## 1.installing needed packages and giving access to the google drive

In order to run the code we need some libraries and packages; the most important package here is MNE. We will use MNE to import data, drop channels, cut data, visualize data, and so on. By giving access to our google drive we can get access to our recordings, python functions, etc.

```
# installing packages
%pip install mne
import joblib
import matplotlib
import numpy as np
import scipy
import mne
%pip install pyEDFlib
from pyedflib import highlevel
```

```
     Installing collected packages: mne
```

```
from google.colab import drive
drive.mount('/gdrive')
%cd /gdrive/MyDrive/
%ls
```

## 2.import data

### 2.1.Import data as EDF format

The function to import .edf data to MNE is **mne.io.read_raw_edf**. I write"preload=True" in order to have access to it later on.

```
data = mne.io.read_raw_edf('record_name.edf',preload=True)
```

**2.2.import data as fdt or set** when we use eeglab( in matlab) to edit our data, the data will be saved as fdt or set. with mne.io.read_raw_eeglab function we can import that kind of data into MNE (python)

```
data =mne.io.read_raw_eeglab('record_name.set',preload= True)
```

### 2.3.importing data as fif

fif is the format that MNE saves the data. the function is mne.io.read_raw_fif

```
data = mne.io.read_raw_fif('record_name.fif')
```

### 2.4. import data as h5 file to mne python

I accessed data using h5py and then put all channels signal in one matrix where the shape of it is (number-of-channels, samples). Then I create my data information and import data into MNE.

```
import h5py
import numpy as np
# note that in some of dreem's recording instead of 'Channel1' it is 'eeg1'
# Open the HDF5 file
with h5py.File('record_name.h5', 'r') as hf:
    f7_o1 = hf['channel1']['raw'][:]
    f8_o2 = hf['channel2']['raw'][:]
    fp1_f8 = hf['channel3']['raw'][:]
    f8_f7 = hf['channel4']['raw'][:]
    fp1_o1 = hf['channel5']['raw'][:]
    fp1_o2 = hf['channel6']['raw'][:]
    fp1_f7 = hf['channel7']['raw'][:]
data = np.vstack([f7_o1, f8_o2, fp1_f8, f8_f7, fp1_o1, fp1_o2, fp1_f7])
# Create an MNE info object
ch_names = ['F7_O1', 'F8_O2', 'Fp1_F8', 'F8_F7', 'Fp1_O1', 'Fp1_O2', 'Fp1_F7']
ch_types = ['eeg'] * len(ch_names)
sfreq = 250
info = mne.create_info(ch_names=ch_names, ch_types=ch_types, sfreq=sfreq)
```

```
 #Create an MNE RawArray object for the concatenated data
raw = mne.io.RawArray(data, info)
```

## 3.Filtering the data

```
# Filter settings
low_cut = 0.3
hi_cut  = 35
#function
dataFilter = raw.filter(low_cut,hi_cut)
dataFilter.save('dataFilter.fif',overwrite=True)


# Apply a notch filter to remove 60 Hz noise
raw.notch_filter(freqs=60.0)
```

### 3.1Saving the Filtered data

Note that the MNE save data in fif format. This step is optional.

```
dataFilter.save('dataFilter.fif',overwrite=True)
```

## 4.Croping the data from time 1 to time 2

Here we sellect the time priod of data that we are interested; note that we can do it in the next section where we are sellecting the channel

```
t1= 0
t2= 3600
dataFilterCroped= dataFilter.crop(tmin= t1 ,tmax=t2)
```

## 5.Add annotation

```
# extract data from hypnogram.txt file
# Open the input file and read its contents
with open('hypnogram.txt', 'r') as input_file:
    input_data = input_file.readlines()
# Remove all columns except the first one and apply the mapping to the values
mapping = {'SLEEP-S0': 'wake', 'SLEEP-REM': 'Rem', 'SLEEP-S1': 'N1', 'SLEEP-S2': 'N2', 'SLEEP-3':'N3', 'SLEEP-MT': 'NA'}
output_data = [mapping.get(line.strip().split()[0], line.strip().split()[0]) for line in input_data]
# Create  three variables from hypnogram data for annotation
onset = [i * 30 for i in range(len(output_data))]
duration = [30] * len(output_data)
description = output_data
# Write the result to the output file
with open('output.txt', 'w') as output_file:
    for onset_value, duration_value, description_value in zip(onset, duration, description):
        output_file.write(f'{onset_value} {duration_value} {description_value}\n')
# Create a copy of the raw object and add the annotations to the copy
raw_with_annot = raw.copy()
annot = mne.Annotations(onset=onset, duration=duration, description=description)
raw_with_annot.set_annotations(annot)
# Export to a new EDF file
mne.export.export_raw('annotated_data.edf', raw, fmt='auto', overwrite=True)
'''after checking many things I believe the problem is when I export the data, before exporting and after annotation
the data has both signals and annotation but after exporting I lose the signals'''
```

## 6.Creat hypnogram graph

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

# Read data from file
data = pd.read_table('hypnogram.txt', header=None)
oldvalues = ['SLEEP-S0', 'SLEEP-REM', 'SLEEP-S1', 'SLEEP-S2', 'SLEEP-S3', 'SLEEP-MT']
newvalues = ['5', '4', '3', '2', '1', '0']
data[0] = data[0].replace(oldvalues, newvalues)
```

```
hypno = data[0]
stage_data = np.repeat(hypno, 30)
fs = 0.0333  # Sampling rate in Hz
t = np.arange(len(hypno))/fs
stage_data_num = stage_data.astype(int)
plt.stairs(stage_data_num, linewidth=2)
# Set the y-axis limits to show the different sleep stages
plt.ylim([-0.5, 6.5])
# Label the y-axis with the different sleep stages
plt.yticks([0, 1, 2, 3, 4, 5], ['NA', 'N3', 'N2', 'N1', 'rem', 'wake'])
plt.ylabel('Sleep stage')
# Label the x-axis with the time
plt.xlabel('Time (minutes)')
plt.show()
```

## 7.extract the time points for each stage periods

```
import pandas as pd
print('''
0= wake
1=N1
2=N2
3=N3
4=REM
''')
# Read data from file
data = pd.read_csv('hypnogram.txt', header=None, sep='\t')

# Define mapping of old values to new values
oldvalues = ['SLEEP-S0', 'SLEEP-REM', 'SLEEP-S1', 'SLEEP-S2', 'SLEEP-S3', 'SLEEP-MT']
newvalues = ['0', '4', '1', '2', '3', '5']

# Replace old values with new values
data[0] = data[0].replace(oldvalues, newvalues)

# Convert the data type of the first column to integer
data[0] = data[0].astype(int)

# Get the sampling frequency
fs = 1/30  # 30 seconds per sample

# Loop over each sleep stage and extract the periods
for stage in range(6):
    # Get the indices where the sleep stage is equal to the current stage
    indices = data.index[data[0] == stage]

    # If there are no periods for this stage, continue to the next stage
    if len(indices) == 0:
        continue

    # Initialize the start and end times of the first period
    start_time = indices[0] * 30
    end_time = start_time

    # Loop over the remaining indices and extract the periods
    for i in range(1, len(indices)):
        # If the current index is consecutive to the previous index, update the end time
        if indices[i] == indices[i-1]+1:
            end_time = indices[i] * 30
        # Otherwise, print the start and end times of the previous period and update the start time
        else:
            print(f"Sleep stage {stage} from {start_time:.1f}s to {end_time:.1f}s")
            start_time = indices[i] * 30
            end_time = start_time

    # Print the start and end times of the last period
    print(f"Sleep stage {stage} from {start_time:.1f}s to {end_time:.1f}s")
```

## 8.Get data for each channel as a numpy array in 1D

**extraxting channels using MNE** t1 equals to start point and t2 equals to end point

```
# t1=0
#t2= 3600
# you can delete tmin and tmax to get the data for whole recording
#channelName_data = raw.get_data(picks= 0, tmin= t1, tmax=t2)
F7_O1_data=dataFilter.get_data(picks=0, tmin= t1, tmax=t2)
F8_O2_data=dataFilter.get_data(picks=1, tmin= t1, tmax=t2)
Fp1_F8_data= dataFilter.get_data(picks=2, tmin= t1, tmax=t2)
F8_F7_data= dataFilter.get_data(picks=3, tmin= t1, tmax=t2)
Fp1_O1_data= dataFilter.get_data(picks=4, tmin= t1, tmax=t2)
Fp1_O2_data= dataFilter.get_data(picks=5, tmin= t1, tmax=t2)
Fp1_F7_data= dataFilter.get_data(picks=6, tmin= t1, tmax=t2)
```

**extracting each channel for edf data with out using MNE**

```
#signals, signal_headers, header = highlevel.read_edf('dreem95.edf')  # reads in the signal data, header for each signal, and the overall edf
F7_O1_data = signals[0]  # in this edf the first signal is data from the F7_O1 channel (look at signal_headers to determine the label for eac
fs = signal_headers[0]['sample_rate']  # Extract the sampling frequency for the F7_O1 signal
F8_O2_data = signals[1]
Fp1_F8_data = signals[2]
F8_F7_data = signals[3]
Fp1_O1_data = signals[4]
Fp1_O2_data = signals[5]
Fp1_F7_data = signals[6]
```

**Checking the shape**: the shape of your array should be (1,number of samples), if it is (number of samples,) use transpose function to change the shape.

```
print(channel_data.shape)# to see the shape
transposed_data = np.transpose(channel_data)# to change the shape
```

Executing (11s) <cell line: 2> › run_line_magic() › _pip_magic() › system() › _system_compat() › _run_command() › _monitor_process() › _poll_process()