```
import numpy as np
from scipy import signal
import matplotlib.pyplot as plt
from scipy.signal import butter, lfilter

#--------------------------------------------------------------------------------------------------
# Preprocessing
#--------------------------------------------------------------------------------------------------
def preprocess_signal(x, plot, filter_method, **kwargs):
    assert x.ndim == 1

    filtered = filter_method(x, **kwargs)

    if plot:
        plt.plot(x, color="gainsboro", label= "Raw")
        plt.plot(filtered, color="royalblue", label= "Post filter")

    return filtered

def preprocess_bvp(x, sampling_rate, plot= False):
    """Low-pass filter for continuous BP signal preprocessing, adaopted from Nabian et al. (2018).
    """
    return preprocess_signal(x, plot, filter_method= butter_filter, order = 2, cutoff_freq= 40., fs= sampling_rate, btype= "lowpass")

def preprocess_ecg(x, sampling_rate, plot= False):
    """From https://github.com/berndporr/py-ecg-detectors/
  - C. Zeelenberg, A single scan algorithm for QRS detection and feature extraction, IEEE Comp.
    in Cardiology, vol. 6, pp. 37-42, 1979.
```

```
    - A. Lourenco, H. Silva, P. Leite, R. Lourenco and A. Fred, "Real Time Electrocardiogram Segmentation
      for Finger Based ECG Biometrics", BIOSIGNALS 2012, pp. 49-54, 2012.
    """
        f1 = 48 / (0.5 * sampling_rate)
        f2 = 52 / (0.5 * sampling_rate)

        sos = signal.butter(4, [f1, f2], btype="bandstop", output="sos")
        zi_coeff = signal.sosfilt_zi(sos)
        zi = zi_coeff * np.mean(x)

        filtered = signal.sosfilt(sos, x, zi=zi)[0]

        if plot:
            plt.plot(x, color="gainsboro", label= "Raw")
            plt.plot(filtered, color="royalblue", label= "Post filter")
            plt.show()

        return filtered

def preprocess_gsr(x, sampling_rate, plot= False):
    return preprocess_signal(x, plot, filter_method= butter_filter, cutoff_freq= 4, fs= sampling_rate,
order= 5, btype= "lowpass")

def preprocess_resp(x, sampling_rate, plot= False):
    # Source: https://www.mdpi.com/1424-8220/20/14/3884/htm
    return preprocess_signal(x, plot, filter_method= butter_bandpass_filter, low= 0.01, high= 1.5,
fs= sampling_rate, order= 2)

def preprocess_emg(x, sampling_rate, plot= False):
```

```
        # Source: https://github.com/PIA-Group/BioSPPy/blob/master/biosppy/signals/emg.py
        def emg_filter(x):
                return butter_filter(x, cutoff_freq= 100, fs= sampling_rate, order= 4, btype= "lowpass")
        return preprocess_signal(x, plot, filter_method= emg_filter)


#-------------------------------------------------------------------------------------------------
# Filter
#-------------------------------------------------------------------------------------------------
def fir_bandpass(x, low, high, fs, order):
        f1 = 2. * low / float(fs)
        f2 = 2. * high / float(fs)
        a = np.array([1])
        b = signal.firwin(numtaps=order,
                                                cutoff= [f1, f2],
                                                pass_zero=False)
        return signal.filtfilt(b, a, x)

def butter_bandpass(low, high, fs, order):
        low = 2. * low / float(fs)
        high = 2. * high / float(fs)
        b, a = butter(order, [low, high], btype='band')
        return b, a

def butter_bandpass_filter(data, low, high, fs, order):
        b, a = butter_bandpass(low, high, fs, order=order)
        y = lfilter(b, a, data)
        return y

def butter_pass(cutoff, fs, order, btype):
```

```
    normal_cutoff = 2. * cutoff / float(fs)
    b, a = signal.butter(order, normal_cutoff, btype= btype)
    return b, a

def butter_filter(data, cutoff_freq, fs, order, btype):
    b, a = butter_pass(cutoff_freq, fs, order=order, btype= btype)
    y = signal.filtfilt(b, a, data, padlen= 2)
    return y

def remove_ecg_wandering(ecg, plot= False):
    """Implements the removal of ecg baseline wandering presented in Thiam et al.

    Detrends the signal by subtracting a fifth-degree polynomial least-squares fit from the given signal.

    Parameters
    ----------
    x: np/list. ECG data of the BioVid dataset to transform.
    plot: bool. Boolean whether to plot the input and result or not. Default is set to False.

    Returns
    -------
    np: transformed data.
    """

    x = np.arange(len(ecg))
    poly_coefficients = np.polyfit(x, ecg, 5)
    poly = np.poly1d(poly_coefficients)
    fitted = poly(x)
```

```
        result = ecg-fitted

        if plot:
                plt.plot(ecg, label= "Input")
                plt.plot(fitted, label= "5 degree poly fit")
                plt.plot(result, label= "Result")
                plt.legend()
                plt.show()

        return result

def preprocess_np(x, sensor_names, sampling_rate):

        if type(x) != np.ndarray:
                print("Type should be numpy but is '{}'".format(type(x)))
                return

        if len(sensor_names) != x.shape[2]:
                print("Given sensor names and shape of numpy should match but are '{}' and
'{}'.".format(sensor_names, x.shape[2]))
                return

        func_dict = {"Bvp": preprocess_bvp, "Eda_E4": preprocess_gsr, "Resp": preprocess_resp,
"Eda_RB": preprocess_gsr,
                                "Ecg": preprocess_ecg, "Emg": preprocess_emg, "gsr": preprocess_gsr,
"ecg": preprocess_ecg, "emg_trapezius": preprocess_emg,
                                "Tmp":None, "Ibi": None, "Hr": None}

        for sensor_idx, sensor in enumerate(sensor_names):
                #plt.plot(x[0, :, sensor_idx], label= sensor + "_raw")
```

```
            func = func_dict[sensor]
            if func is None:
                continue
            x[:, :, sensor_idx] = np.apply_along_axis(arr= x[:, :, sensor_idx], func1d= func, axis= 1, sampling_rate= sampling_rate, plot= False)
            #plt.plot(x[0, :, sensor_idx], label= sensor + "_preprocessed")

    return x


def preprocess_df(df, plot):
    """Function to preprocess a given df.
    Columns named [Eda_E4, Eda_RB, Resp, Ecg, Emg] are preprocessed and updated.

    Parameters
    ----------
    df: Panda. Dataframe to update.
    plot: Bool. Whether to plot the preprocessing process or not.

    Returns
    -------
    df: Preprocessed dataframe
    """

    func_dict = {"Eda_E4": preprocess_gsr, "Resp": preprocess_resp, "Eda_RB": preprocess_gsr, "Ecg": preprocess_ecg, "Emg": preprocess_emg}
    df = df.copy() # Create a copy to avoid 'SettingwithCopyWarning'

    for column in df.columns:
```

```
        if column in func_dict:
                preprocess_func = func_dict[column]
                df[column]= preprocess_func(df[column], plot= plot)
                if plot:
                        plt.title(column)
                        plt.show()

return df
```