1. Import & load libraries

In [ ]:
```
%pip cache purge

%pip install mne
%pip install matplotlib
%pip install numpy
%pip install pandas
%pip install scikit-learn
```

In [ ]:
```
pip list
```

In [ ]:
```
import mne
from mne.datasets import eegbci
from mne.io import read_raw_edf
from mne.io import concatenate_raws
from mne.preprocessing import ICA

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from io import BytesIO
from PIL import Image

import os
import re
import warnings
import glob

from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

from typing import List
```

In [ ]:
```
warnings.filterwarnings("ignore", message="FigureCanvasAgg is non-interactive, and
warnings.filterwarnings("ignore", category=RuntimeWarning, message="Channel locati
```

2. Info

Experimental Protocol

This data set consists of over 1500 one- and two-minute EEG recordings, obtained from ***109 volunteers***, as described below.

Subjects performed different motor/imagery tasks while 64-channel EEG were recorded using the BCI2000 system (http://www.bci2000.org). Each subject performed ***14 experimental runs***: two one-minute baseline runs (one with eyes open, one with eyes closed), and three two-minute runs of each of the four following tasks:

- **TASK 1**: A target appears on either the left or the right side of the screen. The subject opens and closes the corresponding fist until the target disappears. Then the subject relaxes.
- **TASK 2**: A target appears on either the left or the right side of the screen. The subject imagines opening and closing the corresponding fist until the target disappears. Then the subject relaxes.
- **TASK 3**: A target appears on either the top or the bottom of the screen. The subject opens and closes either both fists (if the target is on top) or both feet (if the target is on the bottom) until the target disappears. Then the subject relaxes.
- **TASK 4**: A target appears on either the top or the bottom of the screen. The subject imagines opening and closing either both fists (if the target is on top) or both feet (if the target is on the bottom) until the target disappears. Then the subject relaxes.

Description of data:

The experimental runs were:

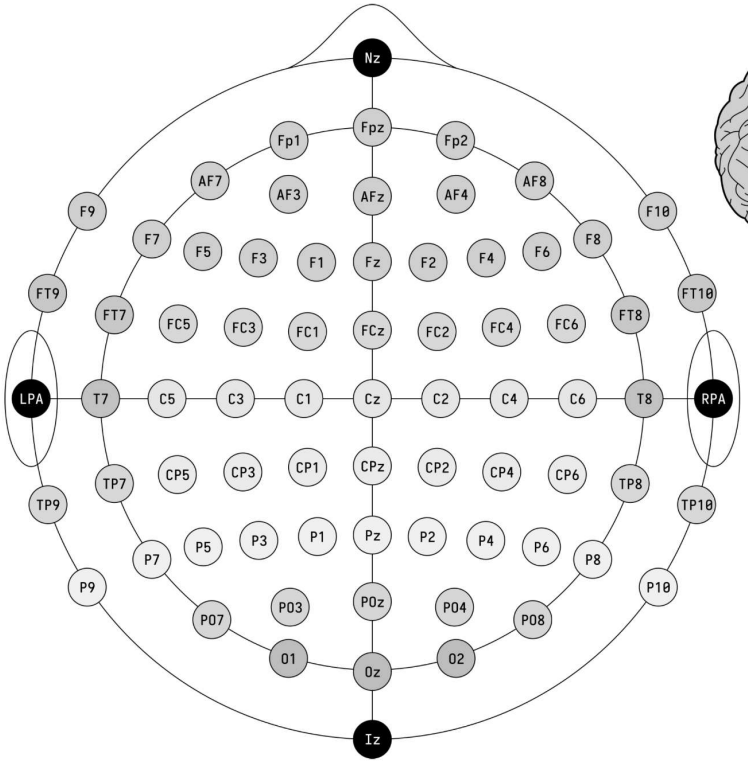
- Baseline, eyes open
- Baseline, eyes closed
- Task 1 (open and close left or right fist)
- Task 2 (imagine opening and closing left or right fist)
- Task 3 (open and close both fists or both feet)
- Task 4 (imagine opening and closing both fists or both feet)

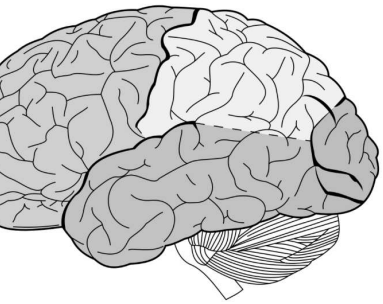
Each annotation includes one of three codes (T0, T1, or T2):

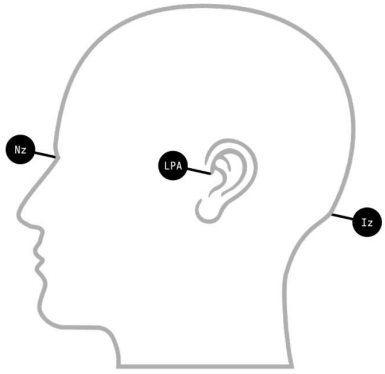
- **T0** corresponds to rest
- **T1** corresponds to onset of motion (real or imagined) of the left fist (in runs 3, 4, 7, 8, 11, and 12) both fists (in runs 5, 6, 9, 10, 13, and 14)
- **T2** corresponds to onset of motion (real or imagined) of the right fist (in runs 3, 4, 7, 8, 11, and 12) both feet (in runs 5, 6, 9, 10, 13, and 14)

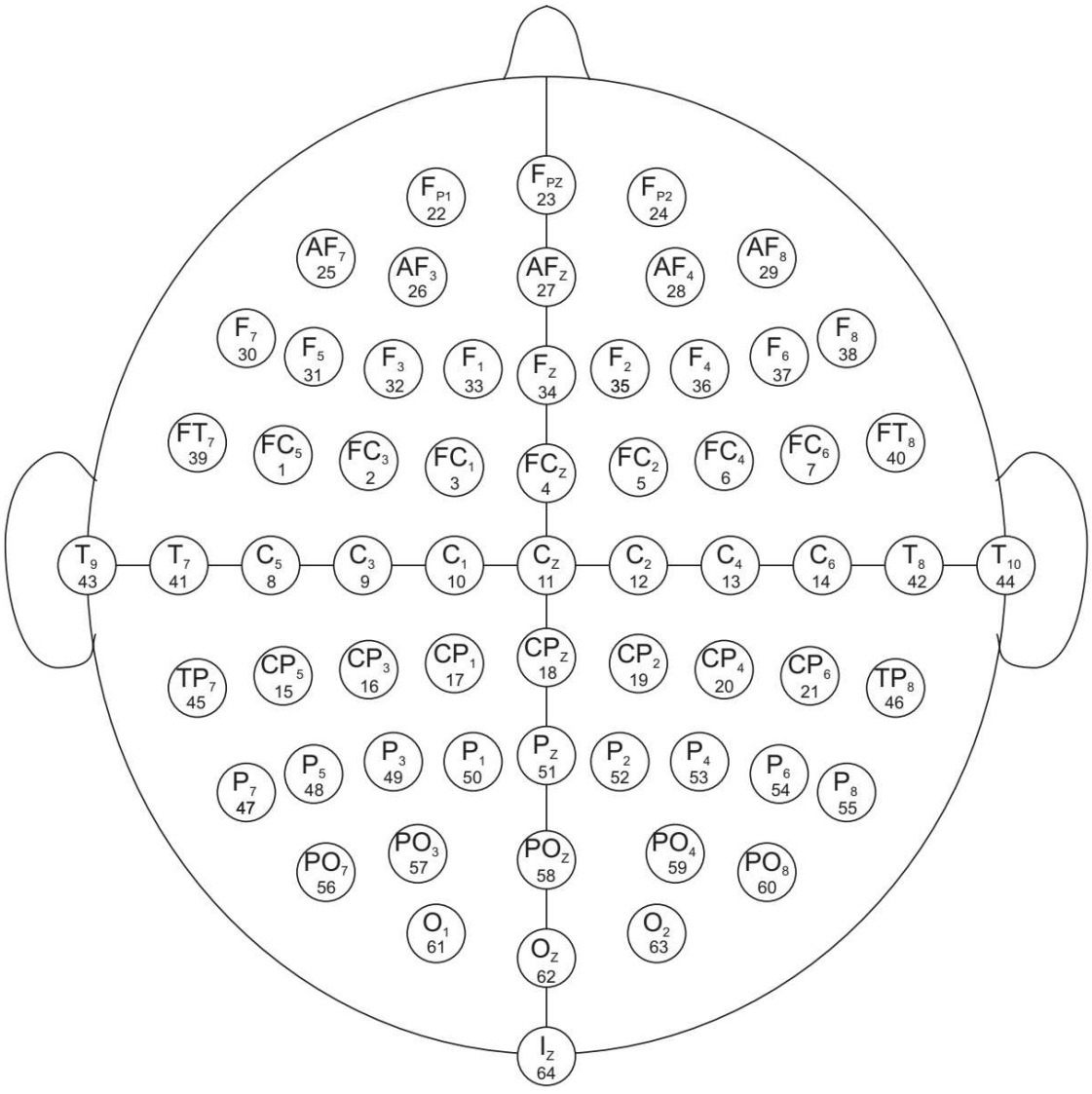
Run	Task
1	Baseline, eyes open
2	Baseline, eyes closed
3, 7, 11	Motor execution: left vs right hand
4, 8, 12	Motor imagery: left vs right hand
5, 9, 13	Motor execution: hands vs feet
6, 10, 14	Motor imagery: hands vs feet

The EEGs were recorded from 64 electrodes as per the international system (excluding electrodes Nz, F9, F10, FT9, FT10, A1, A2, TP9, TP10, P9, and P10)


Nz
Fp1
Fpz
Fp2
AF7
AF3
AFz
AF4
AF8
F9
F7
F5
F3
F1
Fz
F2
F4
F6
F8
F10
FT9
FT7
FC5
FC3
FC1
FCz
FC2
FC4
FC6
FT8
FT10
LPA
T7
C5
C3
C1
Cz
C2
C4
C6
T8
RPA
TP9
TP7
CP5
CP3
CP1
CPz
CP2
CP4
CP6
TP8
TP10
P9
P7
P5
P3
P1
Pz
P2
P4
P6
P8
P10
PO7
PO3
POz
PO4
PO8
O1
Oz
O2
Iz




Nz
LPA
Iz


FP1 22
FPZ 23
FP2 24
AF7 25
AF3 26
AFZ 27
AF4 28
AF8 29
F7 30
F5 31
F3 32
F1 33
FZ 34
F2 35
F4 36
F6 37
F8 38
FT7 39
FC5 1
FC3 2
FC1 3
FCZ 4
FC2 5
FC4 6
FC6 7
FT8 40
T9 43
T7 41
C5 8
C3 9
C1 10
CZ 11
C2 12
C4 13
C6 14
T8 42
T10 44
TP7 45
CP5 15
CP3 16
CP1 17
CPZ 18
CP2 19
CP4 20
CP6 21
TP8 46
P7 47
P5 48
P3 49
P1 50
PZ 51
P2 52
P4 53
P6 54
P8 55
PO7 56
PO3 57
POZ 58
PO4 59
PO8 60
O1 61
OZ 62
O2 63
IZ 64

3. Load data

In [ ]:
```
def load_data(subjects, runs, data_dir="../data/files/"):
    """
    Load and preprocess EEG data for given subjects and runs.
    """
    all_raws = []

    for subject in subjects:
        print(f"\n=== Loading data from_volunteer {subject} ===")
        try:
            # Download data to the specified directory
            raw_fnames = mne.datasets.eegbci.load_data(subject, runs, path=data_di
            raws = [mne.io.read_raw_edf(f, preload=True, verbose=True) for f in ra
            raw = mne.concatenate_raws(raws)
        except Exception as e:
            warnings.warn(f"Skipping subject {subject} due to an error: {e}")
            continue

        # Set standard montage
        try:
            raw.set_montage("standard_1005", on_missing="ignore")
        except Exception as e:
            warnings.warn(f"Could not set montage for subject {subject}: {e}")

        # Extract events from annotations
        try:
            events, _ = mne.events_from_annotations(raw)
            new_annot = mne.annotations_from_events(
                events=events,
                event_desc=new_labels_events,
                sfreq=raw.info['sfreq'],
                orig_time=raw.info['meas_date']
                )
            raw.set_annotations(new_annot)
        except Exception as e:
            warnings.warn(f"Could not update event labels for subject {subject}: {

        all_raws.append(raw)

    if not all_raws:
        raise ValueError("No valid EEG data loaded. Check subject and run IDs.")

    # Concatenate all subjects' data
    return mne.concatenate_raws(all_raws)
```

In [ ]:
```
subjects = [1] # subjects to load (from 1 to 109 volunteers)

runs = [3]  # experimental runs for each subject
# runs = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]

raw_data = load_data(subjects, runs)
```

In [ ]:
```
print(raw_data)
```

In [ ]:
```
def summarize_edf_files(data_dir="../data/files/"):
    """
    Recursively summarize the contents of all .edf files in the given directory.
    """
    edf_files = glob.glob(os.path.join(data_dir, "**/*.edf"), recursive=True)
```

```
    if not edf_files:
        print("No EDF files found in the directory or subdirectories.")
        return

    print(f"Found {len(edf_files)} EDF files in {data_dir}**:\n")

    for edf_file in edf_files:
        try:
            raw = mne.io.read_raw_edf(edf_file, preload=False, verbose=False)
            info = raw.info

            rel_path = os.path.relpath(edf_file, data_dir)  # Get relative path fo

            print(f"File: {rel_path}")
            print(f"  - Channels: {len(info['ch_names'])}")
            print(f"  - Length: {len(raw)}")
            print(f"  - Sampling Frequency: {info['sfreq']} Hz")
            print("-" * 40)

        except Exception as e:
            print(f"Error reading {edf_file}: {e}")

# Run the function
summarize_edf_files()
```

In [ ]:
```
print(raw_data)
print(raw_data.info)
print(raw_data.annotations)
print(raw_data.annotations.description)
print(raw_data.annotations.onset)
print(raw_data.info['ch_names'])
```

In [ ]:
```
eegbci.standardize(raw_data)  # Standardize channel names
print(raw_data.info)
print(raw_data.info['ch_names'])
```

In [ ]:
```
montage = mne.channels.make_standard_montage('standard_1005')
raw_data.set_montage(montage)
```

4. Show events

In [ ]:
```
# Show the events
events, event_id = mne.events_from_annotations(raw_data)
print(event_id)
print(events)
```

In [ ]:
```
event_id = {
    'rest' if k == np.str_('T0') else
    'hands' if k == np.str_('T1') else
    'feets': v
    for k, v in event_id.items()
}
print(event_id)
print(events)
```

In [ ]:
```
plt.rcParams["figure.figsize"] = (14, 5)
mne.viz.plot_events(events, sfreq=raw_data.info['sfreq'],
                    first_samp=raw_data.first_samp, event_id=event_id)
plt.show()
```

5. Select EEG channels

In [ ]:
```
# Select only EEG channels for further analysis
picks = mne.pick_types(raw_data.info, meg=True, eeg=True, stim=False, eog=False, e
print(picks.shape)
print(picks)
```

In [ ]:
```
raw_data.plot(picks=picks);
```

6. Show PSD Power Spectral Density

The Power Spectral Density (PSD) is a way to analyze the frequency content of a signal. It tells us how the power of a signal is distributed across different frequencies.

The PSD describes how the power (or variance) of a signal is distributed over different frequency components.

It is computed using the Fourier Transform, which decomposes a time-domain signal into its frequency components.

In [ ]:
```
psd = raw_data.compute_psd(picks=picks)
psd.plot();
```

7. Apply ICA Independent Component Analysis

ICA (Independent Component Analysis) is a technique used to separate mixed signals into independent components. In EEG/MEG analysis, it is mainly used to remove artifacts such as eye blinks, eye movements, and muscle noise.

ICA decomposes a signal into a set of statistically independent components, allowing us to identify and remove noise sources without affecting neural signals.

ICA is useful for:

- Removing eye blink artifacts (blinks)
- Removing eye movement and muscle artifacts
- Separating neural activity from external noise

In [ ]:
```
# Configure ICA with 25 components
raw_data_ica = raw_data.copy()
raw_data_ica_filtered = raw_data_ica.filter(1, 70, picks=picks)
ica = ICA(n_components=25, random_state=42, method='fastica', max_iter=800)

# Fit ICA to the data
ica.fit(raw_data_ica_filtered, picks=picks)

# Plot ICA components
ica.plot_sources(raw_data_ica_filtered, picks=range(0, 25))
plt.show()
```

In [ ]:
```
ica.apply(raw_data_ica_filtered)
```

In [ ]:
```
raw_data_ica_filtered.plot(picks=picks);
```

In [ ]:
```
# Function to convert MNE plot to an image
def raw_plot_to_image(raw_data, picks):
    buf = BytesIO()
    fig = raw_data.plot(picks=picks, show=False)
    fig.savefig(buf, format='png', bbox_inches='tight', dpi=150)  # Reduce empty s
    plt.close(fig)
    buf.seek(0)
    return np.array(Image.open(buf))

# Create figure with tight layout
fig, axes = plt.subplots(1, 2, figsize=(14, 6))
fig.suptitle('Raw Data BEFORE ICA vs. AFTER ICA', fontsize=16, fontweight='bold')

# Plot BEFORE ICA
img_before = raw_plot_to_image(raw_data, picks)
axes[0].imshow(img_before)
axes[0].set_title('raw_data BEFORE ICA', fontsize=12)
axes[0].axis('off')  # Hide axes completely
axes[0].spines[:].set_visible(False)  # Hide borders

# Plot AFTER ICA
img_after = raw_plot_to_image(raw_data_ica_filtered, picks)
axes[1].imshow(img_after)
axes[1].set_title('raw_data AFTER ICA', fontsize=12)
axes[1].axis('off')  # Hide axes completely
axes[1].spines[:].set_visible(False)

# Use tight_layout to optimize spacing
plt.tight_layout(pad=0.1, w_pad=0.1, h_pad=0.1)  # Reduce the space between subplo

plt.show()
```

In [ ]:
```
psd_ica_raw = raw_data_ica_filtered.compute_psd(picks=picks, fmax=80)
psd_ica_raw.plot();
```

8. Notch filter

A Notch Filter (also called a Band-stop filter) is used to remove specific unwanted frequencies from a signal, particularly when a narrow frequency band causes interference or noise. In signal processing, the "notch" refers to the removal of frequencies within a small range, leaving the other frequencies untouched.

How It Works:

- Passband and Stopband:
  - The passband is the range of frequencies that the filter allows to pass through without attenuation.
  - The stopband is the range of frequencies that the filter suppresses.
  - A notch filter specifically targets and reduces a narrow band of frequencies (the notch), while passing frequencies outside of that band.
- Frequency Range:
  - The notch filter is designed to attenuate a specific frequency (or a small range of frequencies) while allowing the other frequencies to pass through. This is particularly useful when dealing with known interference frequencies, such as the power line frequency (50 Hz or 60 Hz) that can appear in many electrical signals.

Applications of Notch Filters:

- Power line interference: In electrical signals, the 50 Hz or 60 Hz power line frequency is a common source of noise. The notch filter is used to remove this frequency without affecting the rest of the signal.
- Electroencephalography (EEG): In EEG data, notch filters are commonly applied to remove noise caused by electrical equipment, such as power line interference.
- Audio processing: It can also be used to remove hum or buzz noises caused by equipment, such as electrical hum from a microphone or speakers.
- Communication systems: To filter out specific unwanted frequencies or interference.

In [ ]:
```
# # Notch filter
# notch_freq = 60
# raw_data.notch_filter(notch_freq, fir_design='firwin')
# raw_data.compute_psd().plot()
```

In [ ]:
```
# # Band-pass filter keep only alpha and beta waves
# low_cutoff = 8
# high_cutoff = 30
# raw_data.filter(low_cutoff, high_cutoff, fir_design='firwin')
# raw_data.compute_psd().plot()
```

In [ ]:
```
# raw_data.compute_psd().plot()
# raw_data.compute_psd().plot(average=True)
```

In [ ]:
```
raw_filtered.plot_psd(average=False)
plt.show()
```

In [ ]:
```
data = raw_filtered.copy().get_data()

# Standardization
scaler = StandardScaler()
data_standardized = scaler.fit_transform(data.T).T
```

In [ ]:
```
# Covariance Matrix Computation
covariance_matrix = np.cov(data_standardized)

# Eigenvalue and Eigenvector Decomposition
eigenvalues, eigenvectors = np.linalg.eig(covariance_matrix)

# avoidng complex domain parts of eigenvalues
eigenvalues = np.real(eigenvalues)
eigenvectors = np.real(eigenvectors)

# Feature Vector Formation
# Sort eigenvalues and their corresponding eigenvectors
idx = np.argsort(eigenvalues)[::-1]
eigenvalues = eigenvalues[idx]
eigenvectors = eigenvectors[:, idx]
```

In [ ]:
```
# Covariance Matrix Computation
covariance_matrix = np.cov(data_standardized)

# Eigenvalue and Eigenvector Decomposition
eigenvalues, eigenvectors = np.linalg.eig(covariance_matrix)

# avoidng complex domain parts of eigenvalues
eigenvalues = np.real(eigenvalues)
eigenvectors = np.real(eigenvectors)

# Feature Vector Formation
# Sort eigenvalues and their corresponding eigenvectors
idx = np.argsort(eigenvalues)[::-1]
eigenvalues = eigenvalues[idx]
eigenvectors = eigenvectors[:, idx]
```

In [ ]:
```
# Calculate the cumulative explained variance
explained_variance_ratio = np.cumsum(eigenvalues) / np.sum(eigenvalues) * 100

# Create a DataFrame with the number of components and cumulative variance
df_explained_variance = pd.DataFrame({
    'Number of Components': range(1, len(eigenvalues) + 1),
    'Cumulative Explained Variance (%)': explained_variance_ratio
})

# Display the DataFrame 10 first rows
print(f"{df_explained_variance.head(10)}\n\n")

# Select the top k eigenvectors
k = 10  # Number of principal components to keep
eigenvectors_reduced = eigenvectors[:, :k]
```

```
# Recast the Data
data_pca = np.dot(eigenvectors_reduced.T, data_standardized)

# Visualize the cumulative explained variance
plt.figure(figsize=(12, 6))
plt.plot(range(1, k + 1), explained_variance_ratio[:k], marker='o', linestyle='--'
plt.title('Cumulative Explained Variance by PCA Components')
plt.xlabel('Number of PCA Components')
plt.ylabel('Cumulative Explained Variance (%)')
plt.xticks(range(1, k + 1))
plt.grid()
plt.show()
```