Python Basics & EEG Data Loading

Γιώργος Μανίκης

- Before writing any EEG analysis code you need to install the required Python libraries.

- ***pip install mne numpy scipy matplotlib pandas***

- Verify the installation by opening a Python console and running:

```
import mne
import numpy as np
import matplotlib.pyplot as plt
print('MNE version    :', + mne.__version__)
print('NumPy version  :‘ + np.__version__)
```

- **Minimum Python syntax needed for EEG analysis:**

```
# String (text)
subject_id = 'S01'

# Integer
n_channels = 64

# Float (decimal)
sampling_freq = 256.0       # Hz

# Boolean
is_preprocessed = False

# Print multiple values
print(subject_id, n_channels, sampling_freq, is_preprocessed)
# Output: S01 64 256.0 False
```

- **A list is an ordered, mutable collection of items. In EEG analysis, lists are used to store channel names, file paths, event labels, and more:**

```
channels = ['Fp1', 'Fp2', 'F3', 'F4', 'C3', 'C4', 'P3', 'P4', 'O1', 'O2']

# Indexing — Python uses 0-based indexing
print(channels[0])        # 'Fp1'   (first element)
print(channels[-1])       # 'O2'    (last element)

# Slicing: list[start:stop]  (stop is exclusive)
print(channels[0:3])      # ['Fp1', 'Fp2', 'F3']
print(channels[:3])       # same as above
print(channels[6:])       # ['P3', 'P4', 'O1', 'O2']

# Length
print(len(channels))      # 10

# Modifying lists
channels.append('Oz')     # add to end
channels.remove('Fp2')    # remove by value
```

- **Dictionaries store key–value pairs. MNE uses dictionaries extensively for metadata (info), event IDs, and configuration:**

```
subject_info = {
    'id'    : 'S01',
    'age'   : 22,
    'sex'   : 'M',
    'task'  : 'motor_imagery',
}

# Access by key
print(subject_info['task'])       # 'motor_imagery'

# Add or update a key
subject_info['handedness'] = 'right'

# Iterate over keys and values
for key, value in subject_info.items():
    print(f'{key:12s}: {value}')
```

- **Python has two main loop types: for (iterate over a sequence) and while (repeat while a condition is true). EEG scripts use for loops constantly — to process channels, epochs, or subjects:**

```
# for loop over a list
eeg_bands = ['Delta', 'Theta', 'Alpha', 'Beta', 'Gamma']
for band in eeg_bands:
    print('Band:', band)

# range() generates a sequence of integers
for i in range(5):          # 0, 1, 2, 3, 4
    print(i, end=' ')
print()

# enumerate() gives both index and value
for idx, band in enumerate(eeg_bands):
    print(f'{idx}: {band}')

# List comprehension — compact loop that builds a list
upper_bands = [b.upper() for b in eeg_bands]
print(upper_bands)    # ['DELTA', 'THETA', ...]
```

- **Functions let you package reusable logic. In EEG analysis you will write functions to preprocess data, extract features, and evaluate classifiers:**

```
def band_name(low_hz, high_hz):
    """Return the EEG frequency band name for a given range."""
    if   high_hz <= 4 :  return 'Delta'
    elif high_hz <= 8 :  return 'Theta'
    elif high_hz <= 13:  return 'Alpha'
    elif high_hz <= 30:  return 'Beta'
    else              :  return 'Gamma'

# Call the function
print(band_name(8, 13))    # Alpha
print(band_name(1,  4))    # Delta

# Functions can return multiple values as a tuple
def signal_stats(data):
    return data.mean(), data.std(), data.min(), data.max()

import numpy as np
x = np.random.randn(1000)
mean, std, mn, mx = signal_stats(x)
print(f'Mean={mean:.3f}, Std={std:.3f}')
```

- **NumPy: The Heart of EEG Signal Processing**
- NumPy (Numerical Python) provides N-dimensional arrays and a rich library of mathematical functions. It is the foundation of every scientific Python library including MNE, SciPy, and scikit-learn

Creating Arrays

```
import numpy as np


# From a list
arr = np.array([1.0, 2.5, 3.0, 4.7])
print(arr.shape)          # (4,)
print(arr.dtype)          # float64


# Time axis for a 1-second, 256 Hz EEG segment
t = np.linspace(0, 1, 256)          # 256 evenly spaced points from 0 to 1 s
print(t.shape)            # (256,)
print(t[:5])              # [0.     0.004  0.008  0.012  0.016]


# Zeros, ones, random
noise  = np.zeros(256)              # 256 zeros
dc     = np.ones(256) * 5e-6        # DC offset of 5 µV
gauss  = np.random.randn(256) * 10e-6 # Gaussian noise, std=10 µV


# 2-D array: simulated EEG (64 channels × 1024 samples)
eeg_2d = np.random.randn(64, 1024)
print(eeg_2d.shape)       # (64, 1024)
```

- **NumPy: The Heart of EEG Signal Processing**
- MNE stores EEG as 2D (channels × samples) or 3D (epochs × channels × samples) arrays. Understanding indexing is essential

Array Indexing and Slicing

```
eeg = np.random.randn(64, 2560)   # 64 channels, 10 s at 256 Hz

# --- 1-D indexing ---
first_sample  = eeg[0, 0]         # channel 0, sample 0
channel_C3    = eeg[6, :]          # all samples from channel index 6
first_second  = eeg[:, :256]       # all channels, first 256 samples (1 s)

# --- Boolean masking ---
# Find samples where C3 exceeds ±75 µV (artefact detection)
mask = np.abs(channel_C3) > 75e-6
print('Artefact samples:', mask.sum())

# --- Advanced: select channels by index list ---
motor_idx = [6, 7, 8]             # hypothetical C3, Cz, C4 indices
motor_eeg = eeg[motor_idx, :]      # shape (3, 2560)
```

NumPy: The Heart of EEG Signal Processing

Vectorised Operations — No Loops Needed

```
fs = 256.0
t  = np.linspace(0, 1, int(fs))    # 1 second

# Generate a synthetic EEG-like signal: sum of sine waves
alpha  = 20e-6 * np.sin(2 * np.pi * 10 * t)   # 10 Hz alpha, 20 µV
beta   = 5e-6  * np.sin(2 * np.pi * 20 * t)   # 20 Hz beta,   5 µV
noise  = 3e-6  * np.random.randn(len(t))       # Gaussian noise

signal = alpha + beta + noise                  # element-wise sum

# Statistical summary
print(f'Mean  : {signal.mean()*1e6:.2f} µV')
print(f'Std   : {signal.std()*1e6:.2f} µV')
print(f'Range : {signal.ptp()*1e6:.2f} µV')   # peak-to-peak

# Signal processing operations
signal_norm = (signal - signal.mean()) / signal.std()   # z-score
signal_uV   = signal * 1e6                              # convert V → µV

# Matrix multiplication (used in spatial filtering)
W = np.random.randn(64, 64)    # spatial filter matrix
data = np.random.randn(64, 1024)
filtered = W @ data            # @ is matrix multiplication operator
print(filtered.shape)          # (64, 1024)
```

NumPy: The Heart of EEG Signal Processing

Useful NumPy Functions for EEG

Function	Description	EEG Use Case
np.mean(x, axis=)	Mean along an axis	Average across epochs (axis=0) or time (axis=-1)
np.std(x, axis=)	Standard deviation	Signal amplitude measure; z-scoring
np.abs(x)	Absolute value	Amplitude envelope; artefact detection
np.fft.fft(x)	Fast Fourier Transform	Frequency spectrum of a signal
np.linspace(a,b,n)	Evenly spaced numbers	Time axis construction
np.where(cond)	Indices where condition is true	Find artefact locations
np.concatenate([a,b])	Join arrays	Concatenate EEG runs
np.vstack / hstack	Stack arrays vertically / horizontally	Build feature matrices

Matplotlib: Visualising EEG Data

- Matplotlib is the standard Python plotting library. MNE-Python uses it internally for all its built-in visualisations, and you can extend or modify MNE plots using standard Matplotlib commands

- Figure and Axes Architecture

- Every Matplotlib plot consists of a Figure (the canvas) containing one or more Axes (individual subplots). Understanding this hierarchy is key to customising plots

- **Matplotlib: Visualising EEG Data**

```
import matplotlib.pyplot as plt
import numpy as np

fs = 256
t  = np.linspace(0, 2, fs * 2)    # 2 seconds
signal = np.sin(2 * np.pi * 10 * t) * 20   # 10 Hz, ±20 µV

# --- Create figure with one axes ---
fig, ax = plt.subplots(figsize=(10, 3))

ax.plot(t, signal, color='steelblue', linewidth=1.5, label='10 Hz sine')

# Labels and title
ax.set_xlabel('Time (s)', fontsize=12)
ax.set_ylabel('Amplitude (µV)', fontsize=12)
ax.set_title('Synthetic EEG Signal', fontsize=13, fontweight='bold')

# Grid and legend
ax.grid(True, alpha=0.3)
ax.legend(fontsize=11)

# Time window: show only 0–1 s
ax.set_xlim(0, 1)

plt.tight_layout()              # prevent label clipping
plt.savefig('my_plot.png', dpi=150)  # save before show()
plt.show()
```

- Multiple Subplots

```
fig, axes = plt.subplots(3, 1, figsize=(12, 6), sharex=True)

channels_to_plot = ['Fp1', 'C3', 'O1']
colours = ['tomato', 'steelblue', 'seagreen']

for ax, ch, col in zip(axes, channels_to_plot, colours):
    ax.plot(t, np.random.randn(len(t)) * 20, color=col, lw=0.8)
    ax.set_ylabel(ch, fontsize=10)
    ax.grid(True, alpha=0.2)

axes[-1].set_xlabel('Time (s)')
fig.suptitle('Three EEG Channels', fontsize=13, fontweight='bold')
plt.tight_layout()
plt.show()
```

Introduction to MNE-Python

- MNE-Python is the leading open-source library for EEG and MEG analysis. It provides a complete workflow from raw data loading to statistical analysis and source imaging, following the standards used in academic and clinical neuroscience

MNE Object	What it contains	When created
Raw	Continuous EEG: all channels, full recording, time axis, channel info	Loading from file
Epochs	Segmented EEG: 3D array (trials × channels × times), locked to events	Epoching Raw with events
Evoked	Trial-average ERP: 2D array (channels × times)	epochs.average()
TFR	Time-frequency power: 3D (channels × freqs × times)	tfr_morlet() or tfr_multitaper()

Format	Extension	Description
EDF / EDF+	.edf	European Data Format — most clinical EEG systems (Nihon Kohden, Natus, …)
BDF	.bdf	BioSemi Data Format — common in BioSemi ActiveTwo systems
FIF	.fif	MNE native format — preserves all metadata; fast I/O
SET (EEGLAB)	.set	Widely used in EEGLAB; MNE can import .set files directly
BrainVision	.vhdr	ActiChamp, BrainAmp — common in European labs
GDF	.gdf	General Data Format — used in BCI competitions
Neuroscan CNT	.cnt	Compumedics Neuroscan format

Introduction to MNE-Python

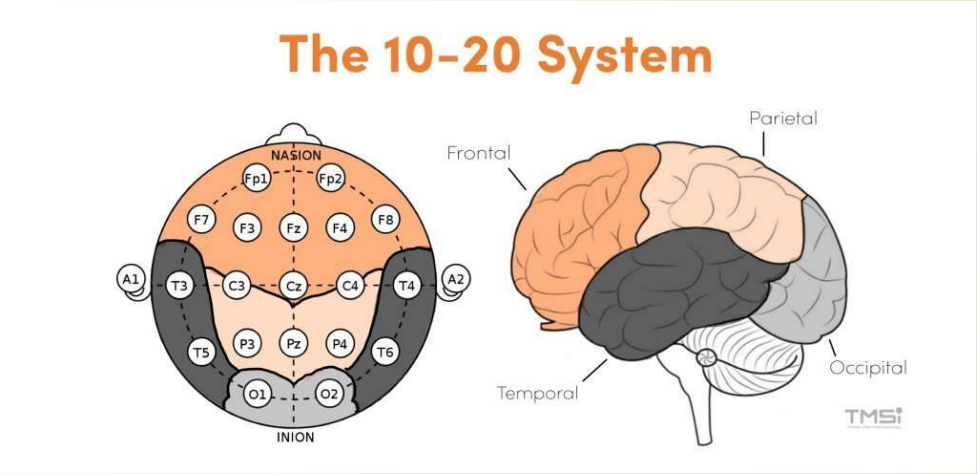
- MNE-Python is the leading open-source library for EEG and MEG analysis. It provides a complete workflow from raw data loading to statistical analysis and source imaging, following the standards used in academic and clinical neuroscience

★ Note
All formats are loaded with the same API:
raw = mne.io.read_raw_edf('file.edf', preload=True)
raw = mne.io.read_raw_bdf('file.bdf', preload=True)
raw = mne.io.read_raw_brainvision('file.vhdr', preload=True)
raw = mne.io.read_raw_eeglab('file.set', preload=True)
Setting preload=True loads the full data into RAM — required for filtering and ICA.

- The 10-20 Electrode System

- MNE-Python is the leading open-source library for EEG and MEG analysis. It provides a complete workflow from raw data loading to statistical analysis and source imaging, following the standards used in academic and clinical neuroscience

Letter Prefix	Region	Example Channels
Fp	Frontal-polar (anterior prefrontal)	Fp1, Fp2
F	Frontal	F3, F4, Fz, F7, F8
C	Central (primary motor/somatosensory)	C3, C4, Cz
T	Temporal	T7, T8, TP9, TP10
P	Parietal	P3, P4, Pz, P7, P8
O	Occipital (primary visual)	O1, O2, Oz



- Loading and Exploring EEG Data

- In this section we load the PhysioNet Motor Imagery EEG dataset using MNE's built-in download utilities. This dataset contains 109 subjects performing four motor imagery tasks recorded with 64 EEG channels at 160 Hz
- Loading the PhysioNet BCI Dataset

```
import mne
from mne.datasets import eegbci
from mne.io import concatenate_raws, read_raw_edf

mne.set_log_level('WARNING')    # suppress verbose output
# Load subject 1, motor imagery runs (runs 6 and 10) ------  Run 6  → left / right hand imagery  -----  Run 10 → left / right hand imagery (second session)
subject = 1
runs    = [6, 10]

# Download (first time) and return file paths
raw_files = eegbci.load_data(subject, runs, verbose=False)
print('Files:', raw_files)

# Read all EDF files and concatenate into one continuous recording
raw = concatenate_raws([    read_raw_edf(f, preload=True, verbose=False) for f in raw_files  ])

# Standardise channel names (PhysioNet uses non-standard names)
eegbci.standardize(raw)

# Assign 10-20 montage for electrode positions
montage = mne.channels.make_standard_montage('standard_1005')
raw.set_montage(montage, verbose=False)
print(raw)    # prints summary of the Raw object
```

- Loading and Exploring EEG Data
- The Raw object is MNE's container for continuous EEG recordings. It stores both the signal data and all associated metadata

```
# --- Basic properties ---
print('Channels     :', len(raw.ch_names))
print('Channel names:', raw.ch_names[:5], '...')
print('Sampling rate:', raw.info['sfreq'], 'Hz')
print('Duration     :', raw.times[-1], 's')
print('n_samples    :', len(raw.times))

# --- raw.info is the metadata dictionary ---
print('\nraw.info keys:', list(raw.info.keys())[:8])
print('sfreq  :', raw.info['sfreq'])
print('nchan  :', raw.info['nchan'])
print('bads   :', raw.info['bads'])    # manually marked bad channels

# --- Extract data as NumPy array ---
# Returns shape (n_channels, n_samples)
data, times = raw.get_data(return_times=True)
print('\nData shape:', data.shape)     # (64, n_samples)
print('Times shape:', times.shape)

# Access a specific channel by name
c3_idx   = raw.ch_names.index('C3')
c3_data  = data[c3_idx, :]            # all samples for C3
c3_uV    = c3_data * 1e6              # convert V → µV
print(f'C3: mean={c3_uV.mean():.2f} µV, std={c3_uV.std():.2f} µV')
```

★ Note

MNE stores EEG in Volts (V) internally, not microvolts (µV). To display in µV, multiply by 1e6.

Typical EEG amplitudes are 1–100 µV, so you will often see values like 25e-6 (= 25 µV).

- Loading and Exploring EEG Data
- Visualising Raw EEG

```
# Scrollable browser (interactive — close window to continue)
raw.plot(
    duration   = 5,          # show 5 seconds at a time
    n_channels = 20,         # show 20 channels per page
    scalings   = 'auto',     # auto-scale each channel
    title      = 'Raw EEG — Motor Imagery')

# Electrode montage on scalp
raw.plot_sensors(show_names=True, kind='topomap')

# Power Spectral Density (quick frequency overview)
raw.compute_psd(fmax=60).plot()

# Quick plot of one channel manually
import matplotlib.pyplot as plt
data, t = raw.get_data(return_times=True)
c3_idx  = raw.ch_names.index('C3')

fig, ax = plt.subplots(figsize=(12, 3))
ax.plot(t[:1280], data[c3_idx, :1280] * 1e6, lw=0.7, color='steelblue')
ax.set_xlabel('Time (s)')
ax.set_ylabel('µV')
ax.set_title('C3 — First 5 s of raw recording')
plt.tight_layout()
plt.show()
```

- Events and Annotations
- In EEG research, the continuous recording is marked with time stamps indicating when stimuli were presented or when the subject performed an action. MNE represents these marks in two related ways

Extracting Events

```
# Convert Annotations in the Raw object to an integer events array
events, event_id = mne.events_from_annotations(raw, verbose=False)

print('events shape :', events.shape)    # (n_events, 3)
print('event_id     :', event_id)        # {'T0':1, 'T1':2, 'T2':3}
print('First 5 events:')
print(events[:5])
# Each row: [sample_number, 0, event_code]
#           ↑ time stamp    ↑   ↑ condition

# ── PhysioNet BCI event codes ──────────────────────────────────
# T0 → rest / baseline
# T1 → left hand motor imagery (or fist clenching)
# T2 → right hand motor imagery (or feet imagery)

# Count events per condition
for name, code in event_id.items():
    count = (events[:, 2] == code).sum()
    print(f'  {name}: {count} events')
```

- Events and Annotations
- In EEG research, the continuous recording is marked with time stamps indicating when stimuli were presented or when the subject performed an action. MNE represents these marks in two related ways

- **Visualising the Event Timeline**

```
fig = mne.viz.plot_events(
    events,
    event_id  = event_id,
    sfreq     = raw.info['sfreq'],
    first_samp = raw.first_samp,
    color      = {'T0': 'grey', 'T1': 'steelblue', 'T2': 'tomato'}
)
```

- Events and Annotations
- In EEG research, the continuous recording is marked with time stamps indicating when stimuli were presented or when the subject performed an action. MNE represents these marks in two related ways

★ Key Concept
Motor Imagery Paradigm (PhysioNet BCI Dataset):
T0 —— Rest / baseline (subject relaxes, eyes open)
T1 —— Imagine left hand movement (or open/close both fists)
T2 —— Imagine right hand movement (or open/close both feet)
Each trial: 4 seconds of imagery, separated by rest periods.
The EEG changes during motor imagery even without actual movement.
This is the principle behind Motor Imagery Brain-Computer Interfaces.

- **Visualising the Event Timeline**

```
fig = mne.viz.plot_events(
    events,
    event_id  = event_id,
    sfreq     = raw.info['sfreq'],
    first_samp = raw.first_samp,
    color      = {'T0': 'grey', 'T1': 'steelblue', 'T2': 'tomato'}
)
```

EEG Preprocessing

Why Do We Preprocess EEG?

Raw EEG signals recorded directly from amplifiers contain a mixture of brain activity, physiological noise from the body, and environmental interference. Without preprocessing, downstream analyses are unreliable. The table below summarises the main contamination sources and their frequency ranges:

Artefact Source	Frequency Range	Amplitude	Affected Channels	Removal Strategy
Eye blinks / movements (EOG)	DC – 10 Hz	50–2000 µV	Fp1, Fp2, AF channels	ICA component rejection
Muscle activity (EMG)	20 – 300 Hz	10–1000 µV	Temporal, frontal	Low-pass filter, epoch rejection
Cardiac pulse (ECG)	1 – 5 Hz	20–200 µV	Temporal, distributed	ICA
Power-line noise	50 Hz (EU) / 60 Hz (US)	1–100 µV	All channels	Notch filter
Slow drifts (sweat, movement)	DC – 0.5 Hz	100–1000 µV	All channels	High-pass filter ≥ 0.5 Hz
Cable / electrode movement	Broadband transients	Very high	Local channels	Bad channel interpolation, epoch rejection

★ Key Concept

The Standard Preprocessing Order:

1. **Load raw data**
2. **Set electrode montage**
3. **Re-reference (average or linked mastoids)**
4. **Notch filter (remove power-line noise)**
5. **Bandpass filter (1–40 Hz for most paradigms)**
6. **Identify and interpolate bad channels**
7. **ICA — fit on filtered data, remove artefact components**
8. **Epoch around events**
9. **Baseline correction**
10. **Amplitude-based epoch rejection**

Order matters! Always filter BEFORE ICA, and ICA BEFORE epoching.