Ejercicio Final - Análisis de Datos Científicos

Materia Análisis de Datos Científicos y Geográficos-

Comisión: ECD.2023.B: Datos Científicos

Especialización en Ciencia de Datos - ITBA

Alumno: Germán Leonarhdt
Profesor: Rodrigo Ramele

Fecha: 10-8-2024

Bases de datos de las muestras

A los fines de este análisis, solo utilizaré las bases de datos de baseline y pestañeos.

Como primer paso, importo las bases e presento las primeras lineas y los encabezados, que contienen la información de la muestra capturada mediante el dispositivo en clases.

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import requests
from io import StringIO

# Importamos bases Baseline y pestañeos

# Baseline
baseline = pd.read_csv('data/baseline.dat', delimiter=' ', names = ['timestamp','counter','eeg','attention','meditation','
baseline = baseline.values
baseline_eeg = baseline[:,2]

print('Baseline - Estructura de la informacion:')
baseline_df = pd.DataFrame(baseline)
print(baseline_df.head())


# Pestañeos
pestaneos = pd.read_csv('data/pestaneos.dat', delimiter=' ', names = ['timestamp','counter','eeg','attention','meditation'
pestaneos = pestaneos.values
pestaneos_eeg = pestaneos[:,2]

print('Pestañeos - Estructura de la informacion:')
pestaneos_df = pd.DataFrame(pestaneos)
print(pestaneos_df.head())
```

```
Baseline - Estructura de la informacion:
              0     1      2    3    4    5
0  1.720127e+09  78.0   90.0  0.0  0.0  0.0
1  1.720127e+09  79.0  104.0  0.0  0.0  0.0
2  1.720127e+09  80.0  104.0  0.0  0.0  0.0
3  1.720127e+09  81.0  100.0  0.0  0.0  0.0
4  1.720127e+09  82.0  105.0  0.0  0.0  0.0
Pestañeos - Estructura de la informacion:
              0     1     2    3    4    5
0  1.720128e+09  49.0   0.0  0.0  0.0  0.0
1  1.720128e+09  50.0 -17.0  0.0  0.0  0.0
2  1.720128e+09  51.0   9.0  0.0  0.0  0.0
3  1.720128e+09  52.0  52.0  0.0  0.0  0.0
4  1.720128e+09  53.0  69.0  0.0  0.0  0.0
```

Control de las bases de datos

Como segundo paso, me aseguro que las bases de datos no contengan valores nulos o duplicados.

In [ ]:
```
# Chequeamos nulos y duplicados

# Baseline
baseline_df = pd.DataFrame(baseline)
baseline_nonulls = baseline_df.dropna()
baseline_null=baseline_df.isnull().sum().sum()
baseline_noduplicates = baseline_df.drop_duplicates()
print('El dataset de Baseline tiene',len(baseline_df),'observaciones. Y si removemos nulos ',len(baseline_nonulls),'observ
print('Si removemos duplicados,',len(baseline_noduplicates),'observaciones, tambien la misma cantidad')

# Pestañeos
```

```
pestaneos_df = pd.DataFrame(pestaneos)
pestaneos_nonulls = pestaneos_df.dropna()
pestaneos_null=pestaneos_df.isnull().sum().sum()
pestaneos_noduplicates = pestaneos_df.drop_duplicates()
print('El dataset de Pestañeos tiene',len(pestaneos_df),'observaciones. Y si removemos nulos ',len(pestaneos_nonulls),'obs
print('Y si removemos duplicados,',len(pestaneos_noduplicates),'observaciones, tambien la misma cantidad')
```

```
El dataset de Baseline tiene 30850 observaciones. Y si removemos nulos  30850 observaciones, la misma cantidad. Es decir, e
l dataset de Baseline tiene 0 nulos.
Si removemos duplicados, 30850 observaciones, tambien la misma cantidad
El dataset de Pestañeos tiene 30826 observaciones. Y si removemos nulos  30826 observaciones, la misma cantidad Es decir, e
l dataset de Pestañeos tiene 0 nulos.
Y si removemos duplicados, 30826 observaciones, tambien la misma cantidad
```

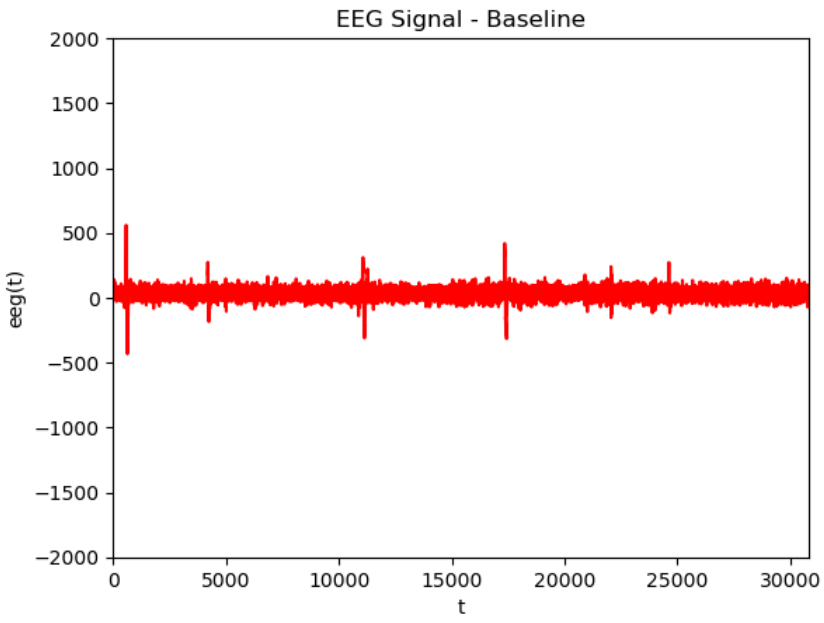
Es posible observar que las dos bases de datos no tienen nulos ni duplicados.

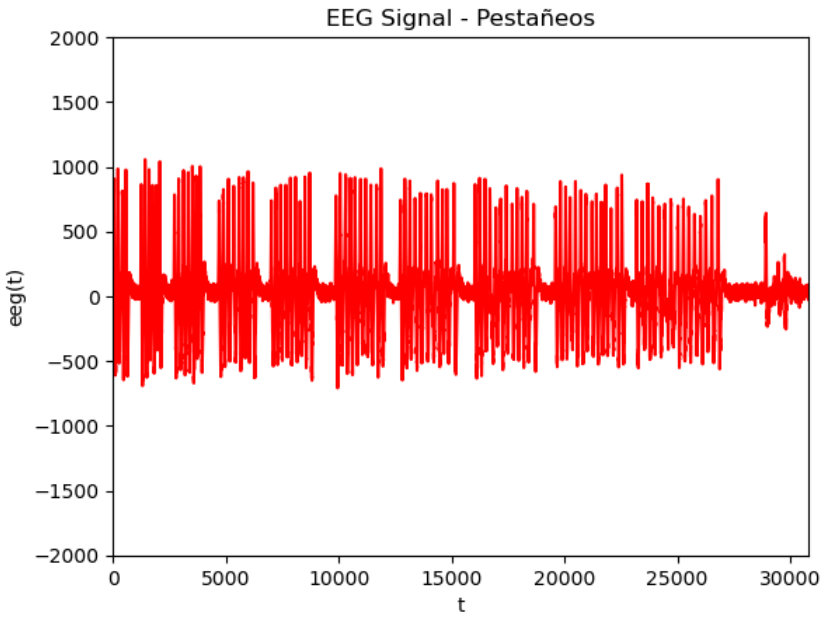
Series graficadas

Gráficos sin filtros

Como siguiente paso, graficamos las series sin modificar, ni aplicar ningún filtro.

In [ ]:
```
# Baseline
plt.plot(baseline_eeg,'r', label='EEG')
plt.xlabel('t');
plt.ylabel('eeg(t)');
plt.title(r'EEG Signal - Baseline')     # r'' representa un raw string que no tiene caracteres especiales
plt.ylim([-2000, 2000]);
plt.xlim([0,len(baseline_eeg)])
plt.savefig('images/Baseline_signal.png')
plt.show()

# Pestañeos
plt.plot(pestaneos_eeg,'r', label='EEG')
plt.xlabel('t');
plt.ylabel('eeg(t)');
plt.title(r'EEG Signal - Pestañeos')     # r'' representa un raw string que no tiene caracteres especiales
plt.ylim([-2000, 2000]);
plt.xlim([0,len(pestaneos_eeg)])
plt.savefig('images/Pestaneos_signal.png')
plt.show()
```





En los gráficos expuestos, es posible observar las diferencias entre uno y otro, donde la serie de pestañeos presenta una mayor variabilidad en torno a cada pestañeo.

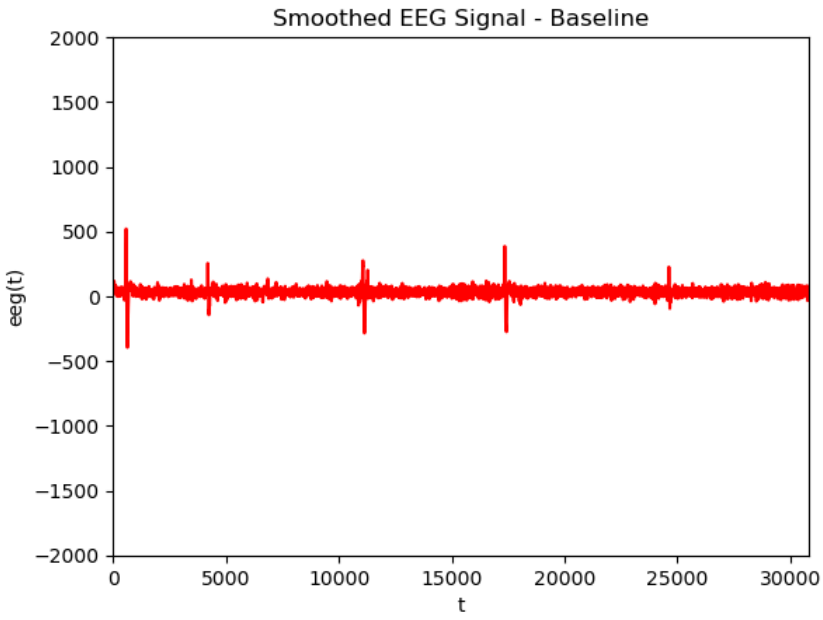
Graficos con filtros temporales

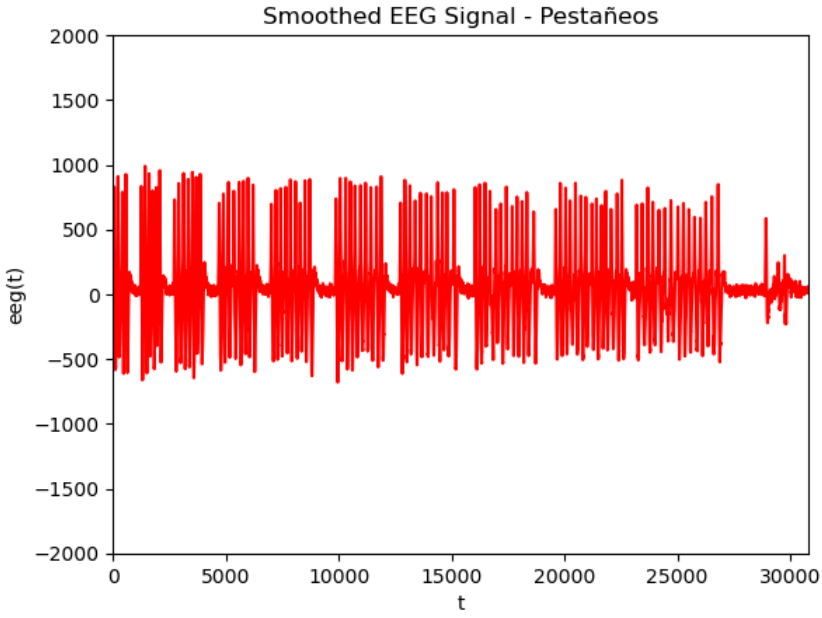
Luego, aplico filtros temporales a las series y las comparamos entre si. Llevaré adelante este proceso con una operación de convulción, y luego normalizando la serie.

Operación de convolución

En primer lugar, aplico la operacion de convulción a cada serie y grafico los resultados obtenidos.

In [ ]:
```
# Aplicamos filtro temporal

# Operacion de convulsion

# Baseline
baseline = pd.read_csv('data/baseline.dat', delimiter=' ', names = ['timestamp','counter','eeg','attention','meditation','
baseline = baseline.values
baseline_eeg = baseline[:,2]
windowlength = 10
baseline_avgeeg = np.convolve(baseline_eeg, np.ones((windowlength,))/windowlength, mode='same')
plt.plot(baseline_avgeeg,'r', label='EEG')
plt.xlabel('t');
plt.ylabel('eeg(t)');
plt.title(r'Smoothed EEG Signal - Baseline')
plt.ylim([-2000, 2000]);
plt.xlim([0,len(baseline_avgeeg)])
plt.savefig('images/baseline_smoothed.png')
plt.show()

# Pestañeos
pestaneos = pd.read_csv('data/pestaneos.dat', delimiter=' ', names = ['timestamp','counter','eeg','attention','meditation'
pestaneos = pestaneos.values
pestaneos_eeg = pestaneos[:,2]

windowlength = 10
pestaneos_avgeeg = np.convolve(pestaneos_eeg, np.ones((windowlength,))/windowlength, mode='same')
plt.plot(pestaneos_avgeeg,'r', label='EEG')
plt.xlabel('t');
plt.ylabel('eeg(t)');
plt.title(r'Smoothed EEG Signal - Pestañeos')
plt.ylim([-2000, 2000]);
plt.xlim([0,len(pestaneos_avgeeg)])
plt.savefig('images/pestaneos_smoothed.png')
plt.show()
```



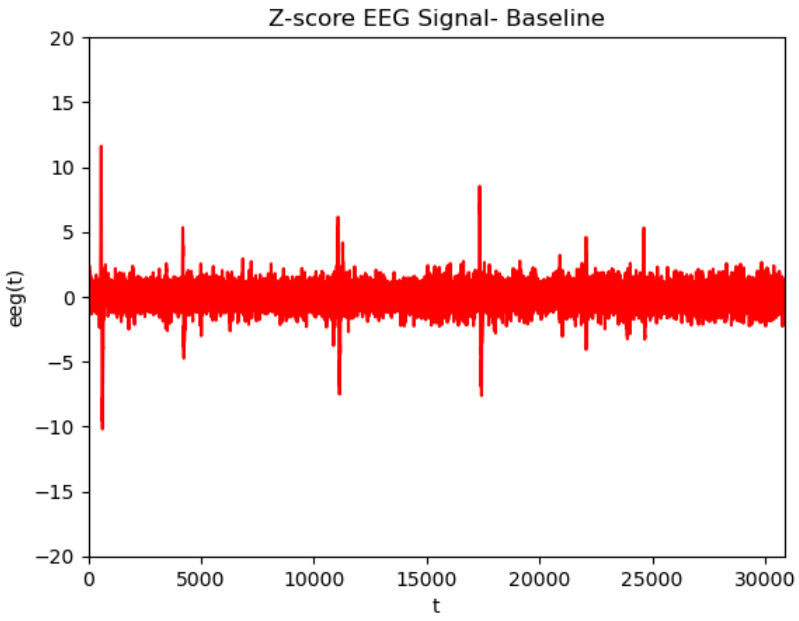


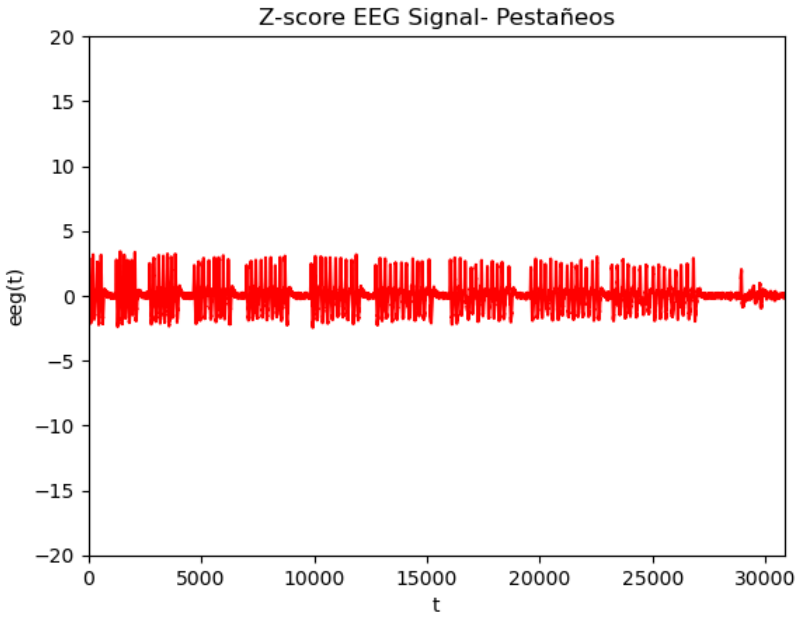
Normalización

La normalización de la serie puede ser considerado un filtro temporal. A continuación, llevo adelante la normalización de ambas series y grafico los resultados.

In [ ]:
```
# Operacion de normalizacion

def z_score_norm(arr):
    """Apply z-score normalization
        to an array or series
    """
    mean_ = np.mean(arr)
    std_ = np.std(arr)

    new_arr = [(i-mean_)/std_ for i in arr]

    return new_arr


# Baseline
baseline = pd.read_csv('data/baseline.dat', delimiter=' ', names = ['timestamp','counter','eeg','attention','meditation','
baseline = baseline.values
baseline_eeg = baseline[:,2]

baseline_eeg_zscore = z_score_norm(baseline_eeg)

plt.plot(baseline_eeg_zscore,'r', label='EEG')
plt.xlabel('t');
plt.ylabel('eeg(t)');
plt.title(r'Z-score EEG Signal- Baseline')
```

```
plt.ylim([-20, 20]);
plt.xlim([0,len(baseline_eeg_zscore)])
plt.savefig('images/baseline_zscoredeeg.png')
plt.show()

# Pestañeos
pestaneos = pd.read_csv('data/pestaneos.dat', delimiter=' ', names = ['timestamp','counter','eeg','attention','meditation'
pestaneos = pestaneos.values
pestaneos_eeg = pestaneos[:,2]

pestaneos_eeg_zscore = z_score_norm(pestaneos_eeg)

plt.plot(pestaneos_eeg_zscore,'r', label='EEG')
plt.xlabel('t');
plt.ylabel('eeg(t)');
plt.title(r'Z-score EEG Signal- Pestañeos')
plt.ylim([-20, 20]);
plt.xlim([0,len(pestaneos_eeg_zscore)])
plt.savefig('images/pestaneos_zscoredeeg.png')
plt.show()
```





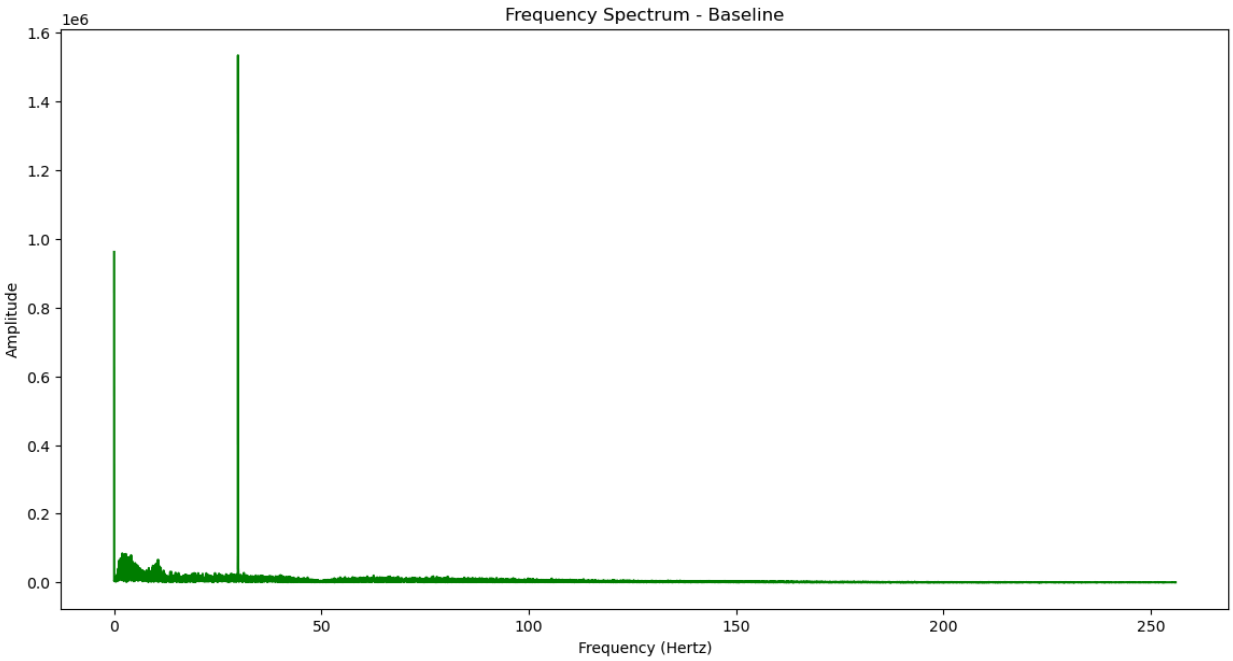
Gráficos con filtros espectrales

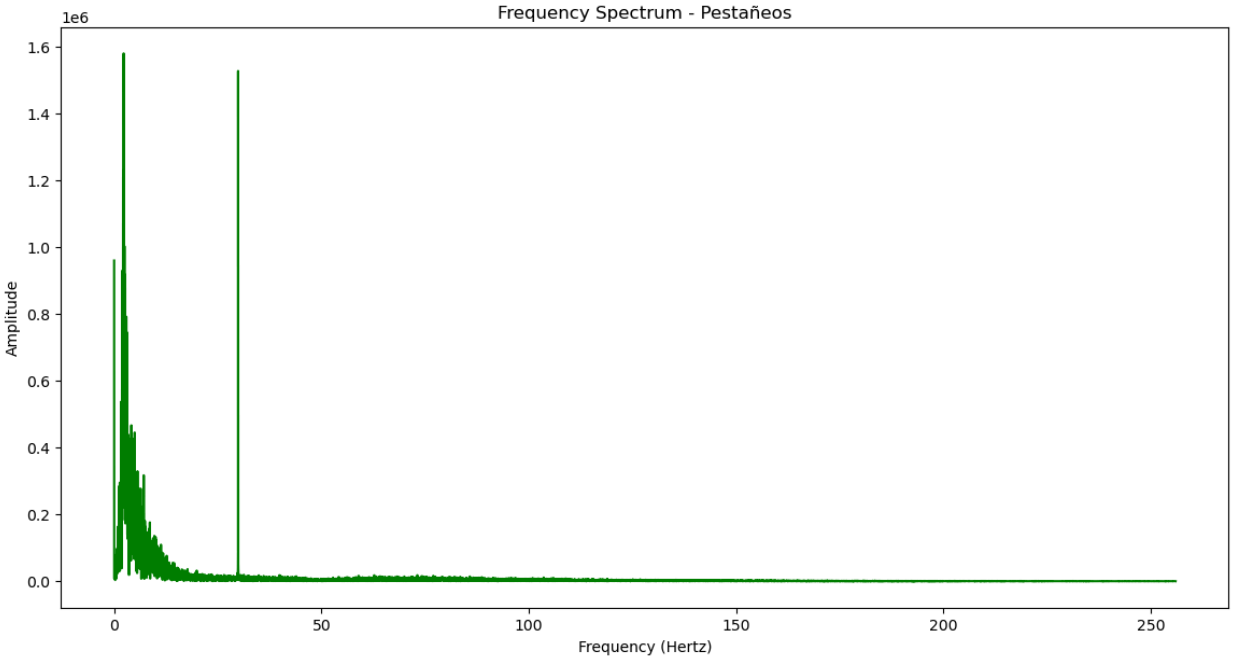
Realizo la transformada de Fourier sobre las series obtenidas, y grafico los resultados.

In [ ]:
```
# Aplicamos filtro espectral


import sys, select
```

```
import time
import datetime
import os

from scipy.fftpack import fft

import math

from scipy.signal import firwin, remez, kaiser_atten, kaiser_beta
from scipy.signal import butter, filtfilt, buttord

from scipy.signal import butter, lfilter
from scipy.fft import rfft, rfftfreq


def butter_bandpass(lowcut, highcut, fs, order=5):
    nyq = 0.5 * fs
    low = lowcut / nyq
    high = highcut / nyq
    b, a = butter(order, [low, high], btype='band')
    return b, a


def butter_bandpass_filter(data, lowcut, highcut, fs, order=5):
    b, a = butter_bandpass(lowcut, highcut, fs, order=order)
    y = lfilter(b, a, data)
    return y

def psd(y):
    # Number of samplepoints
    N = 512
    # sample spacing
    T = 1.0 / 512.0

    # Original Bandpass
    fs = 512.0
    fso2 = fs/2

    y = butter_bandpass_filter(y, 8.0, 15.0, fs, order=6)
    yf = fft(y)


    return np.sum(np.abs(yf[0:int(N/2)]))


Fs = 512.0

# Baseline
baseline = pd.read_csv('data/baseline.dat', delimiter=' ', names = ['timestamp','counter','eeg','attention','meditation','
baseline = baseline.values
baseline_eeg = baseline[:,2]

baseline_normalized_signal = baseline_eeg

N_baseline = len(baseline_normalized_signal)


# Creo una secuencia de N puntos (el largo de EEG), de 0 hasta el largo de la secuencia en segundos (N/Fs).
x = np.linspace(0.0, int(N_baseline/Fs), N_baseline)

# A esa secuencia de EEG le agrego una señal pura de 30 Hz.  Estoy ayuda a visualizar bien que la relación espectral está
baseline_normalized_signal +=  100*np.sin(30.0 * 2.0*np.pi*x)

yf = rfft(baseline_normalized_signal)
xf = rfftfreq(N_baseline, 1 / Fs)

plt.figure(figsize=(14,7))
plt.title('Frequency Spectrum - Baseline')
plt.plot(xf, np.abs(yf), color='green')
plt.ylabel('Amplitude')
plt.xlabel('Frequency (Hertz)')
plt.savefig('images/baseline_spectral.png')
plt.show()


# Pestañeos
pestaneos = pd.read_csv('data/pestaneos.dat', delimiter=' ', names = ['timestamp','counter','eeg','attention','meditation'
pestaneos = pestaneos.values
pestaneos_eeg = pestaneos[:,2]

pestaneos_normalized_signal = pestaneos_eeg

N_pestaneos = len(pestaneos_normalized_signal)


# Creo una secuencia de N puntos (el largo de EEG), de 0 hasta el largo de la secuencia en segundos (N/Fs).
```

```
x = np.linspace(0.0, int(N_pestaneos/Fs), N_pestaneos)

# A esa secuencia de EEG le agrego una señal pura de 30 Hz.  Estoy ayuda a visualizar bien que la relación espectral está
pestaneos_normalized_signal +=  100*np.sin(30.0 * 2.0*np.pi*x)

yf = rfft(pestaneos_normalized_signal)
xf = rfftfreq(N_pestaneos, 1 / Fs)

plt.figure(figsize=(14,7))
plt.title('Frequency Spectrum - Pestañeos')
plt.plot(xf, np.abs(yf), color='green')
plt.ylabel('Amplitude')
plt.xlabel('Frequency (Hertz)')
plt.savefig('images/pestaneos_spectral.png')
plt.show()
```





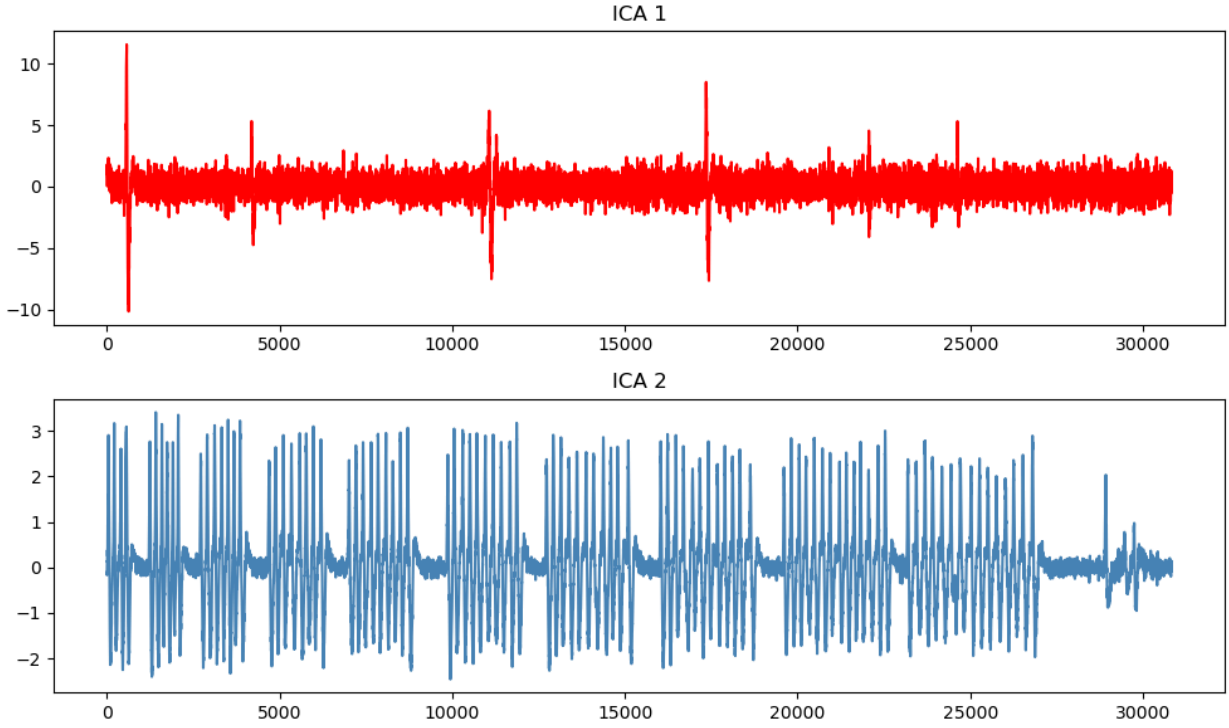
Si comparamos los gráficos, es notoria la diferencia entre ambas series luego de aplicar el filtro espectral. La serie de pestañeos presenta una mayor amplitud en frecuencias mayores que el baseline.

Filtro espacial

Si juntamos ambas series y le aplicamos un filtro espacial, deberiamos poder aislar o separar las frecuencias mas bajas presentes en el baseline y en pestañeos, del componente de pestañeos. A continuación se realiza esa operacion y se exponen los resultados obtenidos.

In [ ]:
```
# Baseline
baseline = pd.read_csv('data/baseline.dat', delimiter=' ', names = ['timestamp','counter','eeg','attention','meditation','
```

```
baseline = baseline.values
baseline_eeg = baseline[:,2]

# Pestañeos
pestaneos = pd.read_csv('data/pestaneos.dat', delimiter=' ', names = ['timestamp','counter','eeg','attention','meditation'
pestaneos = pestaneos.values
pestaneos_eeg = pestaneos[:,2]

import numpy as np
import matplotlib.pyplot as plt
from scipy import signal

from sklearn.decomposition import FastICA, PCA


min_length = min(len(baseline_eeg), len(pestaneos_eeg))

baseline_eeg_ICA = baseline_eeg[:min_length]
pestaneos_eeg_ICA = pestaneos_eeg[:min_length]

S = np.c_[baseline_eeg_ICA, pestaneos_eeg_ICA]

ica = FastICA(n_components=2)
S_ = ica.fit_transform(S)  # Reconstruct signals
A_ = ica.mixing_  # Get estimated mixing matrix
assert np.allclose(S, np.dot(S_, A_.T) + ica.mean_)


plt.figure(figsize=(10, 6))
plt.subplot(2, 1, 1)
plt.plot(S_[:, 0], color='red')
plt.title('ICA 1')
plt.subplot(2, 1, 2)
plt.plot(S_[:, 1], color='steelblue')
plt.title('ICA 2')
plt.tight_layout()
plt.show()
```



Sin embargo, lo que obtuve mediante el filtro espacial, es volver a separarme ambas series de forma similar a las originales.

Signal features

Adapto el scrip de Signal Features, para obtener distintas métricas sobre las series, entre ellas las medidas de Raíz Cuadrada Media (RMS), Peak-To-Peak, Mobidity y Complexity de las series. Obtengo los siguientes resultados.

In [ ]:
```
# Signal Features

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

from scipy.fftpack import fft
```

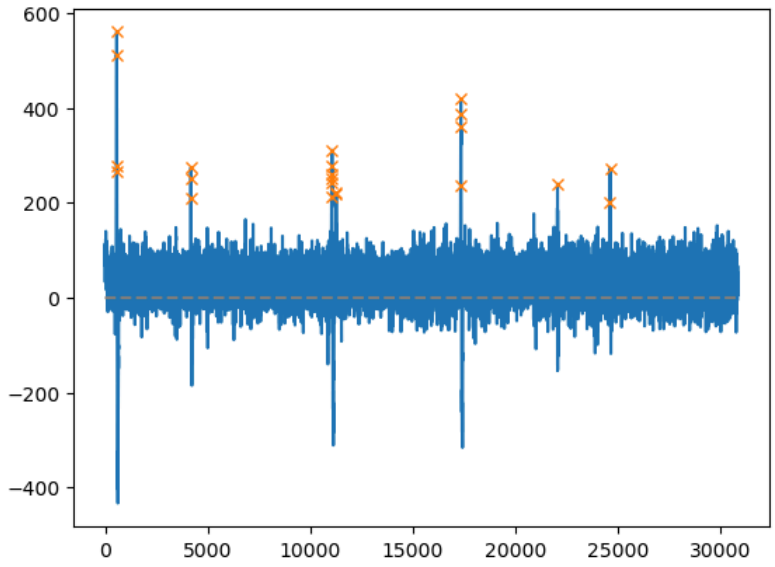
```
import math

from scipy.signal import firwin, remez, kaiser_atten, kaiser_beta
from scipy.signal import butter, filtfilt, buttord

from scipy.signal import butter, lfilter

import matplotlib.pyplot as plt

def butter_bandpass(lowcut, highcut, fs, order=5):
    nyq = 0.5 * fs
    low = lowcut / nyq
    high = highcut / nyq
    b, a = butter(order, [low, high], btype='band')
    return b, a


def butter_bandpass_filter(data, lowcut, highcut, fs, order=5):
    b, a = butter_bandpass(lowcut, highcut, fs, order=order)
    y = lfilter(b, a, data)
    return y


def psd(y):
    # Number of samplepoints
    N = 512
    # sample spacing
    T = 1.0 / 512.0
    # From 0 to N, N*T, 2 points.
    #x = np.linspace(0.0, 1.0, N)
    #y = 1*np.sin(10.0 * 2.0*np.pi*x) + 9*np.sin(20.0 * 2.0*np.pi*x)


    # Original Bandpass
    fs = 512.0
    fso2 = fs/2
    #Nd,wn = buttord(wp=[9/fso2,11/fso2], ws=[8/fso2,12/fso2],
    #   gpass=3.0, gstop=40.0)
    #b,a = butter(Nd,wn,'band')
    #y = filtfilt(b,a,y)

    y = butter_bandpass_filter(y, 8.0, 15.0, fs, order=6)


    yf = fft(y)
    #xf = np.linspace(0.0, int(1.0/(2.0*T)), int(N/2))
    #import matplotlib.pyplot as plt
    #plt.plot(xf, 2.0/N * np.abs(yf[0:int(N/2)]))
    #plt.axis((0,60,0,1))
    #plt.grid()
    #plt.show()

    return np.sum(np.abs(yf[0:int(N/2)]))

def crest_factor(x):
    return np.max(np.abs(x))/np.sqrt(np.mean(np.square(x)))

def hjorth(a):
    r"""
    Compute Hjorth parameters [HJO70]_.
    .. math::
        Activity = m_0 = \sigma_{a}^2
    .. math::
        Complexity = m_2 = \sigma_{d}/ \sigma_{a}
    .. math::
        Morbidity = m_4 =  \frac{\sigma_{dd}/ \sigma_{d}}{m_2}
    Where:
    :math:`\sigma_{x}^2` is the mean power of a signal :math:`x`. That is, its variance, if it's mean is zero.
    :math:`a`, :math:`d` and :math:`dd` represent the original signal, its first and second derivatives, respectively.
    .. note::
        **Difference with PyEEG:**
        Results is different from [PYEEG]_ which appear to uses a non normalised (by the length of the signal) definition
        .. math::
            \sigma_{a}^2 = \sum{\mathbf{x}[i]^2}
        As opposed to
        .. math::
            \sigma_{a}^2 = \frac{1}{n}\sum{\mathbf{x}[i]^2}
    :param a: a one dimensional floating-point array representing a time series.
    :type a: :class:`~numpy.ndarray` or :class:`~pyrem.time_series.Signal`
    :return: activity, complexity and morbidity
    :rtype: tuple(float, float, float)
    Example:
    >>> import pyrem as pr
    >>> import numpy as np
    >>> # generate white noise:
```

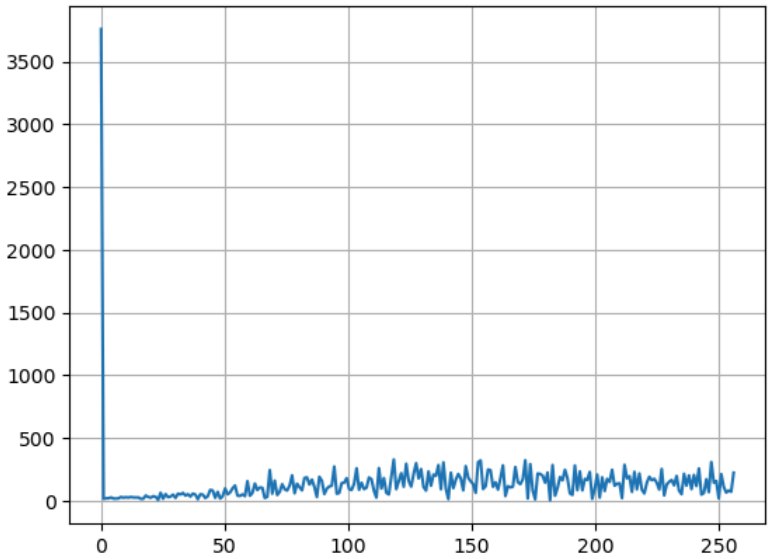
```
    >>> noise = np.random.normal(size=int(1e4))
    >>> activity, complexity, morbidity = pr.univariate.hjorth(noise)
    """

    first_deriv = np.diff(a)
    second_deriv = np.diff(a,2)

    var_zero = np.mean(a ** 2)
    var_d1 = np.mean(first_deriv ** 2)
    var_d2 = np.mean(second_deriv ** 2)

    activity = var_zero
    morbidity = np.sqrt(var_d1 / var_zero)
    complexity = np.sqrt(var_d2 / var_d1) / morbidity

    return activity, morbidity, complexity

def pfd(a):
    r"""
    Compute Petrosian Fractal Dimension of a time series [PET95]_.
    It is defined by:
    .. math::
        \frac{log(N)}{log(N) + log(\frac{N}{N+0.4N_{\delta}})}
    .. note::
        **Difference with PyEEG:**
        Results is different from [PYEEG]_ which implemented an apparently erroneous formulae:
        .. math::
            \frac{log(N)}{log(N) + log(\frac{N}{N}+0.4N_{\delta})}
    Where:
    :math:`N` is the length of the time series, and
    :math:`N_{\delta}` is the number of sign changes.
    :param a: a one dimensional floating-point array representing a time series.
    :type a: :class:`~numpy.ndarray` or :class:`~pyrem.time_series.Signal`
    :return: the Petrosian Fractal Dimension; a scalar.
    :rtype: float
    Example:
    >>> import pyrem as pr
    >>> import numpy as np
    >>> # generate white noise:
    >>> noise = np.random.normal(size=int(1e4))
    >>> pr.univariate.pdf(noise)
    """

    diff = np.diff(a)
    # x[i] * x[i-1] for i in t0 -> tmax
    prod = diff[1:-1] * diff[0:-2]

    # Number of sign changes in derivative of the signal
    N_delta = np.sum(prod < 0)
    n = len(a)

    return np.log(n)/(np.log(n)+np.log(n/(n+0.4*N_delta)))



# Sampling frequency of 512 Hz


# Baseline

print('------------ Baseline ------------')
baseline = pd.read_csv('data/baseline.dat', delimiter=' ', names = ['timestamp','counter','eeg','attention','meditation','
baseline = baseline.values
baseline_eeg = baseline[:,2]

ptp = abs(np.max(baseline_eeg)) + abs(np.min(baseline_eeg))
rms = np.sqrt(np.mean(baseline_eeg**2))
cf = crest_factor(baseline_eeg)

print ('Peak-To-Peak:' + str(ptp))
print ('Root Mean Square:' + str(rms))
print ('Crest Factor:' + str(cf))

from collections import Counter
from scipy import stats

entropy = stats.entropy(list(Counter(baseline_eeg).values()), base=2)

print('Shannon Entropy:' + str(entropy))


activity, complexity, morbidity = hjorth(baseline_eeg)

print('Activity:' + str(activity))
print('Complexity:' + str(complexity))
print('Mobidity:' + str(morbidity))
```

```
fractal = pfd(baseline_eeg)
print('Fractal:' + str(fractal))

import matplotlib.pyplot as plt
from scipy.signal import find_peaks

peaks, _ = find_peaks(baseline_eeg, height=200)
plt.plot(baseline_eeg)
plt.plot(peaks, baseline_eeg[peaks], "x")
plt.plot(np.zeros_like(baseline_eeg), "--", color="gray")
plt.show()


N = 512
T = 1.0 / 512.0

# We can put an additional frequency component to verify that things are working ok
shamsignal = False
if (shamsignal):
    x= np.linspace(0.0, 1.0, N)
    baseline_eeg = baseline_eeg[:512] +  100*np.sin(10.0 * 2.0*np.pi*x)


yf = fft(baseline_eeg)
xf = np.linspace(0.0, int(1.0/(2.0*T)), int(N/2))

plt.close()

plt.plot(xf, 2.0/N * np.abs(yf[0:int(N/2)]))
plt.grid()
plt.show()

print('PSD:' + str(psd(baseline_eeg[:512])))




# Pestañeos
print('------------ Pestañeos ------------')
pestaneos = pd.read_csv('data/pestaneos.dat', delimiter=' ', names = ['timestamp','counter','eeg','attention','meditation'
pestaneos = pestaneos.values
pestaneos_eeg = pestaneos[:,2]

ptp = abs(np.max(pestaneos_eeg)) + abs(np.min(pestaneos_eeg))
rms = np.sqrt(np.mean(pestaneos_eeg**2))
cf = crest_factor(pestaneos_eeg)

print ('Peak-To-Peak:' + str(ptp))
print ('Root Mean Square:' + str(rms))
print ('Crest Factor:' + str(cf))

from collections import Counter
from scipy import stats

entropy = stats.entropy(list(Counter(pestaneos_eeg).values()), base=2)

print('Shannon Entropy:' + str(entropy))


activity, complexity, morbidity = hjorth(pestaneos_eeg)

print('Activity:' + str(activity))
print('Complexity:' + str(complexity))
print('Mobidity:' + str(morbidity))



fractal = pfd(pestaneos_eeg)
print('Fractal:' + str(fractal))

import matplotlib.pyplot as plt
from scipy.signal import find_peaks

peaks, _ = find_peaks(pestaneos_eeg, height=200)
plt.plot(pestaneos_eeg)
plt.plot(peaks, pestaneos_eeg[peaks], "x")
plt.plot(np.zeros_like(pestaneos_eeg), "--", color="gray")
plt.show()


N = 512
T = 1.0 / 512.0
```

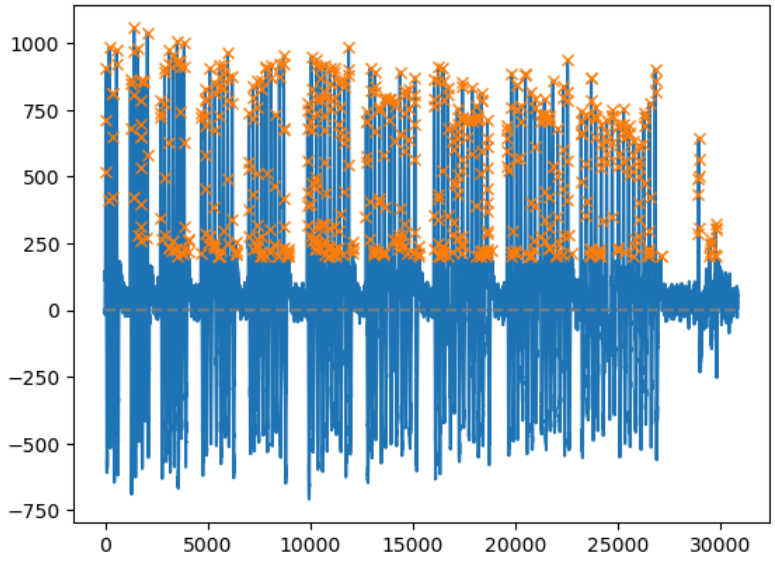
```
# We can put an additional frequency component to verify that things are working ok
shamsignal = False
if (shamsignal):
    x= np.linspace(0.0, 1.0, N)
    pestaneos_eeg = pestaneos_eeg[:512] +  100*np.sin(10.0 * 2.0*np.pi*x)


yf = fft(pestaneos_eeg)
xf = np.linspace(0.0, int(1.0/(2.0*T)), int(N/2))

plt.close()

plt.plot(xf, 2.0/N * np.abs(yf[0:int(N/2)]))
plt.grid()
plt.show()

print('PSD:' + str(psd(pestaneos_eeg[:512])))
```

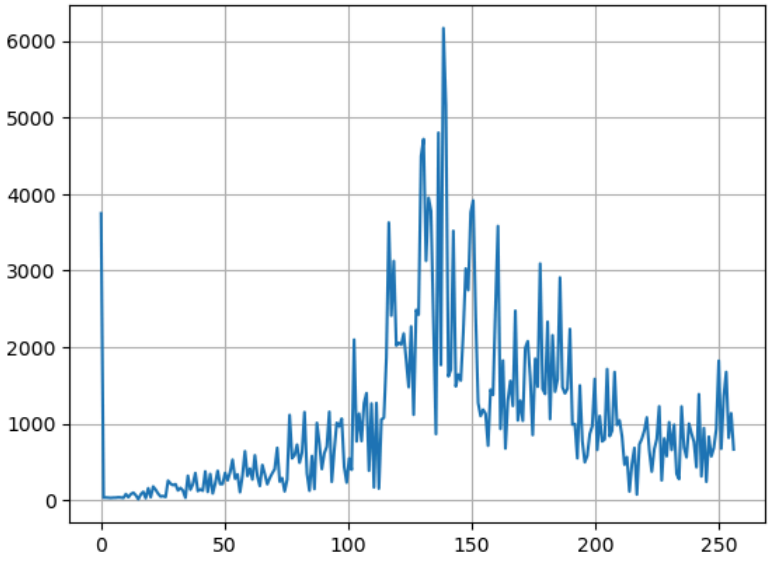
```
------------ Baseline ------------
Peak-To-Peak:993.0
Root Mean Square:55.1968826193238
Crest Factor:10.145500496144875
Shannon Entropy:7.077667431154785
Activity:3046.69585089141
Complexity:0.3300162016062754
Mobidity:3.1062967860621895
Fractal:1.0115239098741198
```






```
PSD:12439.402872819663
------------ Pestañeos ------------
Peak-To-Peak:1765.0
Root Mean Square:302.391681800418
Crest Factor:3.492159551852264
Shannon Entropy:9.695468701567941
Activity:91440.72922208525
Complexity:0.06949341771807699
Mobidity:11.650633910732411
Fractal:1.0093284110411525
```






```
PSD:81060.04405964719
```

A partir de los resultados de las métricas encuentro que la serie de pestañeos tiene una mayor amplitud (Peak to peak) y variabilidad (Mobidity), una mayor potencia promedio (RMS mayor), un cress factor mayor, es decir, la relacion entre el RMS y el valor maximo de la señal, mayor actividad (Activity) y entropia(Shannon Entropy), pero sin embargo, un menor nivel de complejidad.

Estadísticas descriptivas

Luego, realizo una simple descripción de las series utilizando estadísticas descriptivas.

In [ ]:
```
# Baseline

print('\n------------ Baseline ------------')
baseline = pd.read_csv('data/baseline.dat', delimiter=' ', names = ['timestamp','counter','eeg','attention','meditation','
baseline = baseline.values
baseline_eeg = baseline[:,2]

print("Length: {}".format(len(baseline_eeg)))
print("Max value: {}".format(baseline_eeg.max()))
print("Min value: {}".format(baseline_eeg.min()))
print("Range: {}".format(baseline_eeg.max()-baseline_eeg.min()))
print("Average value: {}".format(baseline_eeg.mean()))
print("Variance: {}".format(baseline_eeg.var()))
print("Std: {}".format(math.sqrt(baseline_eeg.var())))


# Pestañeos
pestaneos = pd.read_csv('data/pestaneos.dat', delimiter=' ', names = ['timestamp','counter','eeg','attention','meditation'
pestaneos = pestaneos.values
pestaneos_eeg = pestaneos[:,2]


print('\n------------ Pestañeos ------------')
print("Length: {}".format(len(pestaneos_eeg)))
print("Max value: {}".format(pestaneos_eeg.max()))
```

```
print("Min value: {}".format(pestaneos_eeg.min()))
print("Range: {}".format(pestaneos_eeg.max()-pestaneos_eeg.min()))
print("Average value: {}".format(pestaneos_eeg.mean()))
print("Variance: {}".format(pestaneos_eeg.var()))
print("Std: {}".format(math.sqrt(pestaneos_eeg.var())))
```

```
------------ Baseline ------------
Length: 30850
Max value: 560.0
Min value: -433.0
Range: 993.0
Average value: 31.187649918962723
Variance: 2074.0263434236344
Std: 45.54147937236596

------------ Pestañeos ------------
Length: 30826
Max value: 1056.0
Min value: -709.0
Range: 1765.0
Average value: 31.113962239667813
Variance: 90472.65057583377
Std: 300.7867194139957
```

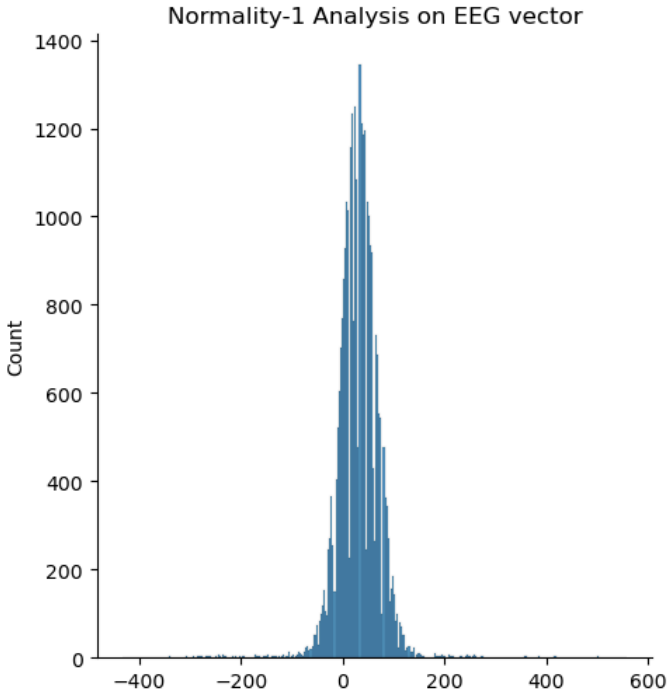
De forma similar a lo antes obtenido, si bien comparte la misma media, la varianza y las medidas de variabilidad tales como rango, o desviacion standar son mayores en la serie de pestañeos.

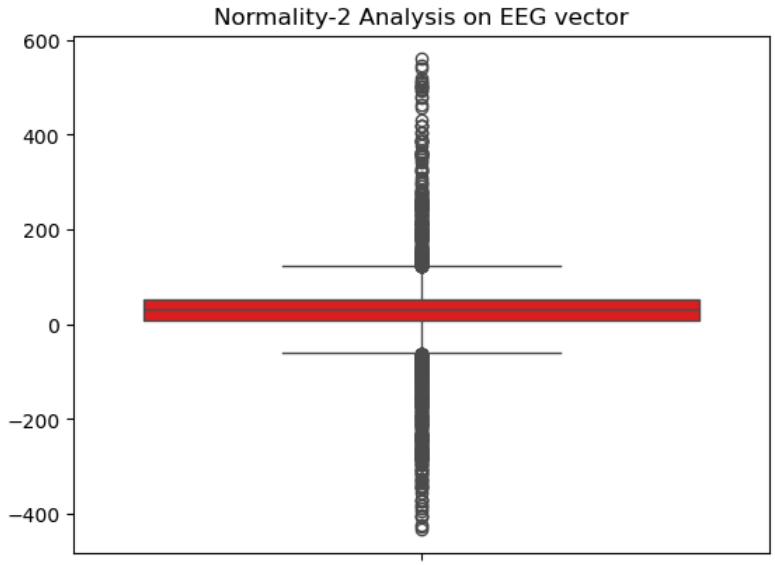
Pruebas de normalidad

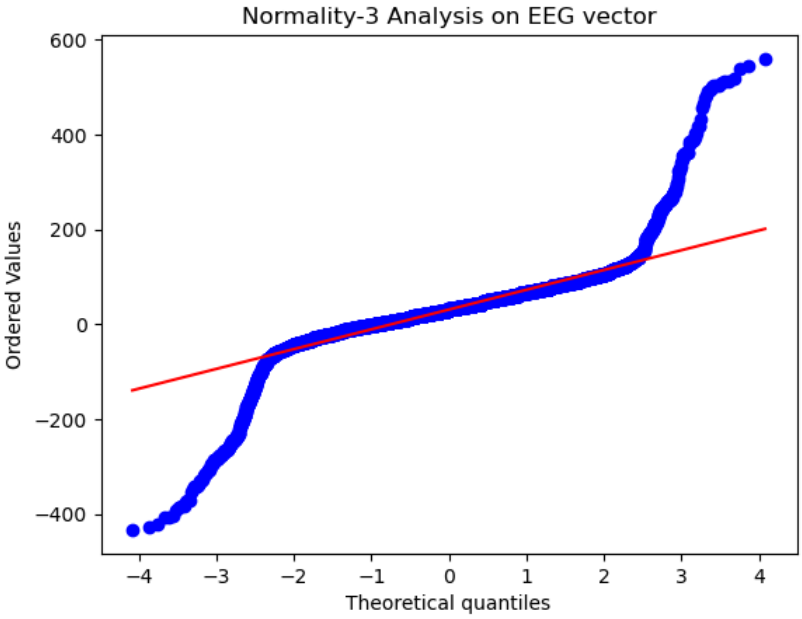
Finalmente, llevo adelante unas pruebas de normalidad de la serie y obtengo los siguientes resultados.

In [ ]:
```
# Prueba de normalidad
import csv
import numpy as np
import seaborn as sns
import math
import scipy
from scipy import stats
import matplotlib.pyplot as plt

# Baseline

print('\n------------ Baseline ------------')
print('normality = {}'.format(scipy.stats.normaltest(baseline_eeg)))
sns.displot(baseline_eeg)
plt.title("Normality-1 Analysis on EEG vector")
plt.show()
sns.boxplot(baseline_eeg,color="red")
plt.title("Normality-2 Analysis on EEG vector")
plt.show()
res = stats.probplot(baseline_eeg, plot = plt)
plt.title("Normality-3 Analysis on EEG vector")
plt.show()

# Pestañeos
print('\n------------ Pestañeos ------------')
print('normality = {}'.format(scipy.stats.normaltest(pestaneos_eeg)))
sns.displot(pestaneos_eeg)
plt.title("Normality-1 Analysis on EEG vector")
plt.show()
sns.boxplot(pestaneos_eeg,color="red")
plt.title("Normality-2 Analysis on EEG vector")
plt.show()
res = stats.probplot(pestaneos_eeg, plot = plt)
plt.title("Normality-3 Analysis on EEG vector")
plt.show()
```

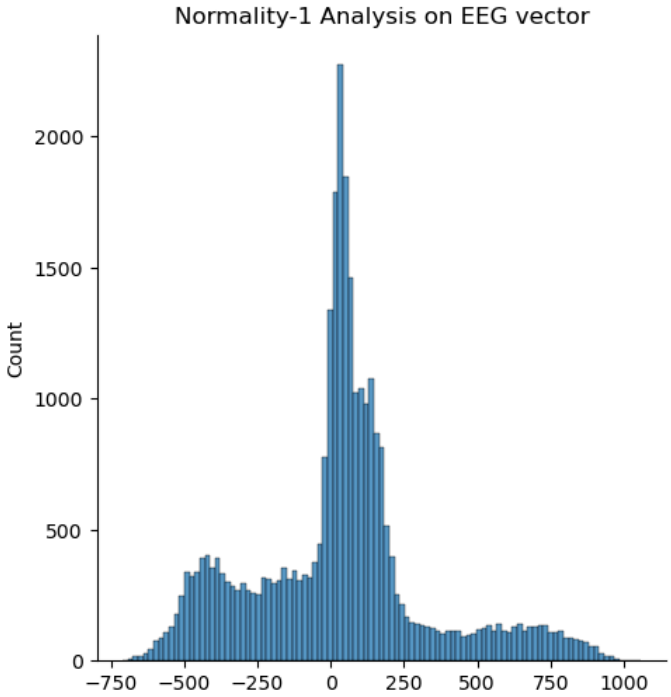
```
------------ Baseline ------------
normality = NormaltestResult(statistic=7707.945455095068, pvalue=0.0)
```

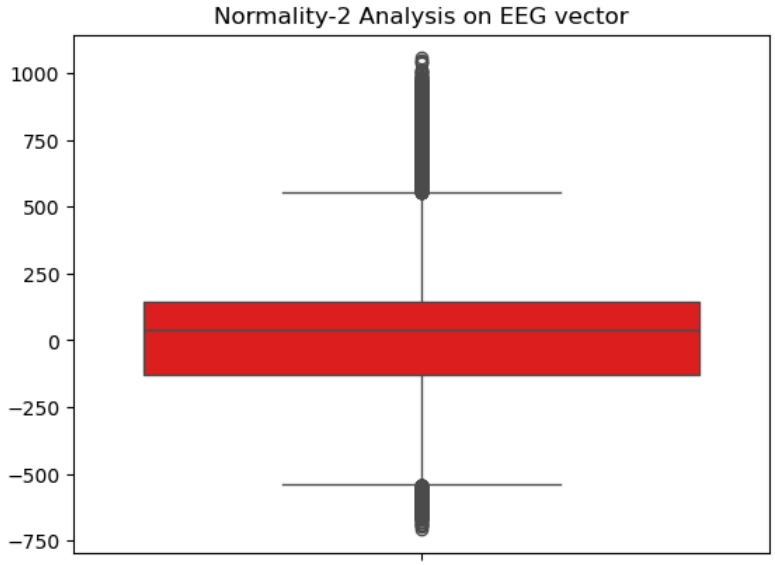


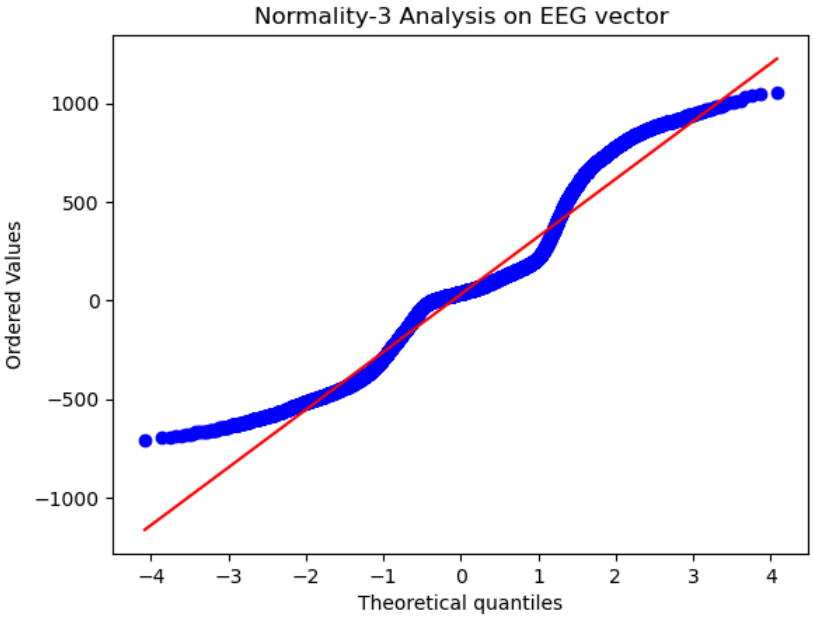





```
------------ Pestañeos ------------
normality = NormaltestResult(statistic=1270.8017475645895, pvalue=1.1191974702668965e-276)
```







Como evidencian los test, la serie baseline pasa el test de normalidad, pero lo hace la serie de pestañeos.

Clasificador de pestañeos

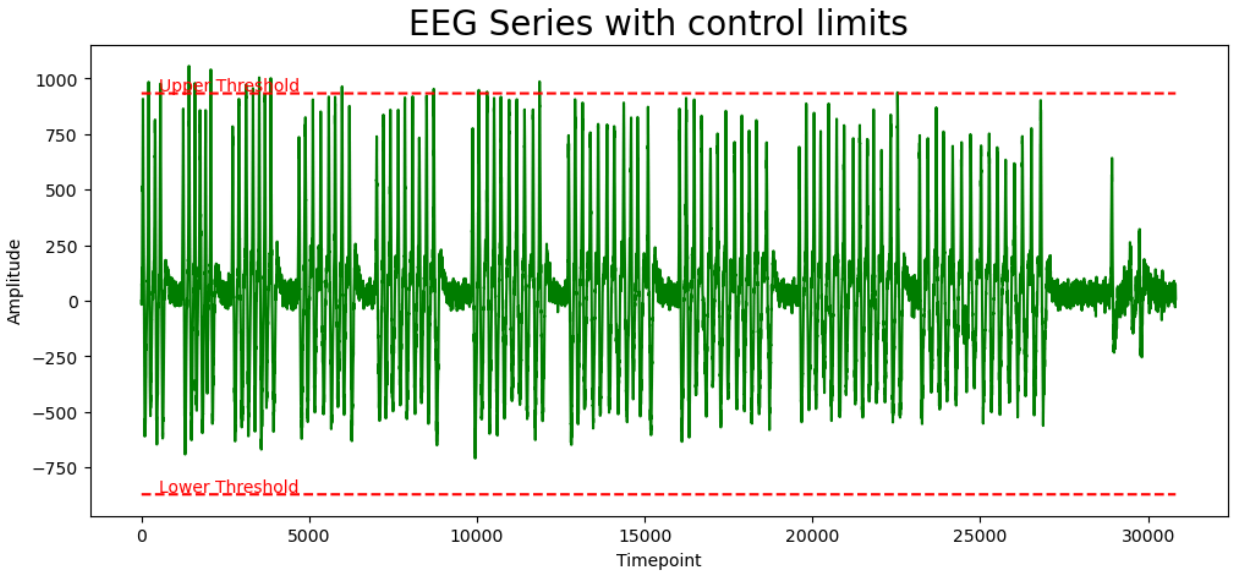
Para obtener un clasificador de pestañeos, directamente aplico la metologia implementada en el scrip eventcounter.py. En dicha metologia, el autor utiliza la estrategia de umbrales para detectar los pestañeos. A continuación muestro los resultados obtenidos en cada una de la estrategias aplicadas por este autor implementadas y adaptadas al dataset del ejercicio.

Contador de pestañeos mediante Umbral como desviaciones estandar del dataset de pestañeos

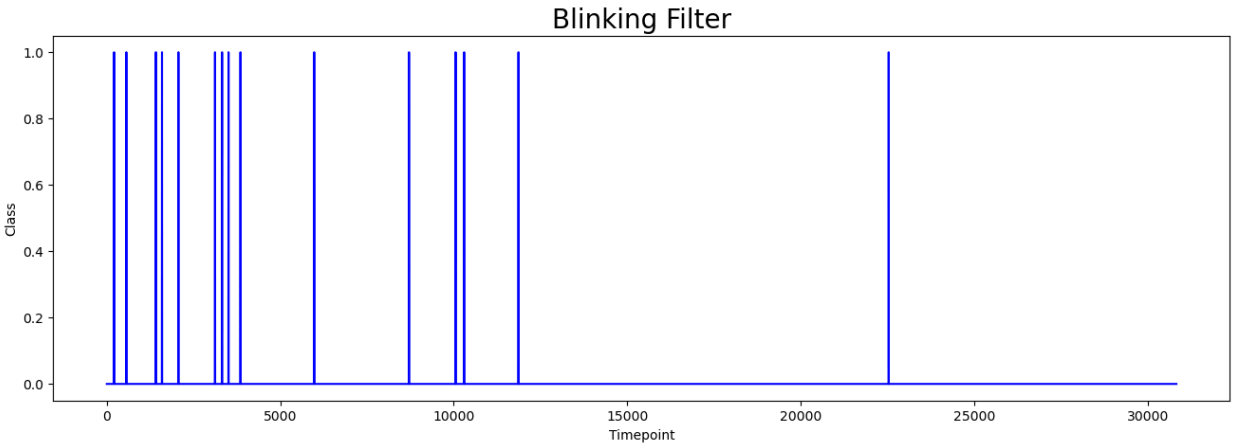
En esta estrategia, calculo los valores atipicos, a mas de 3 desvios standar de la media, y los cuento como pestañeos. Para hacerlo, el umbral es calculado directamente sobre el dataset de pestañeos.

In [ ]:
```
umbral_superior=int(pestaneos_eeg.mean()+3*pestaneos_eeg.std())
print("Upper Threshold: {}".format(umbral_superior))
umbral_inferior=int(pestaneos_eeg.mean()-3*pestaneos_eeg.std())
print("Lower Threshold: {}".format(umbral_inferior))
plt.figure(figsize=(12,5))
plt.plot(pestaneos_eeg,color="green")
plt.plot(np.full(len(pestaneos_eeg),umbral_superior),'r--')
plt.plot(np.full(len(pestaneos_eeg),umbral_inferior),'r--')
plt.ylabel("Amplitude",size=10)
plt.xlabel("Timepoint",size=10)
plt.title("EEG Series with control limits",size=20)
plt.annotate("Upper Threshold",xy=(500,umbral_superior+10),color="red")
plt.annotate("Lower Threshold",xy=(500,umbral_inferior+10),color="red")
plt.show()


filtro_eeg=[]
contador=0
for i in range(len(pestaneos_eeg)):
    if i==0:
        filtro_eeg.append(0)
    elif pestaneos_eeg[i]>umbral_superior:
        filtro_eeg.append(1)
        if pestaneos_eeg[i-1]<=umbral_superior:
            #print(i)
            contador=contador+1
    elif pestaneos_eeg[i]<umbral_inferior:
        filtro_eeg.append(-1)
    else:
        filtro_eeg.append(0)

print("Blinking counter: {}".format(contador))
filtro_eeg=np.asarray(filtro_eeg)
plt.figure(figsize=(16,5))
plt.plot(filtro_eeg,color="blue")
plt.title("Blinking Filter",size=20)
plt.ylabel("Class",size=10)
plt.xlabel("Timepoint",size=10)
plt.show()
```

```
Upper Threshold: 933
Lower Threshold: -871
```




```
Blinking counter: 15
```



Los resultados parecen razonables, un total de 15 pestañeos.

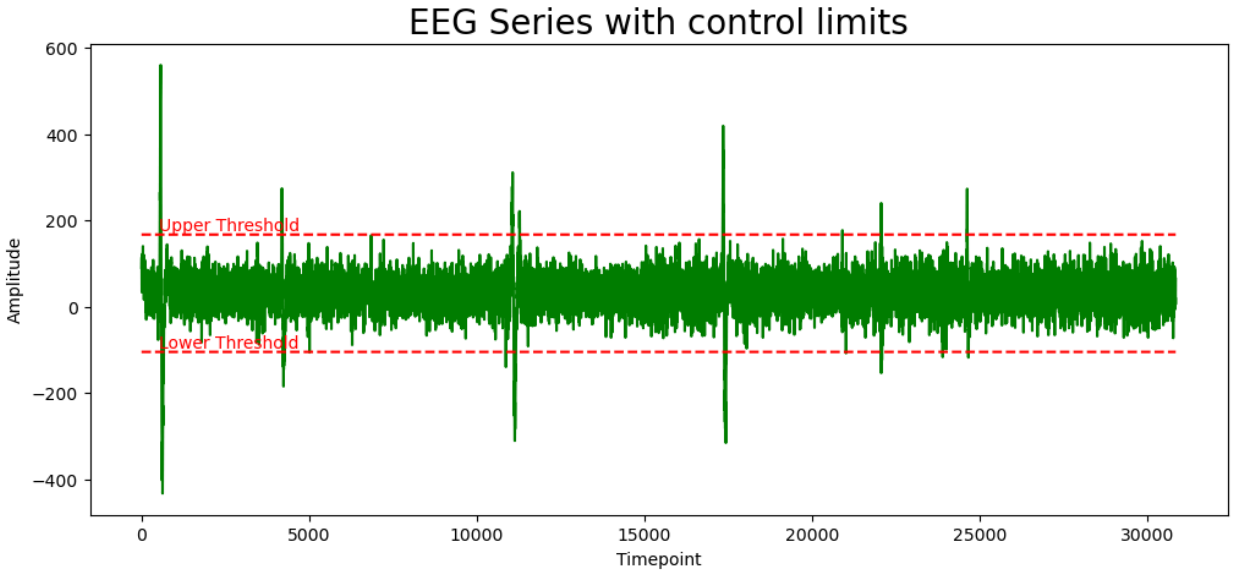
Contador de pestañeos mediante Umbral como desviaciones estandar del dataset de baseline

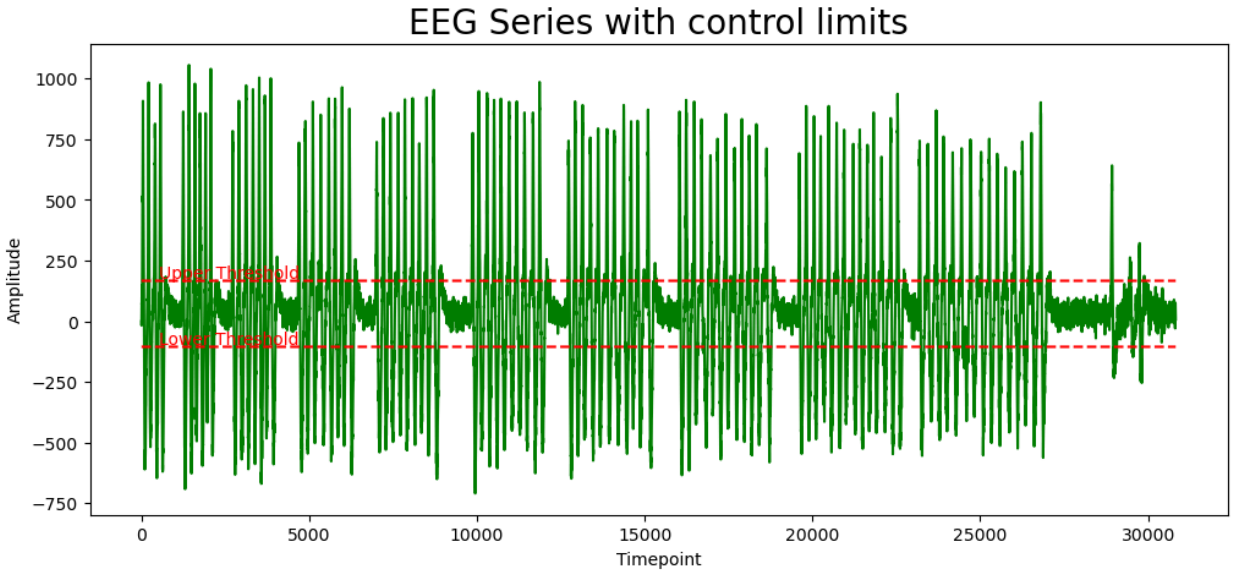
Si bien parecen razonables, los umbrales parecen estar muy encima la mayor parte de la serie. Por eso, para comparalo realizo el mismo ejercicio, pero creando los umbrales en base al baseline.

In [ ]:
```
umbral_superior=int(baseline_eeg.mean()+3*baseline_eeg.std())
print("Upper Threshold: {}".format(umbral_superior))
umbral_inferior=int(baseline_eeg.mean()-3*baseline_eeg.std())
print("Lower Threshold: {}".format(umbral_inferior))
plt.figure(figsize=(12,5))
plt.plot(baseline_eeg,color="green")
plt.plot(np.full(len(baseline_eeg),umbral_superior),'r--')
plt.plot(np.full(len(baseline_eeg),umbral_inferior),'r--')
plt.ylabel("Amplitude",size=10)
plt.xlabel("Timepoint",size=10)
plt.title("EEG Series with control limits",size=20)
plt.annotate("Upper Threshold",xy=(500,umbral_superior+10),color="red")
plt.annotate("Lower Threshold",xy=(500,umbral_inferior+10),color="red")
plt.show()


# Testeo si el umbral funciona para la prueba de pestañeos

#umbral_superior=int(pestaneos_eeg.mean()+3*pestaneos_eeg.std())
#print("Upper Threshold: {}".format(umbral_superior))
#umbral_inferior=int(pestaneos_eeg.mean()-3*pestaneos_eeg.std())
#print("Lower Threshold: {}".format(umbral_inferior))
plt.figure(figsize=(12,5))
plt.plot(pestaneos_eeg,color="green")
plt.plot(np.full(len(pestaneos_eeg),umbral_superior),'r--')
plt.plot(np.full(len(pestaneos_eeg),umbral_inferior),'r--')
plt.ylabel("Amplitude",size=10)
plt.xlabel("Timepoint",size=10)
plt.title("EEG Series with control limits",size=20)
plt.annotate("Upper Threshold",xy=(500,umbral_superior+10),color="red")
plt.annotate("Lower Threshold",xy=(500,umbral_inferior+10),color="red")
plt.show()



filtro_eeg=[]
contador=0
for i in range(len(pestaneos_eeg)):
    if i==0:
        filtro_eeg.append(0)
    elif pestaneos_eeg[i]>umbral_superior:
        filtro_eeg.append(1)
        if pestaneos_eeg[i-1]<=umbral_superior:
           #print(i)
            contador=contador+1
    elif pestaneos_eeg[i]<umbral_inferior:
        filtro_eeg.append(-1)
    else:
        filtro_eeg.append(0)

print("Blinking counter: {}".format(contador))
filtro_eeg=np.asarray(filtro_eeg)
plt.figure(figsize=(16,5))
plt.plot(filtro_eeg,color="blue")
plt.title("Blinking Filter",size=20)
plt.ylabel("Class",size=10)
```

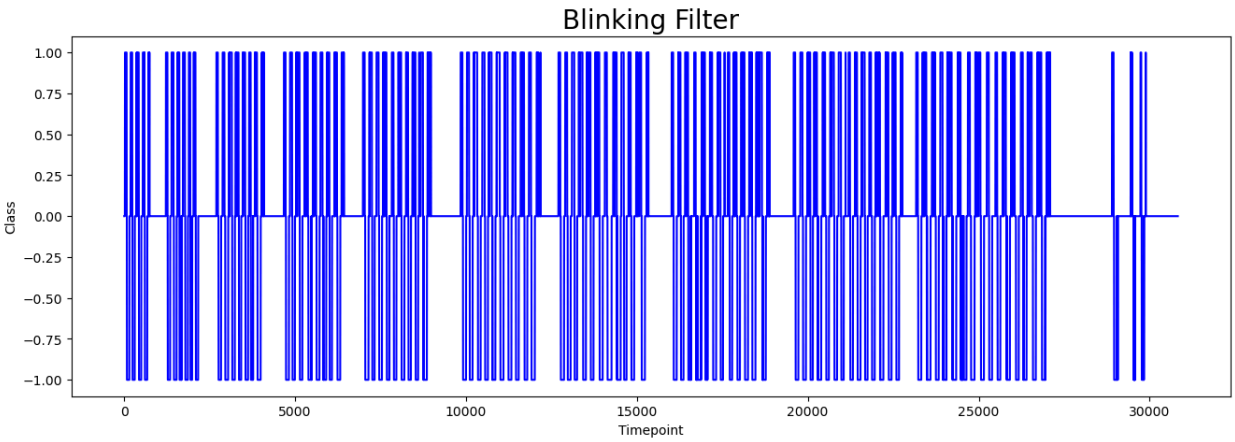
```
plt.xlabel("Timepoint",size=10)
plt.show()
```

```
Upper Threshold: 167
Lower Threshold: -105
```






```
Blinking counter: 289
```



Nos arroja un total de 289 pestañeos, lo cual parece estar un poco desfazado de la realidad.

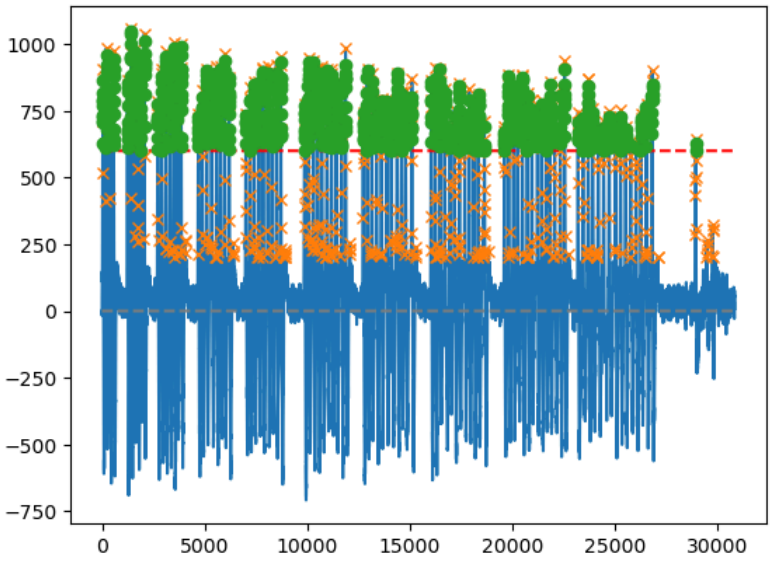
Umbral fijado visualmente

Como tercera alternativa, utilizo la opción presentada por el autor del script eventcounter.py, que fija un umbral visualmente. Personalmente lo fije en 600. Los resultados obtenidos son los siguientes.

In [ ]:
```
# Pruebo el metodo alternativo que propone el creador del script Eventcounter.py, solo cambio el umbral de 420 a 600
# Alternative method
```

```
# The threshold is hardcoded, visually estimated.
signalthreshold = 600

eeg=pestaneos_eeg
# Filter the values above the threshold
boolpeaks = np.where( eeg > signalthreshold  )
print (boolpeaks)

# Pick the derivative
dpeaks = np.diff( eeg )
print (dpeaks)

# Identify those values where the derivative is ok
pdpeaks = np.where( dpeaks > 0)

peaksd = pdpeaks[0]

# boolpeaks and peaksd are indexes.
finalresult = np.in1d(peaksd,boolpeaks)

print (finalresult)
blinkings = finalresult.sum()

peaks1 = peaksd[finalresult]

print ('Blinkings: %d' % blinkings)
#print ('Locations:');print(peaks1)

import matplotlib.pyplot as plt
from scipy.signal import find_peaks

peaks2, _ = find_peaks(eeg, height=200)
plt.plot(eeg)
plt.plot(np.full(len(eeg),signalthreshold),'r--')
plt.plot(peaks2, eeg[peaks2], "x")
plt.plot(peaks1, eeg[peaks1], "o")
plt.plot(np.zeros_like(eeg), "--", color="gray")
plt.show()
```

```
(array([   33,    34,    35, ..., 28925, 28926, 28927], dtype=int64),)
[-17.  26.  43. ... -18. -18. -12.]
[False False False ... False False False]
Blinkings: 865
```



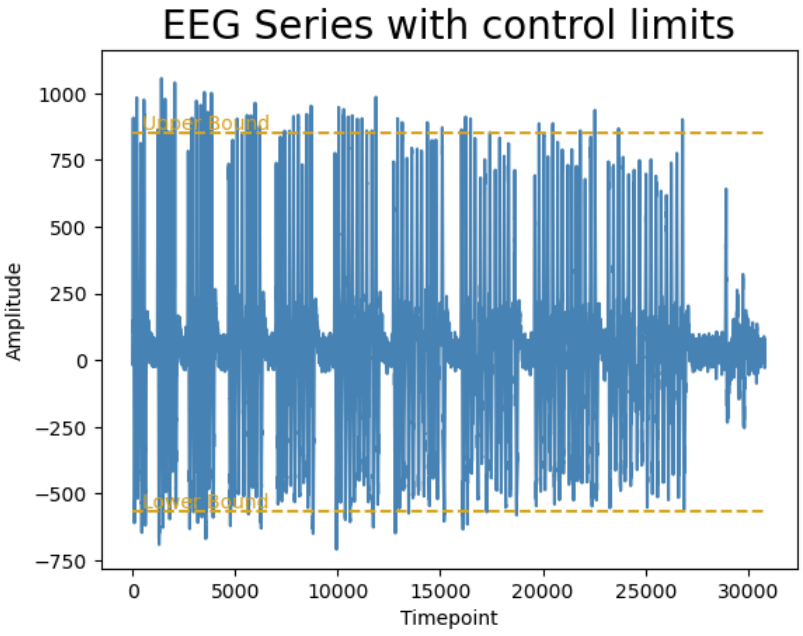
Con esta metologia obtengo un total de 865 pestañeos, lo que suena aun mas desfazado, aun cuando el metodo funciono correctamente. Seria una cuestión de ajustar el umbral correcto para la muestra.

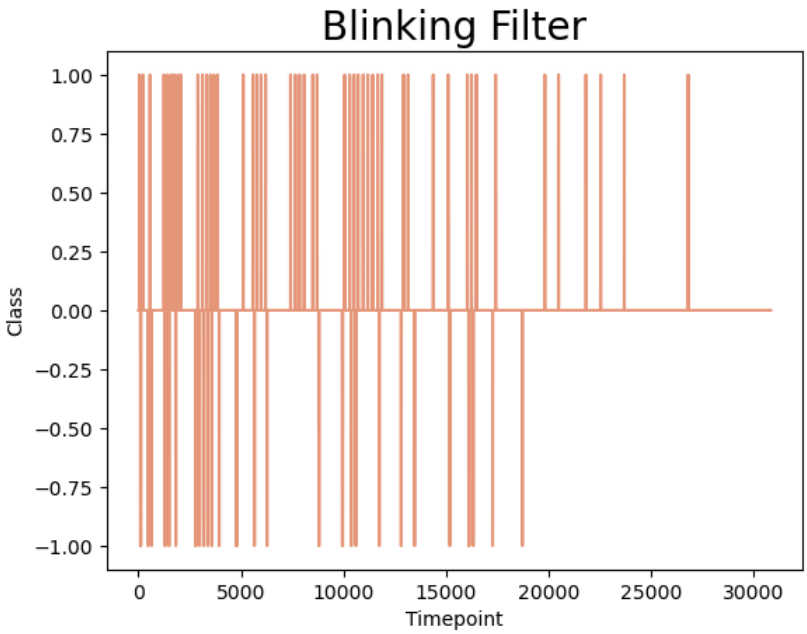
Umbrales fijados por percentiles

Continuo implementando el script eventcounter para implementar su tecnica de fijar umbrales utilizando los percentiles de la distribucion. A continuación expongo los resultados obtenidos:

In [ ]:
```
# Tomo lo realizado en el scrip del Alumno: Francisco Seguí https://github.com/fseguior/:
# "En este caso uso el percentil 1 y el 99, con lo cual se consideran como picos el 2% de los valores
# De esta forma el filtro es dinámico, y se adapta a los valores de la muestra.
# Vemos en el gráfico que el criterio funciona adecuadamente

lowerbound=int(np.percentile(eeg, 1))
upperbound=int(np.percentile(eeg, 99))
```

```
plt.plot(eeg, color="steelblue")
plt.plot(np.full(len(eeg),lowerbound), color="goldenrod", ls="--")
plt.plot(np.full(len(eeg),upperbound), color="goldenrod", ls="--")
plt.ylabel("Amplitude",size=10)
plt.xlabel("Timepoint",size=10)
plt.title("EEG Series with control limits",size=20)
plt.ylim([min(eeg)*1.1, max(eeg)*1.1 ])  ## dinamizo los valores del eje así se adapta a los datos que proceso
plt.annotate("Lower Bound",xy=(500,lowerbound+10),color="goldenrod")
plt.annotate("Upper Bound",xy=(500,upperbound+10),color="goldenrod")
plt.savefig('blinks.png')
plt.show()

# Grafico el filtro de pestañeos/blinking
# Utilizo una función lambda para marcar los pestañeos

blinks = list((map(lambda x: 1 if x >upperbound else ( -1 if x < lowerbound else 0), eeg)))
blinks = np.asarray(blinks)

plt.plot(blinks, color="darksalmon")
plt.title("Blinking Filter",size=20)
plt.ylabel("Class",size=10)
plt.xlabel("Timepoint",size=10)
plt.savefig('blinkingfilter.png')
plt.show()

# Encuentro picos positivos. Filtro los valores donde blink==1, y luego analizo que haya habido un salto realmente (para n
# Con un map y una funcion lambda obtengo una lista con booleanos para los valores donde hay picos realmente.
# Luego los filtro con una función filter y otra lambda
peak=np.where(blinks == 1)[0]

peakdiff=np.diff(np.append(0,peak))

boolpeak=list(map(lambda x : x > 100, peakdiff))

peakslocation=list(filter(lambda x: x, boolpeak*peak))

# Repito para los valles, mismo algoritmo pero busco blinks == -1
valley=np.where(blinks == -1)[0]

valleydiff=np.diff(np.append(0,valley))

boolvalley=list(map(lambda x : x > 100, valleydiff))

valleylocation=list(filter(lambda x: x, boolvalley*valley))

# Hago un append de los valles y los picos, y los ordeno. Luego los cuento para imprimir tanto la cantidad de pestañeos, c

blinklocations=np.sort(np.append(peakslocation,valleylocation))

blinkcount=np.count_nonzero(blinklocations)

print(f'Count of Blinks: {blinkcount}')
print('Location of Blinks');print(blinklocations)
```






```
Count of Blinks: 74
Location of Blinks
[  207   450   557   625  1242  1288  1409  1483  1586  1741  1814  1909
  2058  2783  2898  2986  3111  3182  3318  3378  3502  3564  3674  3848
  3934  4766  5104  5578  5654  5769  5971  6194  6261  7416  7650  7845
  8075  8494  8702  8788  9932 10048 10296 10370 10504 10592 10710 10956
 11176 11413 11667 11735 11859 12799 12919 13146 13462 14371 15096 15172
 16034 16101 16232 16322 16471 17267 17415 18717 19811 20477 21821 22532
 23686 26801]
```

Finalmente, este metodo arroja un resultado de 74 pestañeos en la serie.