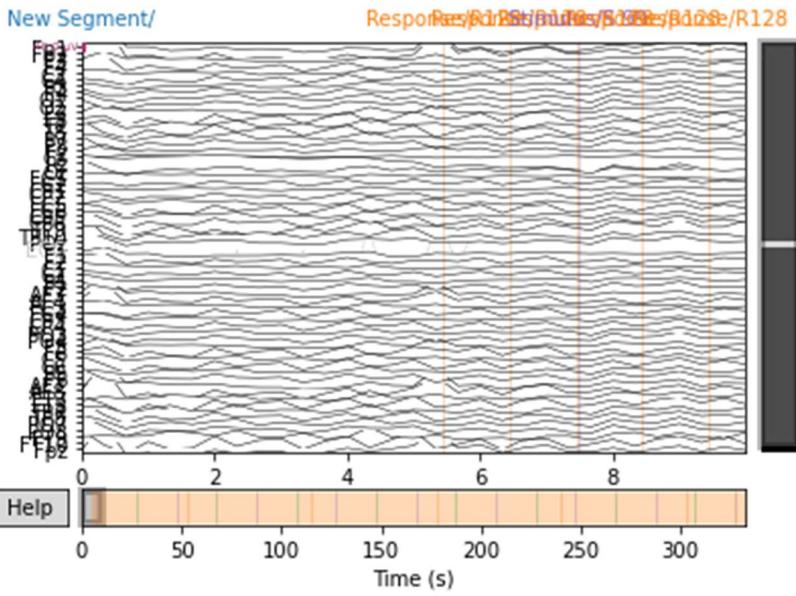
I was noticed that the 32th channel as ECG is not to be considered and I tried to remove it and get all of the rest 63 channels from 63 electrodes. After loading data into memory I got channel names by the aid of eeg.ch_names, I excluded bad channel as ‘ECG’ in this data and filtered it equal to 1 due to fMRI.

Then I just took 10 seconds from 332.920 and I assigned 1 to clipping in order to have more transparent plot.

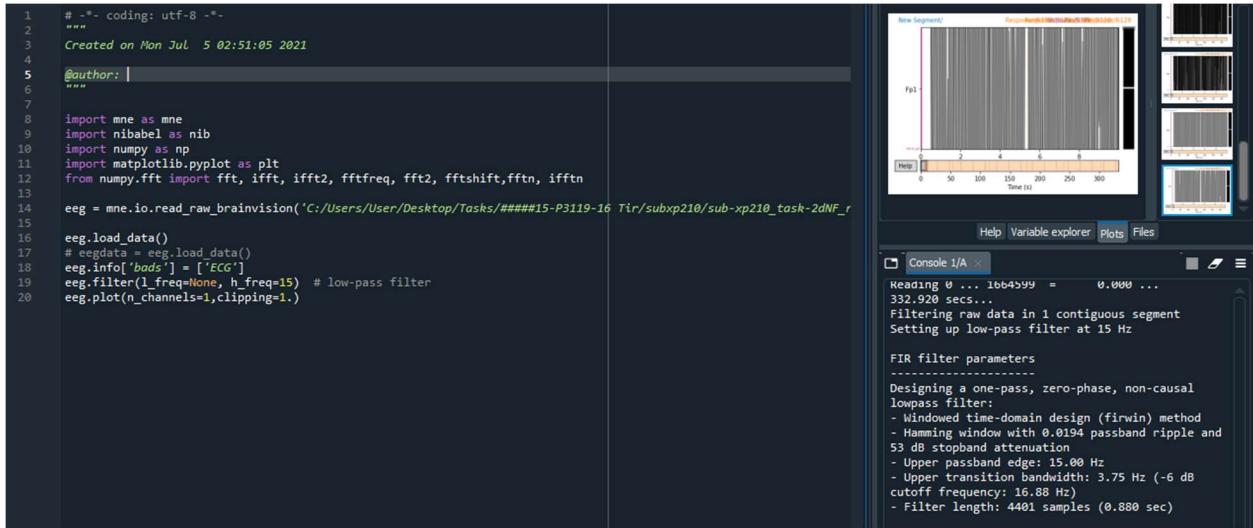


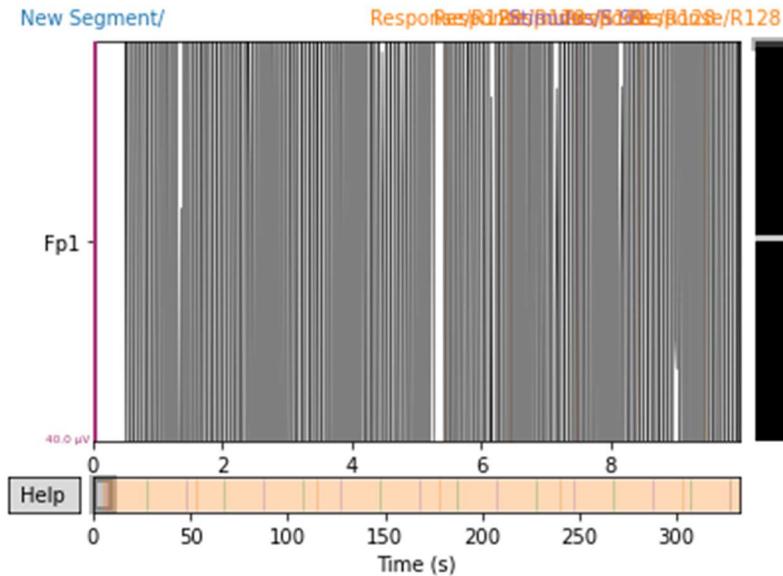
```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Mon Jul  5 02:42:10 2021
4
5  @author:
6  """
7
8  import mne as mne
9  import nibabel as nib
10 import numpy as np
11 import matplotlib.pyplot as plt
12 from scipy import fft import fft, ifft, ifft2, fftfreq, fft2, fftshift, ifftn, ifftn
13 import os
14 import mne
15 import numpy as np
16 import matplotlib.pyplot as plt
17
18
19 eeg = mne.io.read_raw_brainvision('C:/Users/User/Desktop/Tasks/####15-P3119-16_Tir/subxp210/sub-xp210_task-2dNFI_r
20                                     preLoad=True)
21 eeg.info['bads']=['ECG']
22 eeg.load_data()
23 print(eeg.ch_names)
24 ch_names=['Fp1', 'Fp2', 'F3', 'F4', 'C3', 'C4', 'P3', 'P4', 'O1', 'O2', 'F7', 'F8', 'T7', 'T8', 'P7', 'P8', 'Fz',
25 picks=mne.pick_channels(ch_names, include=[], exclude=['bad'], ordered=False)
26
27 eeg.filter(l_freq=None, h_freq=1.0) # low-pass filter
28 eeg.plot(duration=10.0, start=0.0, n_channels=len(picks), show_scrollbars=True, clipping=1)
29 print(eeg.info)
30
```

The screenshot shows a Jupyter Notebook cell with the above Python code. To the right of the cell is a visualization of the EEG data. The plot shows multiple channels of raw EEG signals over time (0 to 10 seconds). Below the plot is a console window displaying the output of the code, including the list of channel names and the resulting filtered EEG object information.



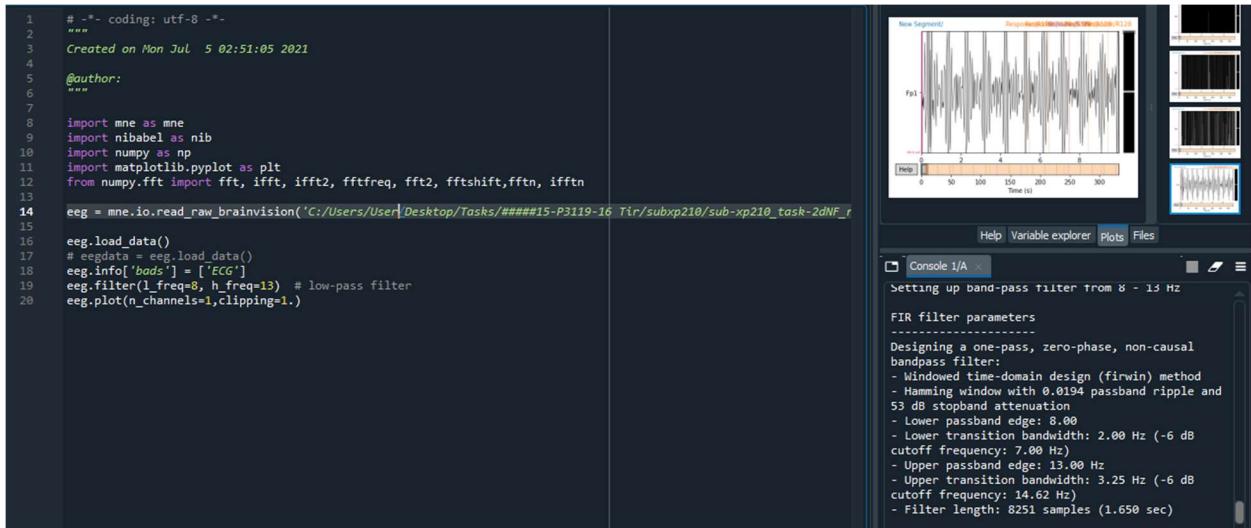
In the another plot, I decided to draw one channel from 64 channel and its name was "Fp1".

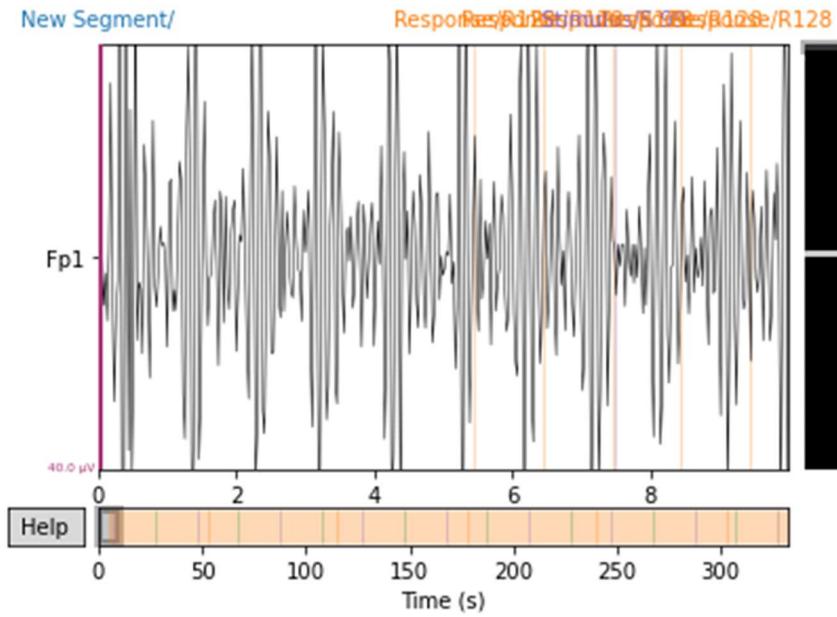




I changed frequency to alpha band which is between 8 to 13 Hz.

```
mne.filter.resample(x, up=1.0, down=1.0, npad=100, axis=- 1, window='boxcar', n_jobs=1, pad='reflect_limited', verbose=None)[source]
```





b) Remove gradient and ballistocardiogram artifacts (figure 3)

- a. Use average artifact subtraction for both (Allen et al. 2000 see moodle)

In order to remove gradient I used reject parameter from mne.Epochs and assigned grad=4000e-13 to remove gradiometers and for BCG or ballistocardiogram I got annotations and removed it from epoch by "reject_by_annotation=True"

```

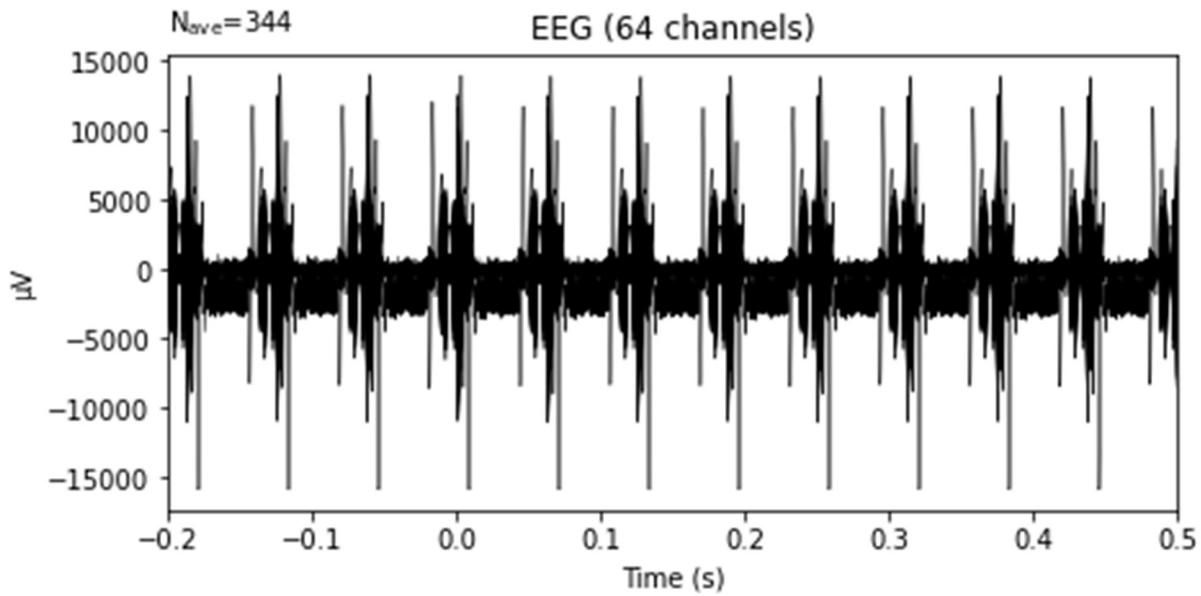
3 Created on Mon Jul  5 06:16:38 2021
4
5 @author:
6 """
7
8 import os
9 import mne
10 from mne.preprocessing import (ICA, create_eog_epochs, create_ecg_epochs,
11                               corrmap)
12 import numpy as np
13
14 raw = mne.io.read_raw_brainvision('C:/Users/Mahsa/Desktop/Tasks/#####15-P3119-16 Tir/subxp210/sub-xp210_task-2d'
15 events=mne.events_from_annotations(raw)
16 print(mne.events_from_annotations(raw))
17
18 reject = dict(grad=4000e-13, # unit: T / m (gradiometers) Gradient Remove
19               mag=4e-12, # unit: T (magnetometers)
20               eeg=40e-6, # unit: V (EEG channels)
21               eog=250e-6 # unit: V (EOG channels)
22             )
23
24 epochs = mne.Epochs(raw, events[0], tmin=-0.2, tmax=0.5,
25                      reject_by_annotation=True)
26 evoked = epochs.average() # compute evoked
27 evoked.plot() # butterfly plot the evoked data
28
29
30
31
32
33
34
35
36
37
38

```

```

99, 'Stimulus/S101': 1128,
  (array([[ 0,  0,  0,  99999,
   [ 27206,  0,  1128],
   [ 32206,  0,  1128],
   ...,
   [1652206,  0,  1128],
   [1657206,  0,  1128],
   [1662206,  0,  1128]]], {'New
Segment/': 99999, 'Response/R128': 1128,
'Stimulus/S 2': 2, 'Stimulus/S 99': 99,
'Stimulus/S101': 101})
Not setting metadata
Not setting metadata
346 matching events found
Setting baseline interval to [-0.2, 0.0] sec
Applying baseline correction (mode: mean)
0 projection items activated

```



c) Run ICA (figure 5) on EEG signal to isolate the alpha band components (figure 6)

a. can use the mne implementation of fastICA

I tried to get events from mne.preprocessing and I plot epochs as follows:

Indeed Independent components analysis (**ICA**) is **used to** take a large data set consisting of many variables and reduce it into smaller number dimensions that can be understood as self-organized functional networks.

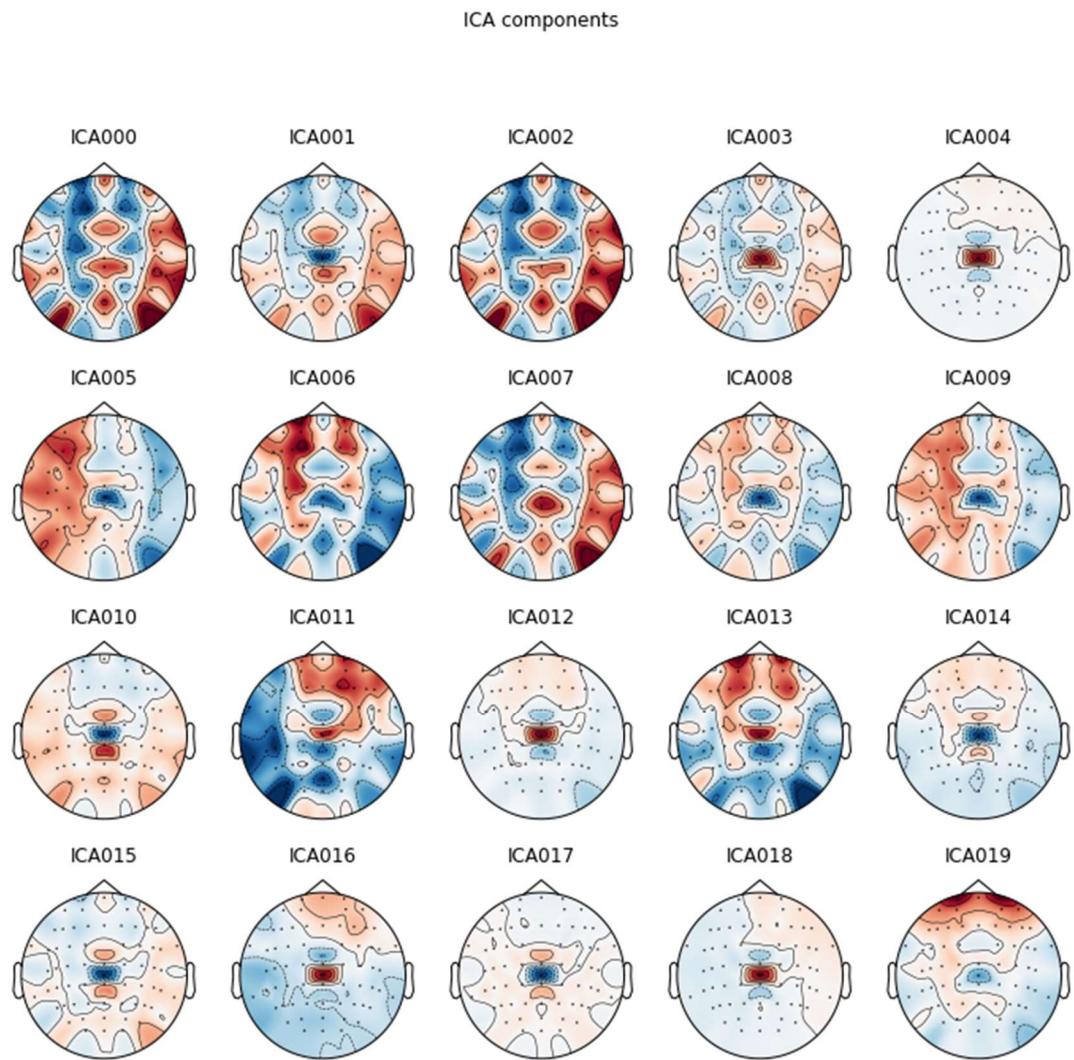
```

1  # -*- coding: utf-8 -*-
2  """
3  Created on Mon Jul  5 07:02:59 2021
4
5  @author:
6  """
7  import os
8  import mne
9  from mne.preprocessing import (ICA, create_eog_epochs, create_ecg_epochs,
10                                 corrmap)
11 import numpy as np
12
13 raw = mne.io.read_raw_brainvision('C:/Users/User/Desktop/Tasks/####15-P3119-16_Tir/subxp210/sub-xp210_task-2dNF_r')
14
15 raw_epochs = mne.preprocessing.find_ecg_events(raw)
16 raw_epochs.plot_image(combine='mean')

```

P3119-16 (Tir/subxp210)
Extracting parameters from C:/Users/Mahsa/Desktop/Tasks/####15-P3119-16_Tir/subxp210/sub-xp210_task-2dNF_run-02_eeg.vhdr...
Setting channel info structure...
Reading 0 ... 1664599 = 0.000 ...
332.920 secs...
Fitting ICA to data using 64 channels (please be patient, this may take a while)
Selecting by number: 20 components
Fitting ICA took 105.9s.

Figures now render in the Plots pane by default.
To make them also appear inline in the Console, uncheck "Mute Inline Plotting" under the Plots pane options menu.



d) Find the alpha band ICA components by visualizing the topography and power spectrum (figure 7).

Data decomposition using Independent Component Analysis (ICA).

This object estimates independent components from `mne.io.Raw`, `mne.Epochs`, or `mne.Evoked` objects. Components can optionally be removed (for artifact repair) prior to signal reconstruction.

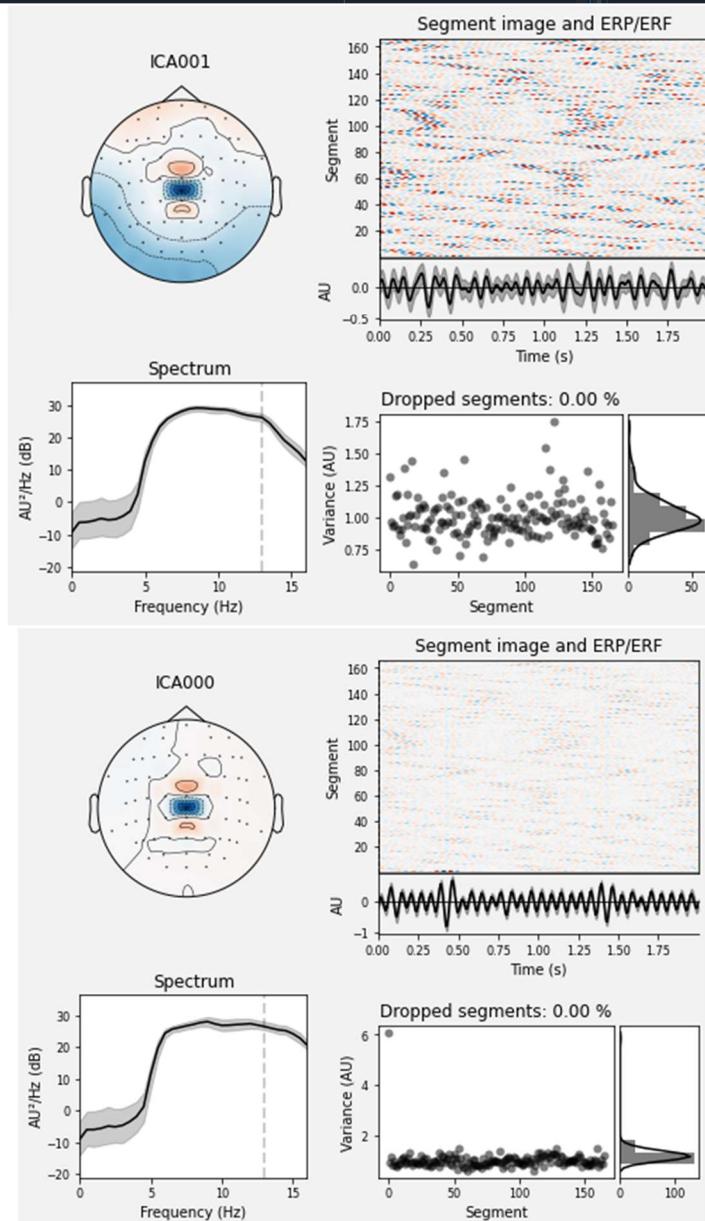
```
mne.preprocessing.ICA(n_components=None, *, noise_cov=None, random_state=None, method='fastica', fit_params=None, max_iter=None, allow_ref_meg=False, verbose=None)
```

```

1  # -*- coding: utf-8 -*-
2  """
3  Created on Mon Jul  5 08:01:38 2021
4
5  @author: |
6
7
8
9  import os
10 import mne
11 from mne.preprocessing import (ICA, create_eog_epochs, create_ecg_epochs,
12                                corrmap)
13 import numpy as np
14
15 raw = mne.io.read_raw_brainvision('C:/Users/User/Desktop/Tasks/#####15-P3119-16_Tir/subxp210/sub-xp210_task-2dNF_r'
16 ##
17 ica = ICA(n_components=20, max_iter='auto', random_state=97)
18
19 filt_raw = raw.copy()
20 filt_raw.load_data().filter(l_freq=8, h_freq=13)
21 filt_raw.set_montage('standard_1020', on_missing='ignore')
22
23 ica.fit(filt_raw)
24 # ica.plot_components()
25
26 ica.plot_properties(filt_raw, picks=[0, 1]) # It is possible [0, 1, 2, ..., 18, 19]

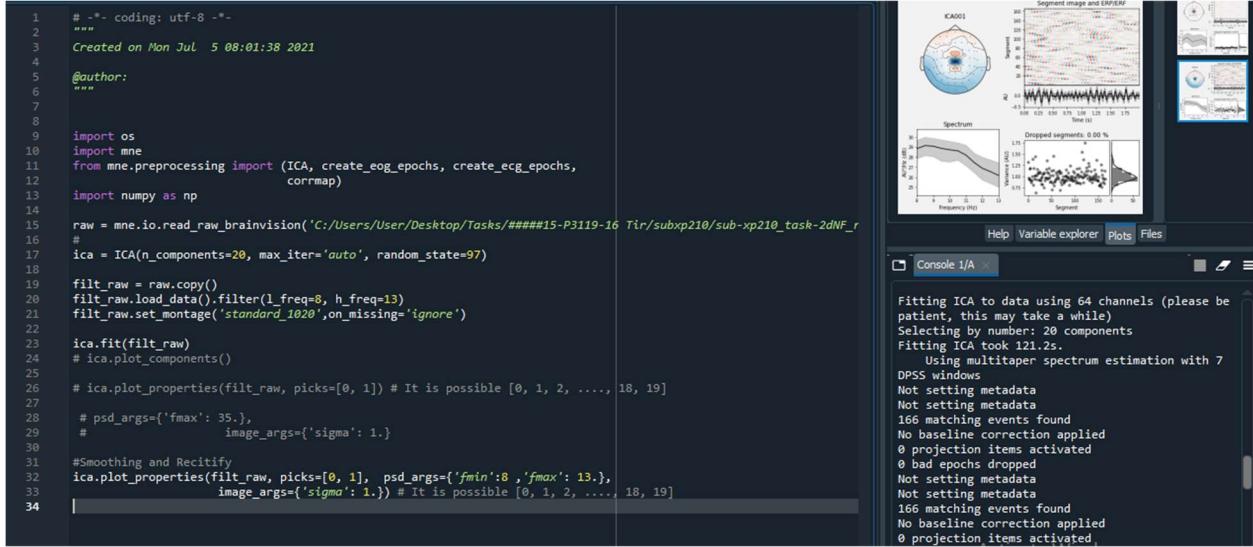
```

Fitting ICA took 159.1s.
Using multitaper spectrum estimation with 7 DPSS windows
Not setting metadata
Not setting metadata
166 matching events found
No baseline correction applied
0 projection items activated
0 bad epochs dropped
Not setting metadata
Not setting metadata
166 matching events found
No baseline correction applied
0 projection items activated
0 bad epochs dropped



e) Apply 8-13 Hz bandpass filter to alpha-band component(s) (to isolate alpha), rectify, and smooth.

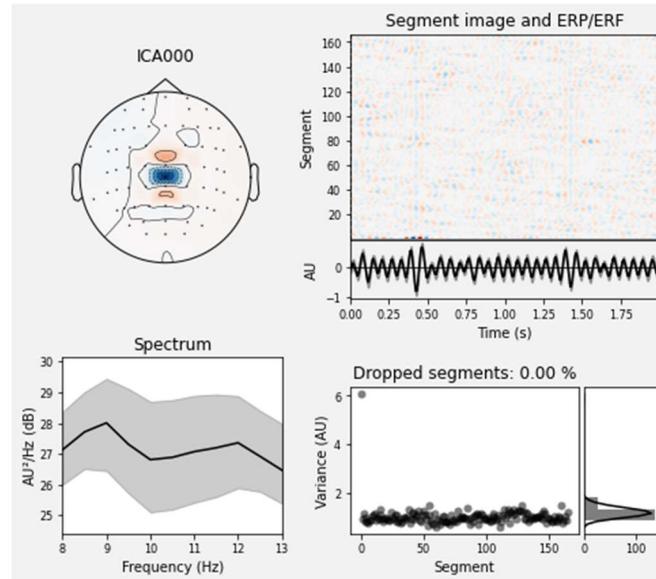
a. There could be multiple components with good alpha, in that case, average them (after rectifying)
I used raw.set_montage for having best parameters to detect sensor locations. Also for having more smoothing plot I used image_args={'sigma':1.}]

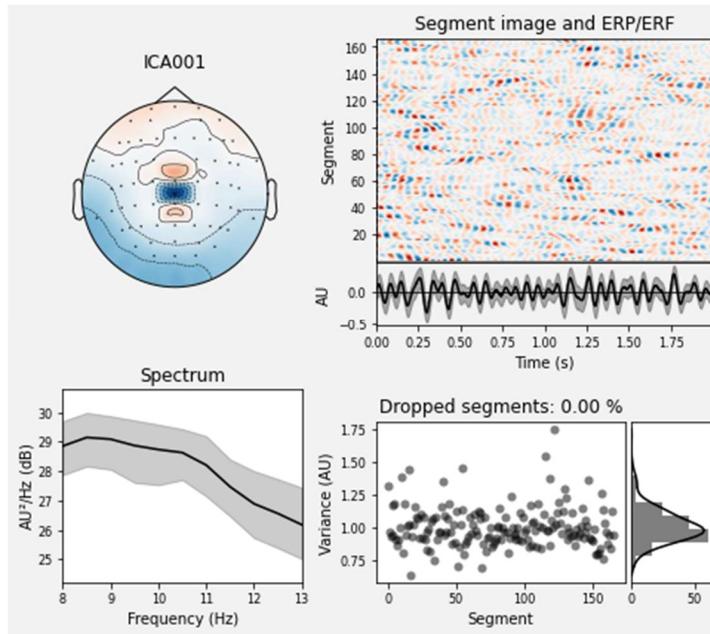


```

1  # -*- coding: utf-8 -*-
2  """
3  Created on Mon Jul  5 08:01:38 2021
4
5  @author:
6  """
7
8
9  import os
10 import mne
11 from mne.preprocessing import (ICA, create_eog_epochs, create_ecg_epochs,
12                                corrmap)
13 import numpy as np
14
15 raw = mne.io.read_raw_brainvision('C:/Users/User/Desktop/Tasks/#####15-P3119-16 Tir/subxp210/sub-xp210_task-2dNF_r'
16 #
17 ica = ICA(n_components=20, max_iter='auto', random_state=97)
18
19 filt_raw = raw.copy()
20 filt_raw.load_data().filter(l_freq=8, h_freq=13)
21 filt_raw.set_montage('standard_1020', on_missing='ignore')
22
23 ica.fit(filt_raw)
24 # ica.plot_components()
25
26 # ica.plot_properties(filt_raw, picks=[0, 1]) # It is possible [0, 1, 2, ..., 18, 19]
27
28 # psd_args={'fmax': 35.},
29 #           image_args={'sigma': 1.}
30
31 #Smoothing and Rectify
32 ica.plot_properties(filt_raw, picks=[0, 1], psd_args={'fmin':8 , 'fmax': 13.},
33                      image_args={'sigma': 1.}) # It is possible [0, 1, 2, ..., 18, 19]
34

```





Step 3: denoise the fMRI dataset. The fMRI dataset must be motion corrected and bandpass filtered:

- a) Load fMRI dataset into python using nibabel

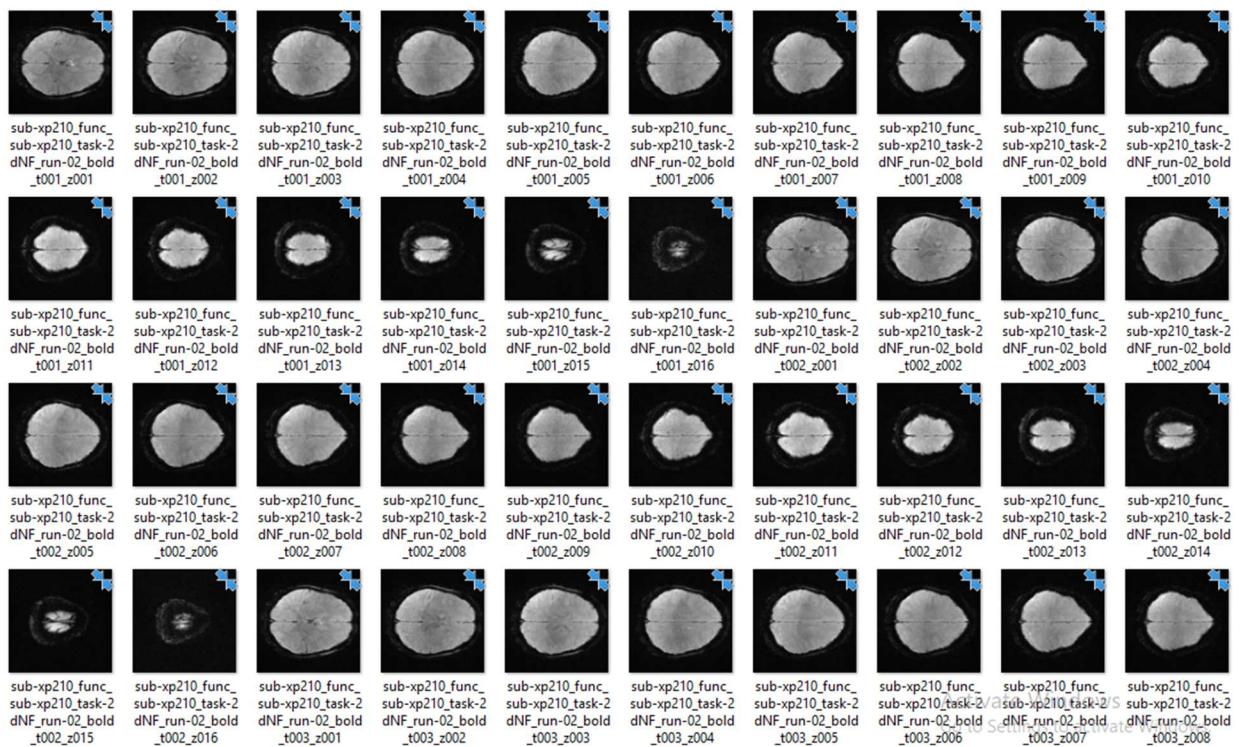
I used nibabel to read nii files and I had to install nibabel into python, “pip install nibabel”.

In fact a nibabel (and numpy) image is the association of three things: 1. An affine array that tells the position of the image array data in a reference space. 2. Image metadata (data about the data) 3. describing the image, usually in the form of an image header.

```

9
10 def main(argv):
11     inputfile = r'C:\Users\Mahsa\Desktop\Tasks\#####15-P3119-16 Tir\nii\sub-xp210_T1w_defaced.nii'
12     outputfile = r'C:\Users\Mahsa\Desktop\Tasks\#####15-P3119-16 Tir\nii\defaced'
13
14
15     # set fn as your 4d nifti file
16     image_array = nibabel.load(inputfile).get_data()
17
18     # ask if rotate
19     ask_rotate = input('Would you like to rotate the orientation? (y/n) ')
20
21     if ask_rotate.lower() == 'y':
22         ask_rotate_num = int(input('OK. By 90° 180° or 270°? '))
23         if ask_rotate_num == 90 or ask_rotate_num == 180 or ask_rotate_num == 270:
24             print('Got it. Your images will be rotated by {} degrees.'.format(ask_rotate_num))
25         else:
26             print('You must enter a value that is either 90, 180, or 270. Quitting...')
27             sys.exit()
28
29     elif ask_rotate.lower() == 'n':
30         print('OK. Your images will be converted it as it is.')
31     else:
32         print('You must choose either y or n. Quitting...')
33         sys.exit()
34
35     # if 4D image inputted
36     if len(image_array.shape) == 4:
37         # set 4d array dimension values
38         nx, ny, nz, nw = image_array.shape
39
40         # set destination folder
41         if not os.path.exists(outputfile):
42             os.makedirs(outputfile)
43             print("Created ouput directory: " + outputfile)
44
45         print('Reading NIfTI file...')
46
47     total_volumes = image_array.shape[3]
        total_slices = image_array.shape[2]

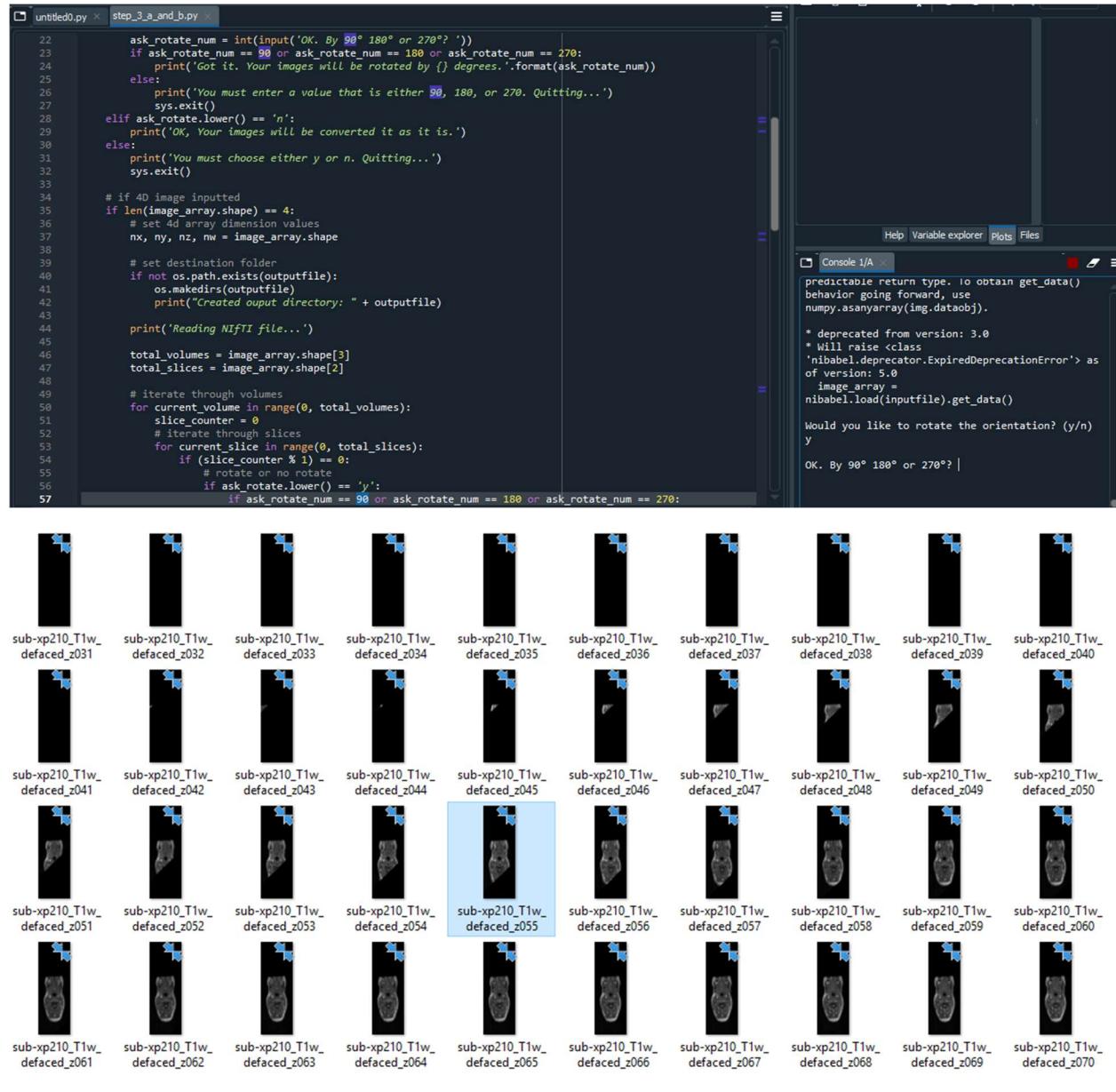
```



b) Motion correct by registering every volume to the first volume

a. find some python library that can do this or create your own, can also do this using FSL or AFNI or some other external library (then load the corrected image in python)

I used numpy.rot90 or nested numpy.rot90(numpy.rot90)=180 or numpy.rot90(numpy.rot90(numpy.rot90))=270 and asked from user to select how much degree is suitable to rotate and adjust image.



The screenshot shows a Jupyter Notebook environment with two panes. The left pane displays a Python script named `step_3_a_and_b.py`. The right pane shows the notebook's interface with a code cell, a console output cell, and a file browser.

```

22 ask_rotate_num = int(input('OK. By 90° 180° or 270°? '))
23 if ask_rotate_num == 90 or ask_rotate_num == 180 or ask_rotate_num == 270:
24     print('Got it. Your images will be rotated by {} degrees.'.format(ask_rotate_num))
25 else:
26     print('You must enter a value that is either 90, 180, or 270. Quitting...')
27     sys.exit()
28 elif ask_rotate.lower() == 'n':
29     print('OK. Your images will be converted it as it is.')
30 else:
31     print('You must choose either y or n. Quitting...')
32     sys.exit()
33
34 # if 4D image inputted
35 if len(image_array.shape) == 4:
36     # set 4d array dimension values
37     nx, ny, nz, nw = image_array.shape
38
39     # set destination folder
40     if not os.path.exists(outputfile):
41         os.makedirs(outputfile)
42         print("Created ouput directory: " + outputfile)
43
44     print('Reading NIfTI file...')
45
46     total_volumes = image_array.shape[3]
47     total_slices = image_array.shape[2]
48
49     # iterate through volumes
50     for current_volume in range(0, total_volumes):
51         slice_counter = 0
52         # iterate through slices
53         for current_slice in range(0, total_slices):
54             if (slice_counter % 1) == 0:
55                 # rotate or no rotate
56                 if ask_rotate.lower() == 'y':
57                     if ask_rotate_num == 90 or ask_rotate_num == 180 or ask_rotate_num == 270:

```

In the notebook's console, the user has run the script and selected to rotate the images. The script then asks if the user wants to rotate the orientation:

```

predictable return type. To obtain get_data()
behavior going forward, use
numpy.asarray(img.dataobj).

* deprecated from version: 3.0
* Will raise <class
'nibabel.deprecator.ExpiredDeprecationError'> as
of version: 5.0
    image_array =
nibabel.load(inputfile).get_data()

Would you like to rotate the orientation? (y/n)
y

OK. By 90° 180° or 270°? |

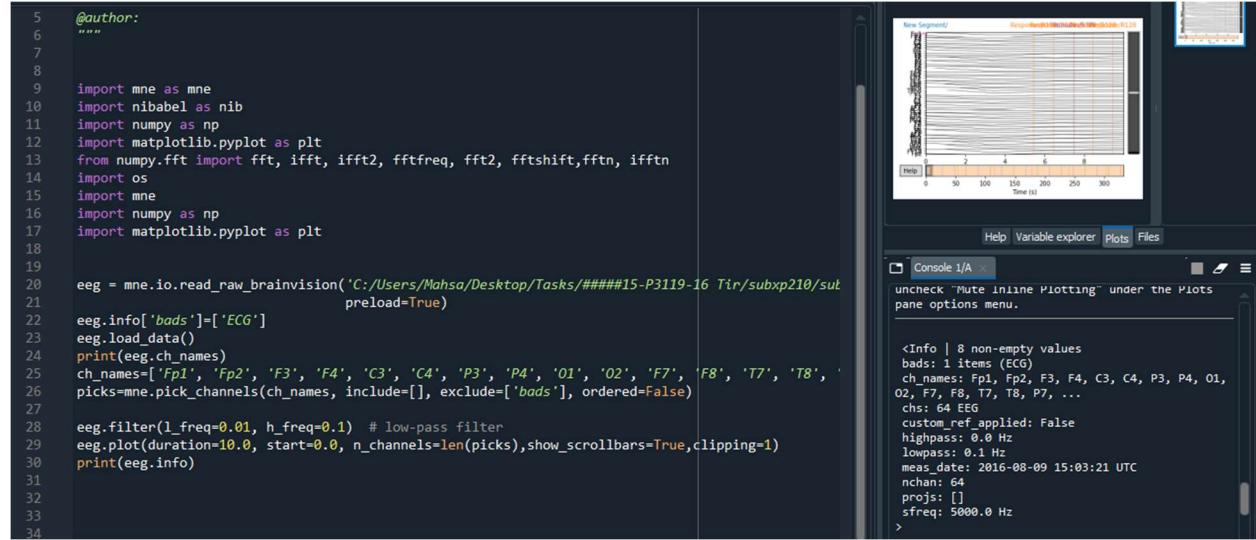
```

Below the code cell, a grid of 60 brain scan images is displayed. The images are labeled with their corresponding file names. The image for volume 55 is highlighted with a light blue background.

sub-xp210_T1w_defaced_z031	sub-xp210_T1w_defaced_z032	sub-xp210_T1w_defaced_z033	sub-xp210_T1w_defaced_z034	sub-xp210_T1w_defaced_z035	sub-xp210_T1w_defaced_z036	sub-xp210_T1w_defaced_z037	sub-xp210_T1w_defaced_z038	sub-xp210_T1w_defaced_z039	sub-xp210_T1w_defaced_z040
sub-xp210_T1w_defaced_z041	sub-xp210_T1w_defaced_z042	sub-xp210_T1w_defaced_z043	sub-xp210_T1w_defaced_z044	sub-xp210_T1w_defaced_z045	sub-xp210_T1w_defaced_z046	sub-xp210_T1w_defaced_z047	sub-xp210_T1w_defaced_z048	sub-xp210_T1w_defaced_z049	sub-xp210_T1w_defaced_z050
sub-xp210_T1w_defaced_z051	sub-xp210_T1w_defaced_z052	sub-xp210_T1w_defaced_z053	sub-xp210_T1w_defaced_z054	sub-xp210_T1w_defaced_z055	sub-xp210_T1w_defaced_z056	sub-xp210_T1w_defaced_z057	sub-xp210_T1w_defaced_z058	sub-xp210_T1w_defaced_z059	sub-xp210_T1w_defaced_z060
sub-xp210_T1w_defaced_z061	sub-xp210_T1w_defaced_z062	sub-xp210_T1w_defaced_z063	sub-xp210_T1w_defaced_z064	sub-xp210_T1w_defaced_z065	sub-xp210_T1w_defaced_z066	sub-xp210_T1w_defaced_z067	sub-xp210_T1w_defaced_z068	sub-xp210_T1w_defaced_z069	sub-xp210_T1w_defaced_z070

c) Apply 0.01 – 0.1 Hz bandpass filter to the time series in each voxel

a. This will remove all frequencies outside 0.01 – 0.1 Hz



Step 4: combine datasets. After completing the pre-processing for both datasets (step 2,3) it is time to combine the datasets and get our final result:

- Resample the smooth, rectified 8-13 Hz bandpass filtered component (step 2e) to 1 Hz (to match fMRI).

A filter removes or attenuates parts of a signal. Usually, filters act on specific *frequency ranges* of a signal — for example, suppressing all frequency components above or below a certain cutoff value. There are *many* ways of designing digital filters;

Artifacts that are restricted to a narrow frequency range can sometimes be repaired by filtering the data. Two examples of frequency-restricted artifacts are slow drifts and power line noise. Here I illustrate how each of these can be repaired by filtering.

```
raw.plot_psd
```

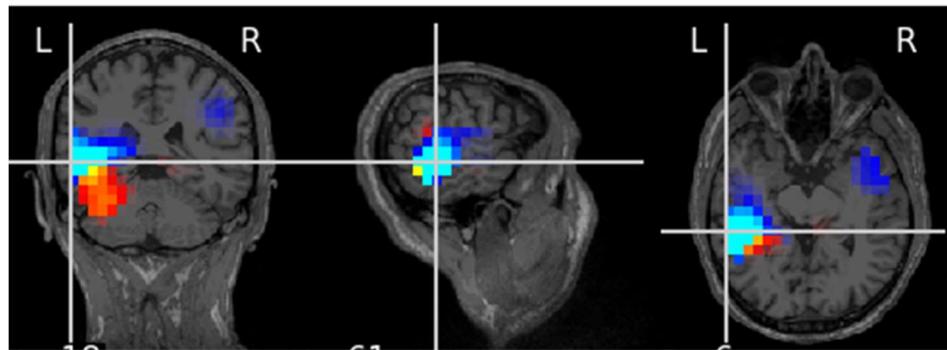
b) Correlate EEG alpha power (8-13 Hz) with fMRI in each voxel, yielding a 106x106x16 image where the value in each voxel is a correlation coefficient.

a. It might help to also bandpass filter the rectified, resampled EEG alpha between 0.01 – 0.1 Hz.

```
24 # Setup for reading the raw data
25 raw = mne.io.read_raw_fif(raw_fname, preload=True)
26 raw.info['bads'] = ['ECG'] # 2 bads channels
27 events = mne.read_events(event_fname)
28
29 # Pick the channels of interest
30 raw.pick(['Fp1'])
31
32 # Read epochs
33 proj = False # already applied
34 epochs = mne.Epochs(raw, events, event_id, tmin, tmax,
35                     baseline=(None, 0), preload=True, proj=proj,
36                     )
37 evoked = epochs.average()
38
39 # Visualize sensor space data
40 evoked.plot_joint()
41
42
43 noise_cov = mne.compute_covariance(epochs, tmin=tmin, tmax=0, method='shrunk',
44                                     rank=None)
45 data_cov = mne.compute_covariance(epochs, tmin=0.04, tmax=0.15,
46                                     method='shrunk', rank=None)
47
48 filters = make_lcmv(evoked.info, forward, data_cov, reg=0.05,
49                      noise_cov=noise_cov, pick_ori='max-power',
50                      weight_norm='nai', rank=None)
51 print(filters)
52
53 stc = apply_lcmv(evoked, filters, max_ori_out='signed')
54
55 clim = dict(kind='value', pos_lims=[0.3, 0.6, 0.9])
56 stc.plot(src=forward['src'], subject='sample', subjects_dir=subjects_dir,
57          clim=clim)
```

b) Show resulting correlation map overlayed on the T1 image (figure 8).

It is the linearly constrained minimum variance (LCMV) beamformer 1 operates on time series. Frequency-resolved data can be reconstructed with the dynamic imaging of coherent sources (DICS) beamforming method 2. As we will see in the following, the spatial filter is computed from two ingredients: the forward model solution and the covariance matrix of the data.



d) Perform cluster-based multiple comparison testing to eliminate spurious clusters

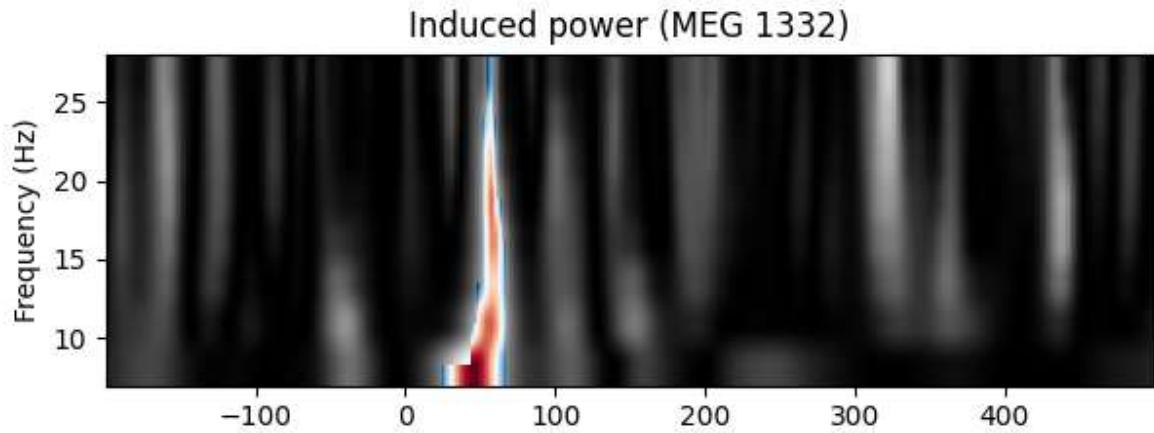
Non-parametric between conditions cluster statistic on single trial power

This script shows how to compare clusters in time-frequency power estimates between conditions. It uses a non-parametric statistical procedure based on permutations and cluster level statistics.

The process contains of:

- extracting epochs for 2 conditions
- compute single trial power estimates
- baseline line correct the power estimates (power ratios)
- compute stats to see if the power estimates are significantly different between conditions.

```
68 threshold = 6.0
69 T_obs, clusters, cluster_p_values, H0 = \
70     permutation_cluster_test([epochs_power_1, epochs_power_2], out_type='mask',
71     n_permutations=100, threshold=threshold, tail=0)
72
73 times = 1e3 * epochs_condition_1.times # change unit to ms
74 evoked_condition_1 = epochs_condition_1.average()
75 evoked_condition_2 = epochs_condition_2.average()
76
77 plt.figure()
78 plt.subplots_adjust(0.12, 0.08, 0.96, 0.94, 0.2, 0.43)
79
80 plt.subplot(2, 1, 1)
81 # Create new stats image with only significant clusters
82 T_obs_plot = np.nan * np.ones_like(T_obs)
83 for c, p_val in zip(clusters, cluster_p_values):
84     if p_val <= 0.05:
85         T_obs_plot[c] = T_obs[c]
86
87 plt.imshow(T_obs,
88            extent=[times[0], times[-1], freqs[0], freqs[-1]],
89            aspect='auto', origin='lower', cmap='gray')
90 plt.imshow(T_obs_plot,
91            extent=[times[0], times[-1], freqs[0], freqs[-1]],
92            aspect='auto', origin='lower', cmap='RdBu_r')
93
94 plt.xlabel('Time (ms)')
95 plt.ylabel('Frequency (Hz)')
96 plt.title('Induced power (%s)' % ch_name)
97
98 ax2 = plt.subplot(2, 1, 2)
99 evoked_contrast = mne.combine_evoked([evoked_condition_1, evoked_condition_2],
100                                         weights=[1, -1])
101 evoked_contrast.plot(axes=ax2, time_unit='s')
102
103 plt.show()
```



Dataset (from openneuro, if you choose to do the bonus)

The problem for MNI standard brains is that the MNI linear transform has not matched the brains completely to the Talairach brain. This is probably because the Talairach atlas brain is a rather odd shape. Therefore, the MNI brains are slightly larger (in particular higher, deeper and longer) than the Talairach brain.

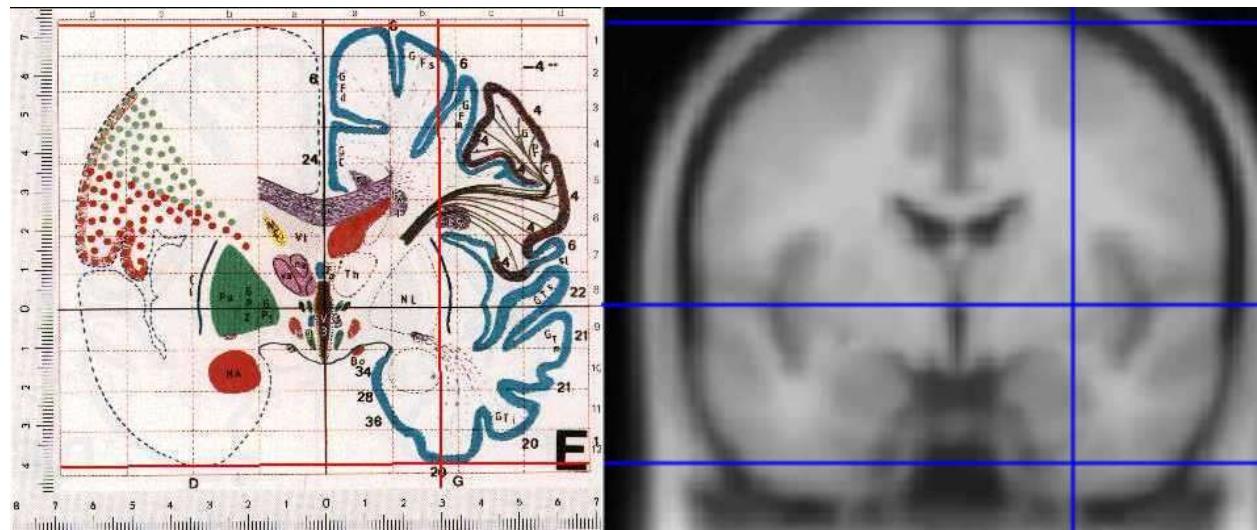


Figure: Differences between MNI and Talairach

<https://imaging.mrc-cbu.cam.ac.uk/imaging/MniTalairach>

There are two approaches in order to register:

1. To get from [McGill](#) [MNI] -SPM96-coordinates to Talairach 88-SPM 95 coordinates:

$$X' = 0.88X - 0.8$$

$$Y' = 0.97Y - 3.32$$

$$Z' = 0.05Y + 0.88Z - 0.44$$

2. non-linear transform of MNI to Talairach

This algorithm gave me the following transformations:

Above the AC ($Z \geq 0$):

$$X' = 0.9900X$$

$$Y' = 0.9688Y + 0.0460Z$$

$$Z' = -0.0485Y + 0.9189Z$$

Below the AC ($Z < 0$):

$$X' = 0.9900X$$

$$Y' = 0.9688Y + 0.0420Z$$

$$Z' = -0.0485Y + 0.8390Z$$

$$T_- = \begin{bmatrix} 0.99 & 0 & 0 & 0 \\ 0 & 0.9688 & 0.042 & 0 \\ 0 & -0.0485 & 0.839 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_+ = \begin{bmatrix} 0.99 & 0 & 0 & 0 \\ 0 & 0.9688 & 0.046 & 0 \\ 0 & -0.0485 & 0.9189 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

I have downloaded the rest parts of XP2 from xp201 to xp222.

openneuro.org/datasets/ds002338/versions/2.0.1

Apps Code cuda cuda code matlab documents Articles RBM nip knee HDL DataSet Concrete opencv > | Reading list

OpenNEURO

PUBLIC DASHBOARD SUPPORT FAQ SIGN IN

xp201 : 1D
xp202 : 1D
xp203 : 1D
xp206 : 1D
xp211 : 1D
xp218 : 1D
xp219 : 1D
xp220 : 1D
xp222 : 1D

Subjects with bi-dimensional feedback display :

xp204 : 2D
xp205 : 2D
xp207 : 2D
xp210 : 2D
xp213 : 2D
xp216 : 2D
xp217 : 2D
xp221 : 2D

EEG DATA

EEG data was recorded using a 64-channel MR compatible solution from Brain Products (Brain Products GmbH, Gilching, Germany).

RAW EEG DATA

sub-xp222_func_nii.gz sub-xp222_anat_nii.gz Show all

```

24 raw = mne.io.read_raw_brainvision('C:/Users/Mahsa/Desktop/Tasks/#####15-P3119-16_Tir/subxp210/sub-xp210_task-2d'
25 raw.load_data()
26 raw.info['bads'] = ['ECG'] # mark bad channels
27 raw.filter(1_fq=1, h_fq=1.0) # low-pass filter
28
29 raw_epochs = mne.preprocessing.find_ecg_events(raw)
30 raw_epochs.plot_image(combine='mean')
31
32 events = mne.find_events(raw, 'STI014') # extract events and epoch data
33 epochs = mne.Epochs(raw, events, event_id=1, tmin=-0.2, tmax=0.5,
34             reject=dict(grad=4000e-13, mag=4e-12))
35 evoked_1 = epochs.average() # compute evoked
36
37 raw = mne.io.read_raw_brainvision('C:/Users/Mahsa/Desktop/Tasks/#####15-P3119-16_Tir/subxp222/sub-xp222_task-2d'
38 raw.load_data()
39 raw.info['bads'] = ['ECG'] # mark bad channels
40 raw.filter(1_fq=1, h_fq=1.0) # low-pass filter
41
42 raw_epochs = mne.preprocessing.find_ecg_events(raw)
43 raw_epochs.plot_image(combine='mean')
44
45 events = mne.find_events(raw, 'STI014') # extract events and epoch data
46 epochs = mne.Epochs(raw, events, event_id=1, tmin=-0.2, tmax=0.5,
47             reject=dict(grad=4000e-13, mag=4e-12))
48 evoked_2 = epochs.average() # compute evoked
49
50 aud_minus_vis = mne.combine_evoked([evoked_1, evoked_2], weights=[1, -1])
51 aud_minus_vis.plot_joint()
52
53
54
55
56
57
58

```

The screenshot shows a Jupyter Notebook cell with Python code for EEG data analysis. The code reads raw EEG data, finds ECG events, and creates epochs. It then averages the evoked responses for two conditions and combines them. A plot of the combined evoked response is shown, featuring topoplots for three time points: 0.127 s, 0.208 s, and 0.273 s. The plot displays spatial distribution of activity across 64 channels. Below the plot, a 'Console I/A' tab shows command-line output related to the analysis parameters.

