Appendix 4: Features Extraction Code

```
# pip install natsort

import sys
from io import StringIO

error_output = StringIO()
sys.stderr = error_output

warning_output = StringIO()
sys.stdout = warning_output

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy import signal
from scipy.signal import firwin, lfilter
from scipy.signal import butter, filtfilt
from scipy.signal import cheby2, filtfilt
from scipy.signal import savgol_filter
from scipy.signal import find_peaks, peak_widths
from scipy.integrate import simpson
from natsort import natsorted
import shutil
import os
import pywt
```

code to sort out _1 _2 _3 .txt files

```
folders = ['_1', '_2', '_3']
for folder in folders:
    os.makedirs(folder, exist_ok=True)

files = os.listdir('data')

for file in files:
    if file.endswith('_1.txt'):
        shutil.move(os.path.join('data', file), '_1')
    elif file.endswith('_2.txt'):
        shutil.move(os.path.join('data', file), '_2')
    elif file.endswith('_3.txt'):
        shutil.move(os.path.join('data', file), '_3')

print("Files sorted into folders _1, _2, and _3.")
```

```
Files sorted into folders _1, _2, and _3.
```

author's signature [illegible]

folder _1

```
path = "_1/10_1.txt"

ppg = pd.read_csv(path, sep="\t", header=None).T.dropna()
```

reading provided annotated dataframe with patients physical features

```
df = pd.read_csv('PPG-BP dataset.csv',
                                sep="\t",
                                skiprows=1,
                                # header=None,
                                )
df.set_index('subject_ID', inplace=True)

df.head(3)
```

```
            Num. Sex(M/F)  Age(year)  Height(cm)  Weight(kg)  \
subject_ID
2              1   Female         45         152          63
3              2   Female         50         157          50
6              3   Female         47         150          47

            Systolic Blood Pressure(mmHg)  Diastolic Blood
Pressure(mmHg)  \
subject_ID

2                                     161
89
3                                     160
93
6                                     101
71

            Heart Rate(b/m) BMI(kg/m^2)         Hypertension Diabetes
\
subject_ID

2                        97       27,27  Stage 2 hypertension      NaN

3                        76       20,28  Stage 2 hypertension      NaN

6                        79       20,89                Normal      NaN


           cerebral infarction cerebrovascular disease
subject_ID
2                          NaN                     NaN
```

author’s signature [illegible]

```
3                    NaN                    NaN
6                    NaN                    NaN
```

filter functions

```
def mwa(signal, window=20):

    signal_mwaved = signal.rolling(window, center=True,
closed='both').mean()
    signal_mwaved = np.asarray(signal_mwaved).ravel()
    return signal_mwaved


def moving_average_window(signal, window_size=100):

    signal = np.asarray(signal).ravel()
    smoothed_signal = np.zeros(len(signal))
    for i in range(len(signal)):
        start_index = max(0, i - window_size // 2)
        end_index = min(len(signal), i + window_size // 2 + 1)
        smoothed_signal[i] = np.mean(signal[start_index:end_index])

    return smoothed_signal



#def mwa (signal, window):
#    signal_mwaved = signal.rolling(window).mean()
#    return signal_mwaved


def detrend (signal, poly_order=3, offset=400):
    no_nan = signal[~np.isnan(signal)]
    poly_coefficients = np.polyfit(range(len(no_nan)), no_nan,
poly_order)
    poly_baseline = np.poly1d(poly_coefficients)
    ppg_detrended = no_nan - poly_baseline(range(len(no_nan))) +
offset

    return ppg_detrended


def fir_filter_hanning(signal, cutoff_freq =  0.45 * 800, fs = 800,
num_taps=50):
    nyquist_freq = 0.5 * fs
    normalized_cutoff_freq = cutoff_freq / nyquist_freq
    taps = np.hanning(num_taps)
    taps /= np.sum(taps)
    signal = np.asarray(signal).ravel()
    filtered_signal = np.convolve(signal, taps, mode='same')
```

author's signature [illegible]

```
    return filtered_signal



def filter_savicky_golay(signal, window_length=20, polyorder=4):
    if window_length > len(signal):
        window_length = len(signal) - 1  # Adjust window length if
it's too large

    filtered_signal = savgol_filter(signal,
window_length=window_length, polyorder=polyorder, mode='nearest')

    return filtered_signal



def filter_butterworth(signal, fs=1000):
    # 4th order Butterworth filter with cutoff at 0.1 * Nyquist
frequency
    b, a = butter(4, 0.1, 'low', fs=fs)
    filtered_signal = filtfilt(b, a, signal)

    return filtered_signal


def wavelet_transform_db4(signal, level = 4):
    waveletname = 'db4'
    [cA1, cD1, cD2, cD3, cD4] = pywt.wavedec(signal, waveletname,
level = level)

    return cA1
```

signal processing functions

```
def peak_1_coord(signal) -> int:
    """ returns X coordinate of the cycle peak"""

    try:
        signal_squeezed = signal.squeeze()
        peaks, properties = find_peaks(signal_squeezed, prominence=1,
width=20)
        peak_1 = peaks[0]
    except:
        print(signal, key)

    return peak_1


def peak_2_coord(signal) -> int:
```

author’s signature [illegible]

```
    """ returns X coordinate of the cycle peak"""

    try:
        signal_squeezed = signal.squeeze()
        peaks, properties = find_peaks(signal_squeezed, prominence=1,
width=20)
        peak_2 = peaks[1]
    except:
        print(signal, key)

    return peak_2


def peak_3_coord(signal) -> int:
    """ returns X coordinate of the cycle peak"""

    peak_3 = None  # Set a default value

    try:
        signal_squeezed = signal.squeeze()
        peaks, properties = find_peaks(signal_squeezed, prominence=1,
width=20)

        if len(peaks) >= 3:
            peak_3 = peaks[2]
    except Exception as e:
        print(f"An error occurred: {e}")

    return peak_3



def dia_starts(signal) -> int:
    """ extracts start of diastole at right_ips parameter of
find_peaks properties"""

    try:
        signal_squeezed = signal.squeeze()
        peaks, properties = find_peaks(signal_squeezed, width = 100)
        dia_start_coord   = properties['right_ips']
        dia_start_coord = dia_start_coord[0]
    except:
        print(signal, key)

    return int(dia_start_coord)




def SNR(signal) -> int:
```

author's signature [illegible]

```
    """ returns the ratio between level of signal and its noise"""

    try:
        signal = np.array(signal, dtype=float)

        # Filter out nan values from the signal array
        signal = signal[~np.isnan(signal)]

        noise_level = np.std(signal)
        signal_power = np.mean(signal**2)
        noise_power = noise_level**2

        snr_db = 10 * np.log10(signal_power / noise_power)
    except Exception as e:
        print(e)
        snr_db = np.nan

    return round(snr_db, 2)


def pi(signal) -> int:
    """ returns perfusion index the ratio of pulsatile blood flow to
non-pulsatile
    blood flow in a patient's peripheral tissue"""
    try:
        signal = np.array(signal, dtype=float)
        ac_component = np.max(signal) - np.min(signal)
        dc_component = np.mean(moving_average_window(signal))
        pi = ac_component / dc_component
        pi_round = round(pi, 2)
    except:
        print(signal, key)
    return pi_round

def sys_to_dia_ratio(signal, fs = 1000 ) -> int:
    """ fs is cut-off frequency according to Nyquist theorem for this
sample rate is taken of 1000"""

    try:
        signal = np.array(signal, dtype=float)
        peaks, _ = find_peaks(moving_average_window(signal),
height=np.mean(signal))
        systolic_phase_duration = len(peaks) / fs
        diastolic_phase_duration = len(signal) / fs -
systolic_phase_duration
        ratio = systolic_phase_duration / diastolic_phase_duration
        ratio_round = round(ratio, 2)
    except:
        print(signal, key)
```

author’s signature [illegible]

```
    return ratio_round


def avg_pulse_rate(signal, fs = 1000) -> int:
    """ calculates avg pulse rate to compare with given one """

    try:
        signal = np.array(signal, dtype=float)
        peaks, _ = find_peaks(moving_average_window(signal),
height=np.mean(signal), prominence=1, width=20)
        ibis = np.diff(peaks) / fs
        pulse_rates = 60 / ibis
        avg_pulse_rate = np.mean(pulse_rates)
        avg_pulse_rate = round(avg_pulse_rate, 2)
    except:
        print(signal, key)


    return avg_pulse_rate

def extract_first_cycle(signal) -> int:
    """ the sequence of functions to define dicrotic notch and peak of
diastole phase of the first cycle:
    1. returns the cut off the first cycle """

    try:
        signal = np.array(signal, dtype=float)
        peaks, prop = find_peaks(moving_average_window(signal),

height=np.mean(moving_average_window(signal)),
                                 prominence=1,
                                 width=20)
        first_start = prop['left_bases'][0]
        first_end = prop['left_bases'][1]
        first_cycle =
moving_average_window(signal[first_start:first_end])
    except:
        print(signal, key)

    return first_cycle

def first_derivative(signal) -> int:
    """ the sequence of functions to define dicrotic notch and peak of
diastole phase of the first cycle:
    2. returns the first derivative curve for the first cycle """

    try:
        signal = np.array(signal, dtype=float)
        first_cycle =
extract_first_cycle(moving_average_window(signal, window_size=120))
```

author's signature [illegible]

```python
        first_derivative = np.diff(first_cycle)
    except:
        print(signal, key)

    return first_derivative

def second_derivative(signal) -> int:
    """ the sequence of functions to define dicrotic notch and peak of 
diastole phase of the first cycle:
    3. returns the second derivative curve for the first cycle"""

    try:
        signal = np.array(signal, dtype=float)
        first_cycle = 
extract_first_cycle(moving_average_window(signal, window_size=120))
        first_derivative = np.diff(first_cycle)
        second_derivative = np.diff(first_derivative)
    except:
        print(signal, key)

    return second_derivative

def dic_notch(signal) -> int:
    """ the sequence of functions to define dicrotic notch and peak of 
diastole phase of the first cycle:
    4. returns the dicrotic notch based on the second derivative """

    try:
        signal = np.array(signal, dtype=float)
        second_deriv_result = second_derivative(signal)
        peaks_deriv, prop_deriv = 
find_peaks(moving_average_window(second_deriv_result, 

window_size=40), 

height=np.mean(second_deriv_result)
                                            )
        notch = peaks_deriv[3]
    except UnboundLocalError:
        #print("UnboundLocalError occurred")
        notch = None
    except IndexError:
        #print("IndexError occurred")
        notch = None

    return notch



def dia_peak(signal) -> int:
```

author's signature [illegible]

```
    """ the sequence of functions to define dicrotic notch and peak of
diastole phase of the first cycle:
    5. returns the peak of diastole phase based on the second
derivative """

    try:
        signal = np.array(signal, dtype=float)
        second_deriv_result = second_derivative(signal)
        peaks_deriv, prop_deriv =
find_peaks(moving_average_window(second_deriv_result,

window_size=40),

height=np.mean(second_deriv_result)
                                                            )
        notch = peaks_deriv[3]
        end = peaks_deriv[4]
        peak = (notch + end) / 2

    except IndexError:
        peak = None

    return peak



def AUC(signal):
    """ area under curve returns integrated area under the given
waveform cycle"""

    auc = None  # Set a default value

    try:
        signal_squeezed = signal.squeeze()
        first_cycle =
extract_first_cycle(moving_average_window(signal_squeezed,
window_size=100))
        auc = simpson(first_cycle, dx=5)

    except Exception as e:
        print(f"An error occurred: {e}")

    if auc is not None:
        auc = round(auc, 2)

    return auc


def cycle_width_0_25(signal):
```

author’s signature [illegible]

```
    try:
        # signal = np.array(signal, dtype=float)
        #first_cycle =
extract_first_cycle(moving_average_window(signal, window_size=120))
        peaks, _ = find_peaks(signal, prominence=1, width=20)
        width_025 = peak_widths(signal, peaks, rel_height= 0.75)[0][0]

    except:
        print(signal, key)

    return round(width_025, 2)


def cycle_width_0_50(signal):

    try:
        # signal = np.array(signal, dtype=float)
        # first_cycle =
extract_first_cycle(moving_average_window(signal, window_size=120))
        peaks, _ = find_peaks(signal, prominence=1, width=20)
        width_050 = peak_widths(signal, peaks, rel_height= 0.50)[0][0]

    except:
        print(signal, key)

    return round(width_050, 2)


def cycle_width_0_75(signal):

    try:
        # signal = np.array(signal, dtype=float)
        # first_cycle =
extract_first_cycle(moving_average_window(signal, window_size=120))
        peaks, _ = find_peaks(signal, prominence=1, width=20)
        width_075 = peak_widths(signal, peaks, rel_height= 0.25)[0][0]

    except:
        print(signal, key)

    return round(width_075, 2)
```

Cycle to read signals in

```
folder_path = "/Users/alexandra/Desktop/bäkatöö/_1"
signals = {}

# List all files in the folder
files = os.listdir(folder_path)
```

author’s signature [illegible]

```
files = natsorted(files)


for file_name in files:
    if file_name.endswith(".txt"):
        file_path = os.path.join(folder_path, file_name)


        # Read the contents of the file and store them as a numpy
array
        with open(file_path, 'r') as file:
            data = np.genfromtxt(file, delimiter='\t')


        # Store the numpy array in the 'signals' dictionary with the
file name as the key
        signals[file_name] = data

# getting signal prefixes without _number.txt, which would correspond
to patients IDs in a dataframe

new_signals = {}

for file_name in signals.keys():
    base_name = os.path.splitext(file_name)[0]
    patient_id = base_name.split('_')[0]
    new_signals[patient_id] = signals[file_name]


# df.head(4)
```

Calling the functions for signals from the cycle, and writing the results into dataframe

```
snr_feat = []
pk_1_savgol         = []
pk_2_savgol         = []
pk_3_savgol         = []
dia_coord_savgol    = []
perf_index_savgol   = []
sys_dia_ratio_savgol = []
avg_PR_savgol       = []
area_under_1_cycle_savgol = []
cyc_width_025 = []
cyc_width_050 = []
cyc_width_075 = []


for key, value in new_signals.items():
```

author’s signature [illegible]

```
    snr = SNR(value)

    ppg_mwad = moving_average_window(value, window_size=100)
    filtered_signal_SG = filter_savicky_golay(ppg_mwad)
    detrended_signal_SG = detrend(filtered_signal_SG)

    peak_1_sg     = peak_1_coord(detrended_signal_SG)
    peak_2_sg     = peak_2_coord(detrended_signal_SG)
    peak_3_sg     = peak_3_coord(detrended_signal_SG)
    dia_coord_sg  = dia_starts(detrended_signal_SG)
    perf_ind_sg   = pi(detrended_signal_SG)
    s_d_ratio_sg  = sys_to_dia_ratio(detrended_signal_SG)
    PR_sg         = avg_pulse_rate(detrended_signal_SG)
    area_under_cycle = AUC(detrended_signal_SG)
    cycle_width_25 = cycle_width_0_25(detrended_signal_SG)
    cycle_width_50 = cycle_width_0_50(detrended_signal_SG)
    cycle_width_75 = cycle_width_0_75(detrended_signal_SG)



    snr_feat.append(snr)
    pk_1_savgol.append(peak_1_sg)
    pk_2_savgol.append(peak_2_sg)
    pk_3_savgol.append(peak_3_sg)
    dia_coord_savgol.append(dia_coord_sg)
    perf_index_savgol.append(perf_ind_sg)
    sys_dia_ratio_savgol.append(s_d_ratio_sg)
    avg_PR_savgol.append(PR_sg)
    area_under_1_cycle_savgol.append(area_under_cycle)
    cyc_width_025.append(cycle_width_25)
    cyc_width_050.append(cycle_width_50)
    cyc_width_075.append(cycle_width_75)



df["SNR"] = snr_feat
df["peak_1_coord"]       = pk_1_savgol
df["peak_2_coord"]       = pk_2_savgol
df["peak_3_coord"]       = pk_3_savgol
df["diastole_start"]     = dia_coord_savgol
df["perfusion_index"]    = perf_index_savgol
df["sys_to_dia_ratio"]   = sys_dia_ratio_savgol
df["average_Pulse_Rate"]  = avg_PR_savgol
df["area_under_first_cycle"] = area_under_1_cycle_savgol
df["cycle_width_at_25"] = cyc_width_025
df["cycle_width_at_50"] = cyc_width_050
df["cycle_width_at_75"] = cyc_width_075
```

author’s signature [illegible]

```
df.to_csv("PPG_features_.csv", index=False)

snr_feat = []
pk_1_savgol         = []
pk_2_savgol         = []
pk_3_savgol         = []
dia_coord_savgol    = []
perf_index_savgol   = []
sys_dia_ratio_savgol = []
avg_PR_savgol       = []
dicr_notch_savgol   = []
area_under_1_cycle_savgol = []




for key, value in new_signals.items():

    ### ================ PART 1: Calculating properties by calling
functions

    # SNR ratio
    snr = SNR(value)

    # signal filtered with moving average
    ppg_mwad = moving_average_window(value, window_size=100)


    # properties of a signal filtered with Savicky-Golay:

    filtered_signal_SG = filter_savicky_golay(ppg_mwad)
    detrended_signal_SG = detrend(filtered_signal_SG)

    peak_1_sg     = peak_1_coord(detrended_signal_SG)
    peak_2_sg     = peak_2_coord(detrended_signal_SG)
    peak_3_sg     = peak_3_coord(detrended_signal_SG)
    dia_coord_sg  = dia_starts(detrended_signal_SG)
    perf_ind_sg   = pi(detrended_signal_SG)
    s_d_ratio_sg  = sys_to_dia_ratio(detrended_signal_SG)
    PR_sg         = avg_pulse_rate(detrended_signal_SG)
    d_notch_sg    = dic_notch(detrended_signal_SG)
    area_under_cycle = AUC(detrended_signal_SG)



    ### ================ PART 2: saving functions result to list

    snr_feat.append(snr)
    pk_1_savgol.append(peak_1_sg)
```

author’s signature [illegible]

```
    pk_2_savgol.append(peak_2_sg)
    pk_3_savgol.append(peak_3_sg)
    dia_coord_savgol.append(dia_coord_sg)
    perf_index_savgol.append(perf_ind_sg)
    sys_dia_ratio_savgol.append(s_d_ratio_sg)
    avg_PR_savgol.append(PR_sg)
    dicr_notch_savgol.append(d_notch_sg)
    area_under_1_cycle_savgol.append(area_under_cycle)



    ### ================ PART 4: writing results to the dataframe from
3.


df["SNR"] = snr_feat
df["peak_1_coord"]       = pk_1_savgol
df["peak_2_coord"]       = pk_2_savgol
df["peak_3_coord"]       = pk_3_savgol
df["diastole_start"]     = dia_coord_savgol
df["perfusion_index"]    = perf_index_savgol
df["sys_to_dia_ratio"]   = sys_dia_ratio_savgol
df["average_Pulse_Rate"]  = avg_PR_savgol
df["dicrotic_notch_coord"] = dicr_notch_savgol
df["area_under_first_cycle"] = area_under_1_cycle_savgol



df.to_csv("patient_data_with_properties.csv", index=False)

df.shape
```

```
(219, 26)
```

author's signature [illegible]