Magnetoencephalography and Electroencephalography Brainwave Activity Visualization

1st Tobias
Computing Science Faculty
University of Alberta
Edmonton, Canada
ttobias@ualberta.ca

2nd Haoyu Qiu
Computing Science Faculty
University of Alberta
Edmonton, Canada
hqiu3@ualberta.ca

3rd Sanjana Guntha
Computing Science Faculty
University of Alberta
Edmonton, Canada
sguntha@ualberta.ca

***Abstract*—Understanding human brain function is crucial for various applications, including mental health treatments and cognitive science. This report presents a project aimed at visualizing brainwave activity using Magnetoencephalography (MEG) and Electroencephalography (EEG) data. The project addresses challenges in data visualization and proposes methodologies for data processing, classification, and 3D visualization. Evaluation metrics validation using cross-validation techniques is also discussed. Through innovative visualization techniques and rigorous validation, the project aims to contribute to advancements in neuroscience research.**

I. Introduction

The intricacies of human brain function are pivotal for unraveling mysteries surrounding consciousness and improving mental health treatments. Despite its complexity, advancements in neuroscience, particularly through MEG and EEG analysis, offer promising avenues for deeper insights into brain activity. However, challenges in visualizing brainwave data, including limited techniques and complex interpretation, hinder comprehensive analysis. This project aims to address these challenges by proposing methodologies for data processing, classification, and 3D visualization. Through innovative visualization techniques and rigorous validation, the project seeks to contribute to advancements in neuroscience research and applications in mental health and cognitive science.

II. Approach and Methodologies

A. Approach Overview

The proposed approach for this proposal involves several steps:

- **Data Processing and Classification :** The program reads, classifies, and maps the MEG and EEG dataset, categorizing brainwave signals based on frequency and associated mental activities.
- **Segmentation and 2D Visualization :** The clustered brainwave data are segmented into specific brain areas, and 2D visualization are generated to visualize the spatial distribution of neural activity.
- **3D Visualization :** The final goal is to visualize the clustered and segmented brainwave data on a 3D model, providing context understanding of how certain brainwaves work.
- **Evaluation Metrics Validation :** The project utilizes leave-one-out cross-validation (LOOCV) with Random Forest Classifier to assess the classifier's performance in distinguishing auditory and visual conditions based on raw evoked data, ensuring robust evaluation of classification accuracy.

B. Dataset Overview

In this project, we initially utilized the MNE Sample Dataset as the foundational dataset. These data were collected using the Neuromag Vectorview system at MGH/HMS/MIT Athinoula A. Martinos Center for Biomedical Imaging. The EEG data were acquired concurrently with the MEG using a 60-channel electrode cap. The original MRI dataset was obtained using a Siemens 1.5 T Sonata scanner with an MPRAGE sequence. During the experiment, checkerboard patterns were presented to the subjects in the left and right visual fields, interspersed with tones to the left or right ear. The interval between stimuli was set at 750 ms. Additionally, a smiley face occasionally appeared at the center of the visual field, prompting subjects to press a key with their right index finger as quickly as possible upon its appearance. The sample dataset comprises two main directories: MEG/sample (containing the MEG/EEG data) and subjects/sample (containing the MRI reconstructions).

TABLE I
Contents of the MEG/sample directory

File	*Contents*
`Sample/audvis_raw.fif`	The raw MEG/EEG data
`Audvis.ave`	A template script for off-line averaging
`Auvis.cov`	Noise-covariance matrix template

Moreover, we explored the same methodology with an additional dataset to assess the adaptability of our approach to different types of data. The second dataset utilized EEG

TABLE II
OVERVIEW OF THE CONTENTS OF THE SUBJECTS/SAMPLE DIRECTORY.

File / directory	*Contents*
`bem`	Directory for the forward modelling data
`bem/watershed`	BEM surface segmentation
`bem/inner_skull.surf`	Inner skull surface for BEM
`bem/outer_skull.surf`	Outer skull surface for BEM
`bem/outer_skin.surf`	Skin surface for BEM
`sample-head.fif`	Skin surface in fif format
`surf`	Surface reconstructions
`mri/T1`	The T1-weighted MRI data

data records from individuals with alcohol use disorder. These data originated from a comprehensive study investigating EEG correlates of genetic predisposition to alcoholism. The dataset includes measurements from 64 electrodes positioned on the subjects' scalps, sampled at 256 Hz (with 3.9-msec epochs) for 1 second.

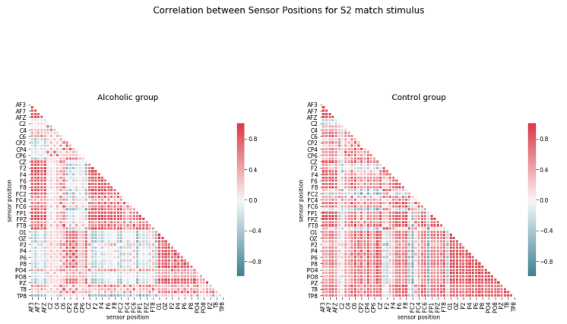

Fig. 1. Alcoholic Dataset

The subjects were divided into two groups: alcoholic and control. Each subject was exposed to either a single stimulus (S1) or two stimuli (S1 and S2), which were pictures of objects selected from the 1980 Snodgrass and Vanderwart picture set. When two stimuli were presented, they were either matched, with S1 identical to S2, or non-matched, where S1 differed from S2.

C. Data Visualization Methodologies

The data visualization methodologies proposed in this project encompass various techniques such as signal traces, scalp topographies, and 3D field maps, providing researchers with comprehensive insights into neural activity patterns. These visualizations offer researchers a nuanced understanding of brain function, facilitating the identification of spatial patterns, cognitive processes, and event-related potentials, thus enhancing the depth of analysis and interpretation in neuroscience research.

1. **Signal traces :** Signal traces are graphical representations of the electrical activity recorded from each channel type, generated using the plotting method in MNE-Python. These traces can be customized by excluding "bad" channels, selecting specific channels for plotting, and color-coding the traces based on channel location using the picks parameter and `spatial_colors()=True`. Additionally, the function allows the overlay of a trace of the root mean square (RMS) across channels, referred to as `gfp=True`. This feature accurately displays the RMS for magnetoencephalography (MEG) data and the global field power for electroencephalography (EEG) data, providing insights into the overall signal magnitude and distribution across channels.

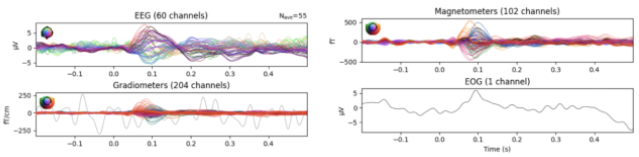

Fig. 2. Signal Traces

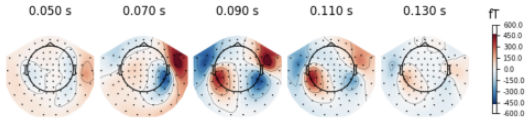

Fig. 3. Scalp Topographies

2. **Scalp topographies :** Plotting scalp topographies entails creating visual representations that showcase the average field distribution over the scalp at particular moments or intervals. These representations offer detailed insights into the spatial patterns of neural activity across the scalp's surface. By utilizing methods such as `plot_topomap()`, users can generate these topographic maps, which are valuable tools in neuroscientific research and data analysis. These maps allow researchers to observe how neural activity is distributed across different scalp regions, aiding in the interpretation of cognitive processes, event-related potentials (ERPs), and other aspects of brain function. Through the visualization of scalp topographies, researchers can identify spatial patterns of brain activity associated with specific stimuli, cognitive tasks, or experimental conditions, enhancing their understanding of neural dynamics and brain functioning.

3. **Arrow Maps :** Arrow maps enhance scalp topographies by incorporating arrows to represent the magnitude and direction of the magnetic field at a specific time point. Implemented with the function `mne.viz.plot_arrowmap()`, they offer valuable insights into the spatial distribution of magnetic fields across the scalp surface, particularly useful in magnetoencephalography (MEG) data analysis.

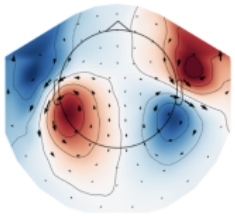

Fig. 4. Arrow Maps

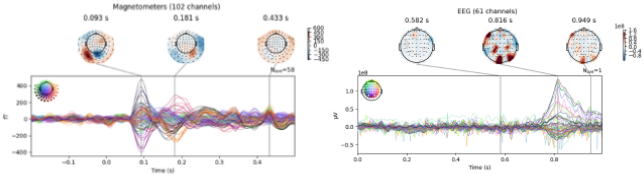

Fig. 5. Join Plots

4. **Join Plots :** Joint plots are comprehensive visualizations that merge butterfly plots with scalp topographies, offering an initial overview of evoked data. By default, these plots automatically position topographies based on peak finding, facilitating easy interpretation. For instance, when plotting conditions such as the right-visual-field condition, separate figures are generated for each channel type if no picks are specified. This integration of butterfly plots and scalp topographies provides researchers with a holistic view of neural activity patterns, aiding in the identification of significant features and trends within the data.

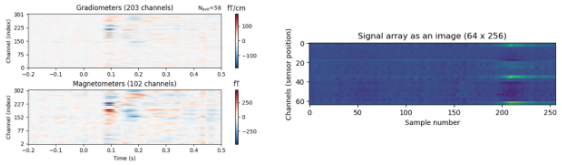

Fig. 6. Image Plots

5. **Image Plots :** The `plot_image()` method for Evoked objects provides a visualization similar to that of Epochs, yet with a distinct presentation: instead of each row representing one epoch, each row represents one channel. This layout allows for a focused examination of individual channels over time. Similar to `epochs.plot_image()`, `evoked.plot_image()` offers a picks parameter for channel selection and various customization options for further analysis and visualization refinement. Researchers can explore specific channels' temporal dynamics and patterns more effectively using this method, enhancing their understanding of neural activity captured by the Evoked object.

Fig. 7. Topographical Subplots

6. **Topographical subplots :** Topographical subplots visually represent sensor-level analyses, displaying the response at each sensor within a topographical layout. `plot_topo()` displays a single condition, while `plot_evoked_topo()` handles multiple conditions if provided with a list of Evoked objects. Legend entries are automatically generated from the Evoked objects' comment attribute. By default, `plot_evoked_topo()` includes all MEG sensors, requiring adjustments to focus on EEG sensors using methods like `mne.pick_typed()`. This method aids in comparing and analyzing neural responses across different conditions efficiently.

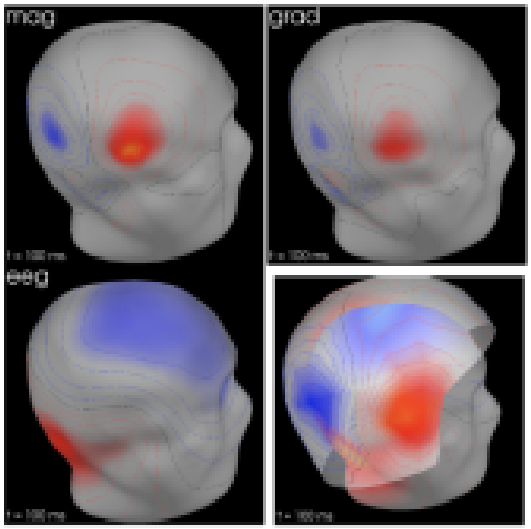

Fig. 8. 3D Field Maps

7. **3D Field Maps :** 3D field maps provide an alternative visualization to the two-dimensional scalp topographies, offering a three-dimensional representation of the field.

To generate these maps, a trans file is necessary to transform locations between the coordinate systems of the MEG device and the head surface, typically based on MRI data. By default, MEG sensors estimate the field on the helmet surface, while EEG sensors estimate the field on the scalp. Once computed, these maps can be plotted using `evoked.plot_field()`. Additionally, MEG sensors can be used to estimate the scalp field by specifying `meg_surf='head'`. Comparing scalp field estimates from different sensor types enables researchers to gain insights into neural activity patterns across different modalities.

III. Evaluation Metrics Validation

For the validation in this project we are utilizing leave-one-out cross-validation (LOOCV) with MNE-Python's `cross_val_multiscore` function to evaluate the performance of a Random Forest Classifier in distinguishing between auditory and visual conditions based on raw evoked data. This approach iteratively trains and evaluates the classifier on subsets of the data, leaving one sample out as a test set in each iteration. By computing performance metrics for each fold of the cross-validation, the methodology ensures robust assessment of the classifier's generalization ability, enabling meaningful insights into the classification task.

TABLE III
LOOCV Evaluation Metrics

Cross-validation scores	***Mean Accuracy***
[0. 1. 0. 0.]	0.25

Accuracy of the brainwave classification.

IV. Project Contribution

My contribution to this project involves conducting research to identify the most suitable method for visualizing brainwave activity. Through careful examination, I determined that MNEPython offers the most appropriate tools for this project. Utilizing the MNE sample dataset, I plotted the data accordingly to create both 2D and 3D visualizations of brainwave activity, effectively distinguishing between right and left audio/visual clusters. Furthermore, I integrated evaluation metrics into the analysis, employing leave-one-out cross-validation (LOOCV) with Random Forest Classifier. This approach allows for the assessment of the classifier's performance in discerning auditory and visual conditions based on raw evoked data.

V. Instructions to Compile the Code

This code has been thoroughly tested using Spyder MNE, a free and open-source scientific environment tailored for Python developers, specifically designed for scientists, engineers, and data analysts. Please visit https://www.spyder-ide.org/ to download the Spyder MNE software. To execute the MNE Sample Dataset Visualization code, simply run the attached script in Spyder with MNE. However, visualizing the Alcoholic Group Dataset requires manual download of the dataset and adjustment of the data path. Please download the dataset from the Kaggle link https://www.kaggle.com/datasets/nnair25/Alcoholics. After downloading, adjust the code to load the dataset from the specified path `'SMNI_CMI_TRAIN/*.csv'` according to the downloaded location. Upon execution, the code will generate visualizations of the brainwave activity data in both 2D and 3D formats, providing valuable insights into the underlying patterns and clusters. The 2D images will be displayed in the right side plot panels, showing various visualization methods elaborated in this paper. The 3D visualization will appear in the MNE PyVista scene, which has several display configurations such as play and pause animation, animation time duration, maximum values, and contour lines. The cross-validation score and mean accuracy will be displayed in the iPython console section, which also shows the compiled codes step by step.

References

[1] McClay, W. A., Yadav, N., Ozbek, Y., Haas, A., Attias, H. T., & Nagarajan, S. S. (2015). A Real-Time Magnetoencephalography Brain-Computer Interface Using Interactive 3D Visualization and the Hadoop Ecosystem. *Brain Sciences*, 5(4), 419-440. https://doi.org/10.3390/brainsci5040419

[2] Hassan, M., Shamas, M., Khalil, M., Falou, W. E., & Wendling, F. (2015). EEGNET: An Open Source Tool for Analyzing and Visualizing M/EEG Connectome. *PLOS ONE*, 10(9), e0138297. https://doi.org/10.1371/journal.pone.0138297

[3] McClay, W. (2018). A Magnetoencephalographic/Encephalographic (MEG/EEG) Brain-Computer Interface Driver for Interactive iOS Mobile Videogame Applications Utilizing the Hadoop Ecosystem, MongoDB, and Cassandra NoSQL Databases. *Diseases*, 6(4), 89. https://doi.org/10.3390/diseases6040089

[4] Fred, A. L., Kumar, S. N., Kumar Haridhas, A., Ghosh, S., Purushothaman Bhuvana, H., Sim, W. K., Vimalan, V., Givo, F. A., Jousmäki, V., Padmanabhan, P., & Gulyás, B. (2022). A Brief Introduction to Magnetoencephalography (MEG) and Its Clinical Applications. *Brain Sciences*, 12(6), 788. https://doi.org/10.3390/brainsci12060788

VI. Source Code

A. MNE Sample Dataset Visualization

```
import os
import numpy as np
import mne
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import LeaveOneOut
from mne.decoding import cross_val_multiscore


sample_data_folder = mne.datasets.sample.data_path()
      # Setting the path to MNE sample data
sample_data_evk_file = os.path.join(sample_data_
    folder, 'MEG', 'sample', 'sample_audvis-ave.fif'
    ) # Constructing the path to the evoked file

# Reading evoked data from a file, applying baseline
     correction, and storing in a list
evokeds_list = mne.read_evokeds(sample_data_evk_file
    , baseline=(None, 0), proj=True, verbose=False)

# Show the condition names and baseline correction
    status for each evoked dataset
for e in evokeds_list:
```

```
    print(f'Condition: {e.comment}, baseline: {e.baseline}')

conds = ('aud/left', 'aud/right', 'vis/left', 'vis/right')
evks = dict(zip(conds, evokeds_list))  # Creating a dictionary of evoked datasets with condition names as keys

# Plotting evoked data for 'aud/left' condition
evks['aud/left'].plot(exclude=[])

# Plotting magnetometer data for 'aud/left' condition with spatial colors and global field power
evks['aud/left'].plot(picks='mag', spatial_colors=True, gfp=True)

times = np.linspace(0.05, 0.13, 5)
# Plotting topographic maps for magnetometer data of 'aud/left' condition at specified times
evks['aud/left'].plot_topomap(ch_type='mag', times=times, colorbar=True)

# Plotting arrow maps for magnetometer data of 'aud/left' condition
mags = evks['aud/left'].copy().pick_types(meg='mag')
mne.viz.plot_arrowmap(mags.data[:, 175], mags.info, extrapolate='local')

# Plotting joint plot for evoked data of 'vis/right' condition
evks['vis/right'].plot_joint()

# Plotting comparison of evoked datasets with different combination methods
def custom_func(x):
    return x.max(axis=1)

for combine in ('mean', 'median', 'gfp', custom_func):
    mne.viz.plot_compare_evokeds(evks, picks='eeg', combine=combine)

# Plotting comparison of evoked datasets for a specific MEG channel
mne.viz.plot_compare_evokeds(evks, picks='MEG 1811', colors=dict(aud=0, vis=1), linestyles=dict(left='solid', right='dashed'))

temp_list = list()
# Creating a temporary list of evoked datasets with modified comments
for idx, _comment in enumerate(('foo', 'foo', '', None, 'bar'), start=1):
    _evk = evokeds_list[0].copy()
    _evk.comment = _comment
    _evk.data *= idx  # Multiplying data to differentiate traces
    temp_list.append(_evk)

# Plotting comparison of temporary evoked datasets
mne.viz.plot_compare_evokeds(temp_list, picks='mag')

# Plotting image of evoked data for 'vis/right' condition
evks['vis/right'].plot_image(picks='meg')

# Plotting comparison of evoked datasets with customization options for EEG channels
mne.viz.plot_compare_evokeds(evks, picks='eeg', colors=dict(aud=0, vis=1), linestyles=dict(left='solid', right='dashed'), axes='topo', styles=dict(aud=dict(linewidth=1), vis=dict(linewidth=1)))

# Plotting topographic maps of evoked data for all conditions
mne.viz.plot_evoked_topo(evokeds_list)

subjects_dir = os.path.join(sample_data_folder, 'subjects')
sample_data_trans_file = os.path.join(sample_data_folder, 'MEG', 'sample', 'sample_audvis_raw-trans.fif')

# Making and plotting field maps for 'aud/left' condition
maps = mne.make_field_map(evks['aud/left'], trans=sample_data_trans_file, subject='sample', subjects_dir=subjects_dir)
evks['aud/left'].plot_field(maps, time=0.1)

# Making and plotting field maps for each channel type
for ch_type in ('mag', 'grad', 'eeg'):
    evk = evks['aud/right'].copy().pick(ch_type)
    _map = mne.make_field_map(evk, trans=sample_data_trans_file, subject='sample', subjects_dir=subjects_dir, meg_surf='head')
    fig = evk.plot_field(_map, time=0.1)
    mne.viz.set_3d_title(fig, ch_type, size=20)

# Evaluation

# Setting the path to MNE sample data
sample_data_folder = mne.datasets.sample.data_path()

# Constructing the path to the evoked file
sample_data_evk_file = os.path.join(sample_data_folder, 'MEG', 'sample', 'sample_audvis-ave.fif')

# Reading evoked data from a file, applying baseline correction, and storing in a list
evokeds_list = mne.read_evokeds(sample_data_evk_file, baseline=(None, 0), proj=True, verbose=False)

# Creating a dictionary of evoked datasets with condition names as keys
conds = ('aud/left', 'aud/right', 'vis/left', 'vis/right')
evks = dict(zip(conds, evokeds_list))

# Define features (X) and labels (y) for classification
X = []
y = []
for cond_name, evk in evks.items():
    X.append(evk.data)  # Using raw evoked data as features
    if 'aud' in cond_name:
        y.append(0)  # Assigning label 0 for auditory conditions
    else:
        y.append(1)  # Assigning label 1 for visual conditions
X = np.array(X)
y = np.array(y)

# Flatten the data for RandomForestClassifier
X_flattened = np.concatenate([x.reshape(1, -1) for x in X])

# Define the machine learning model (Random Forest Classifier)
estimator = RandomForestClassifier()
```

```
# Perform leave-one-out cross-validation (LOOCV) and evaluate the model's performance
loo = LeaveOneOut()
scores = cross_val_multiscore(estimator, X_flattened, y, cv=loo)

# Print the cross-validation scores and mean accuracy
print("Cross-validation scores:", scores)
print("Mean accuracy:", np.mean(scores))
```

MNE_Dataset.py

B. Alcoholic Group Dataset Visualization

```
import mne

import numpy as np
import pandas as pd
import os
from tqdm import tqdm
import glob
from matplotlib import pyplot as plt


#to get the dataframe records from the dataset
def get_dataframe_records(df, name, trial_number, matching_condition, channel_list):
    df_record = df[df['name'].eq(name) & df['trial number'].eq(trial_number) & df['matching condition'].eq(matching_condition)].set_index(['sensor position']).loc[channel_list]
    return df_record

#the function to get the signal array for visualization
def get_signal_array(df, name, trial_number, matching_condition, channel_list):
    df_record = df[df['name'].eq(name) & df['trial number'].eq(trial_number) & df['matching condition'].eq(matching_condition)].set_index(['sensor position']).loc[channel_list]
    return df_record.to_numpy()[:, 4:]


#The function to plot the topomap for the eeg data
def plot_topomap(signal_array, save_path_animation=None, show_names=False, start_time=0.05, end_time=1, step_size=0.1):

    montage = mne.channels.make_standard_montage('standard_1020')

    ch_to_remove = []
    for ch in channel_list_fixed:
        if ch not in list(set(montage.ch_names).intersection(channel_list_fixed)):
            ch_to_remove.append(channel_list_fixed.index(ch))
    arr = np.delete(signal_array.copy(), ch_to_remove, axis=0)

    info = mne.create_info(ch_names=list(set(montage.ch_names).intersection(channel_list_fixed)), sfreq=256, ch_types='eeg')

    evkd = mne.EvokedArray(arr, info)

    evkd.set_montage(montage)

    evkd.plot_topomap(np.arange(start_time, end_time, step_size), ch_type='eeg', time_unit='s', ncols=5, nrows=2, show_names=show_names)


#The function to plot the jointed topomap for the eeg data
def plot_joint_topomap(signal_array, save_path_animation=None, show_names=False, start_time=0.05, end_time=1, step_size=0.1):

    montage = mne.channels.make_standard_montage('standard_1020')

    ch_to_remove = []
    for ch in channel_list_fixed:
        if ch not in list(set(montage.ch_names).intersection(channel_list_fixed)):
            ch_to_remove.append(channel_list_fixed.index(ch))
    arr = np.delete(signal_array.copy(), ch_to_remove, axis=0)

    info = mne.create_info(ch_names=list(set(montage.ch_names).intersection(channel_list_fixed)), sfreq=256, ch_types='eeg')
    evkd = mne.EvokedArray(arr, info)

    evkd.set_montage(montage)

    evkd.plot_joint()

sample_data_folder = mne.datasets.sample.data_path()
sample_data_evk_file = os.path.join(sample_data_folder, 'MEG', 'sample',
                                    'sample_audvis-ave.fif')


#Load the csv file into dataframe
_dfs_list = []
p = glob.glob('SMNI_CMI_TRAIN/*.csv')
for files in tqdm(glob.glob('SMNI_CMI_TRAIN/*.csv')):
    _dfs_list.append(pd.read_csv(files))
print(_dfs_list)
df = pd.concat(_dfs_list)
del(_dfs_list)
df = df.drop(['Unnamed: 0'], axis=1)
df.head(3)

channel_list = list(set(df['sensor position']))
channel_list.sort()


#The dictionary to correct the channel name
channel_mapping_dict = {
    'AFZ':'AFz',
    'CPZ':'CPz',
    'CZ':'Cz',
    'FCZ':'FCz',
    'FP1':'Fp1',
    'FP2':'Fp2',
    'FPZ':'Fpz',
    'FZ':'Fz',
    'OZ':'Oz',
    'POZ':'POz',
    'PZ':'Pz',
}

channel_mapping_full = dict()


#map the channel names
for ch in channel_list:
    if ch in channel_mapping_dict:
```

```
        channel_mapping_full[ch] = channel_mapping_
            dict[ch]
    else:
        channel_mapping_full[ch] = ch

channel_list_fixed = [channel_mapping_full[ch] for
    ch in channel_list]

df['sensor position'] = df['sensor position'].map(
    channel_mapping_full)
df.head(3)

transposed_dataframe_list = []

#organize and reconstruct the dataframe containing
    EEG data
for group_dataframe in tqdm(df.groupby(['name', '
    trial number', 'matching condition', 'sensor
    position', 'subject identifier'])):
    tmp = pd.DataFrame(group_dataframe[1]['sensor
        value']).T
    tmp.columns = [f'sample_{idx}' for idx in range
        (256)]
    tmp['name'] = group_dataframe[0][0]
    tmp['trial number'] = group_dataframe[0][1]
    tmp['matching condition'] = group_dataframe
        [0][2]
    tmp['sensor position'] = group_dataframe[0][3]
    tmp['subject identifier'] = group_dataframe
        [0][4]

    transposed_dataframe_list.append(tmp)

df = pd.concat(transposed_dataframe_list)
df = df[[*df.columns[-5:],*df.columns[0:-5]]]
df = df.reset_index(drop=True)
df.head(3)


#visualize the dataset
df_record = get_dataframe_records(df, 'co2a0000364',
     0, 'S1 obj', channel_list_fixed)


signal_array = get_signal_array(df, 'co2a0000364',
    10, 'S1 obj', channel_list_fixed)

plt.title('Signal Array as an image (64 x 256)')
plt.ylabel('Sensor Position)')
plt.xlabel('Sample Numbers')
plt.imshow(signal_array.astype(int))
plt.show()


#generate plot of signal over sample numbers
channels_to_display = ['AF1', 'CP3', 'F1']
for channel in channels_to_display:
    plt.xlabel('Sample number')
    plt.plot(signal_array[channel_list.index(channel
        )])
plt.legend(channels_to_display)

info_data = mne.create_info(ch_names=channel_list_
    fixed, sfreq=256, ch_types=['eeg']*64)
raw = mne.io.RawArray(signal_array, info_data)

standard_1020_montage = mne.channels.make_standard_
    montage('standard_1020')
raw.drop_channels(['X', 'Y', 'nd'])
raw.set_montage(standard_1020_montage)

raw.plot_psd()
raw.plot_psd(average=True)

raw_filtered = raw.copy().filter(8,30, verbose=False
    )
raw_filtered.plot_psd()
raw_filtered.plot_psd(average=True)

plt.imshow(raw.get_data())
plt.show()
plt.imshow(raw.copy().filter(1,10, verbose=False).
    get_data())
plt.show()
plt.plot(raw.copy().get_data()[40])
plt.plot(raw.copy().filter(8,30, verbose=False).get_
    data()[40])

ica = mne.preprocessing.ICA(random_state=42, n_
    components=20)
ica.fit(raw.copy().filter(1,None, verbose=False),
    verbose=False)
ica.plot_components()

plot_topomap(signal_array, show_names=False)


plot_joint_topomap(signal_array)
```

Alcoholic_Group.py