```
import numpy as np
import pandas as pd
from scipy import signal

# CALCULATE MEDIAN

def calculate_median(data):
    data = pd.concat([data, data.groupby(['subjectNbr', 'channel', 'cleaning',
'normalization', 'wdwLen', 'butter'], observed=False)
                      .median(numeric_only = True)
                      .reset_index()
                      .assign(windowNbr=99)])
    data = data.dropna()
    data[['subjectNbr', 'dataset', 'diagnosis', 'windowNbr']] = data[['subjectNbr', 'dataset',
'diagnosis', 'windowNbr']].astype('uint8')
    data[['normalization', 'wdwLen', 'butter', 'cleaning']] = data[['normalization', 'wdwLen',
'butter', 'cleaning']].astype('int64')
    data.reset_index(drop=True)
    return data

# CALCULATE ALPHA BAND

def butter_bandpass(lowcut, highcut, fs, order):
    return signal.butter(order, [lowcut, highcut], fs=fs, btype='bandpass', output='sos')

def butter_bandpass_filter(data, lowcut, highcut, fs, order):
    # get original index to later insert result back into original structure
    original_index = data.index
    # drop nans to ensure filtering works
    data_clean = data.dropna()
    # filter signal
    sos = butter_bandpass(lowcut, highcut, fs, order=order)
    y = signal.sosfiltfilt(sos, data_clean)
    # insert result back into original structure
    result = pd.Series(np.nan, index=original_index)
    result[data_clean.index] = y
    return result


# FREQUENCY MARKERS

def alpha_peak_frequency(a):

    # transformation to frequency spectrum
    freqs, psd = signal.welch(a.to_numpy(), 256, nperseg=512)

    # extract alpha band
    idx_alpha = np.logical_and(freqs >= 8, freqs <= 13)
```

```
    psd_alpha = psd[idx_alpha]
    freqs_alpha = freqs[idx_alpha]

    # find peaks
    idx_peak = np.where(psd_alpha==max(psd_alpha))
    result = freqs_alpha[idx_peak][0]
    return result


def abs_centroid_frequency(a):
    # transformation to frequency spectrum
    freqs, psd = signal.welch(a.to_numpy(), 256, nperseg=512)

    # extract alpha band
    idx_alpha = np.logical_and(freqs >= 8, freqs <= 13)
    psd_alpha = psd[idx_alpha]
    freqs_alpha = freqs[idx_alpha]

    # calculate centroid frequency
    return np.sum(freqs_alpha*psd_alpha)/np.sum(psd_alpha)


def rel_centroid_frequency(a):
    # transformation to frequency spectrum
    freqs, psd = signal.welch(a.to_numpy(), 256, nperseg=512)

    # extract alpha band
    idx_alpha = np.logical_and(freqs >= 8, freqs <= 13)
    psd_alpha = psd[idx_alpha]
    freqs_alpha = freqs[idx_alpha]
    # calculate centroid frequency
    alpha_centroid_freq = np.sum(freqs_alpha*psd_alpha)/np.sum(psd_alpha)

    # extract full band
    idx_full = np.logical_and(freqs >= 1, freqs <= 40)
    psd_full = psd[idx_full]
    freqs_full = freqs[idx_full]
    # calculate centroid frequency
    total_centroid_freq = np.sum(freqs_full*psd_full)/np.sum(psd_full)

    return alpha_centroid_freq/total_centroid_freq


def total_centroid_frequency(a):
    # transformation to frequency spectrum
    freqs, psd = signal.welch(a.to_numpy(), 256, nperseg=512)

    # extract full band
```

```
    idx_full = np.logical_and(freqs >= 1, freqs <= 40)
    psd_full = psd[idx_full]
    freqs_full = freqs[idx_full]

    # calculate centroid frequency
    total_centroid_freq = np.sum(freqs_full*psd_full)/np.sum(psd_full)
    return total_centroid_freq


def bandpower(data, low=1, high=40, relative=False):

    from scipy.integrate import simpson
    from scipy.signal import welch

    sf = 256
    window_new = data

    # Define window length
    nperseg = 512

    # Compute the modified periodogram (Welch)
    freqs, psd = welch(window_new.to_numpy(), sf, nperseg=nperseg)

    # Frequency resolution
    freq_res = freqs[1] - freqs[0]

    # Find closest indices of band in frequency vector
    idx_alpha = np.logical_and(freqs >= low, freqs <= high)

    # Integral approximation of the spectrum using Simpson's rule
    bp = simpson(psd[idx_alpha], dx = freq_res)

    if relative:
        bp /= simpson(psd, dx=freq_res)

    return bp
```

```
# ENVELOPE MARKERS
# envelope markers were calculated with statsmodel (stats) and numpy (np) as follows
# envelope kurtosis: stats.kurtosis()
# envelope skewness: stats.skew()
# envelope median:np.median()
# envelope interquartile range: stats.iqr()
# envelope variance: np.var()
# envelope range: np.ptp()
```