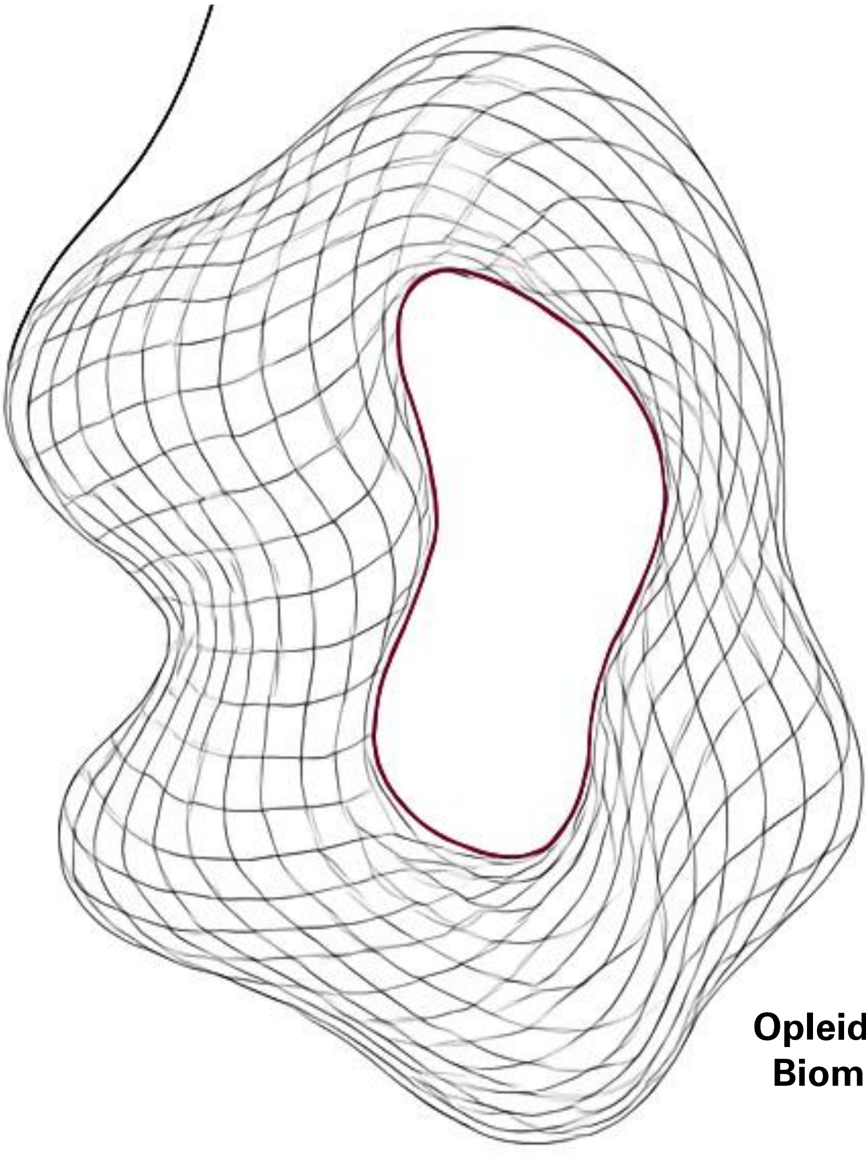
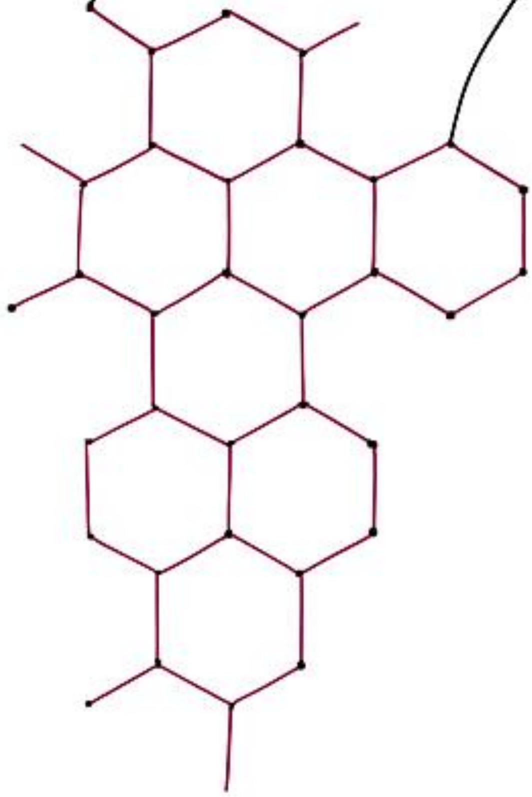
ADAPTIVE GOAL-SETTING APPROACHES FOR SLEEP MANAGEMENT IN DUTCH OLDER ADULTS

By Thijmen Schuurman, s2558963
20-05-2025

Graduation committee:
ir. F. Muijzer (University of Twente, EEMCS-BSS)
dr. N. Sharma (University of Twente, ET-DPM-IxD)
A. van Kraaij, MSc (Extern, OnePlanet Research Center)

Chair:
dr. A. Witteveen (University of Twente, EEMCS-BSS)





Opleiding BSc Biomedische Technologie
Biomedical Signals and Systems (BSS)

UNIVERSITEIT TWENTE.

PREFACE

In this report for my bachelor's thesis, I describe the research I conducted on the implementation of an adaptive goal-setting feature within a mobile application for older adults. This research is part of the MOCIA project, which aims to develop a lifestyle-based application to help older adults practice various domains to maintain and enhance their cognitive health.
Over the past 15 weeks, I have discovered, learned, and enjoyed a lot. I started this project with little to no prior knowledge of many of the topics and skills involved. It was very meaningful for me to help lay the foundation for future research in this area.
Throughout this process, I received a lot of support from three people who helped me understand the project and what I was working on, supported me on various challenges, and guided me to the finish line. I would like to sincerely thank ir. Frodo Muijzer (University of Twente), dr. Nikita Sharma (University of Twente), and Alex van Kraaij (OnePlanet Research Center). In addition, I would like to thank Annemieke Witteveen, PhD (University of Twente) for being the chair of my graduation committee.

Enschede, 20-05-2025
Thijmen Schuurman

TABLE OF CONTENTS

ABSTRACT (ENGLISH)

This study is conducted to assess the implementation of an adaptive goal-setting feature for sleep duration and quality within a custom mobile application for older adults. The study is part of the MOCIA project, which aims to develop a lifestyle-based mobile application to help maintain and enhance the cognitive health by promoting positive behaviors across multiple domains.
A framework was developed to implement the adaptive goal-setting feature and assess its feasibility for future use. Eleven subjects participated in the study, which lasted 15 days. During this period, participants wore a Garmin vívosmart® 5 continuously and completed a daily questionnaire about their perceived sleep quality.
Sleep durations were calculated using a custom algorithm that processed both the objective and subjective data. Based on this, personalized recommendations were generated and provided to the participants. These included their personalized sleep goals as well as generic tips to support behavioral change and improve sleep quality.
At the end of the study, participants attended a focus group session to evaluate the study. Combined with the collected data, multiple limitations were identified. The custom algorithm failed to return sleep durations in 31% of the measurements (42 out of 135), and only 40% of the returned sleep durations matched the self-reported sleep duration within ± 45 minutes. Despite these limitations, 3 out of 11 participants showed improvements across all measured domains, e.g. increased consistency in sleep duration, improved perceived sleep quality, fewer toilet visits during the night, and a higher frequency of falling asleep within 30 minutes. Five additional participants improved in most of these domains. Furthermore, 70% of the participants preferred behavior-based recommendations over fixed sleep duration goals, and 78% valued visual feedback, emphasizing the potential of personalized, engaging recommendations to support healthy sleep behavior.
Technical problems with the OnePlanet Research mobile application were also reported, which led to two dropouts and some frustration among other participants. This application uses the Garmin Health SDK, which provides access to all health and fitness activity data, such as heart rate (HR), heart rate variability (HRV) and steps, which is beneficial for the custom algorithm developed by imec NL. A more stable solution may be switching to the Garmin Health API via the Garmin Connect application, although this comes with its own downsides.
Overall, this study has the potential to be usable and effective in supporting positive sleep behavior change to ultimately maintain and enhance the cognitive health of older adults.

ABSTRACT (NEDERLANDS)

Deze studie is uitgevoerd om de implementatie te beoordelen van een adaptieve doelstellingsfunctie voor slaapduur en -kwaliteit, in een mobiele applicatie voor ouderen. De studie maakt deel uit van het MOCIA-project, dat als doel heeft om een leefstijlgerichte mobiele applicatie te ontwikkelen ter ondersteuning van het behoud en de verbetering van cognitieve gezondheid, door het stimuleren van positief gedrag op meerdere vlakken.
Er is een *framework* ontwikkeld om de adaptieve doelstellingsfunctie te implementeren, en de haalbaarheid ervan voor toekomstig gebruik te beoordelen. Elf deelnemers namen deel aan de studie, welke 15 dagen duurde. Gedurende deze periode droegen de deelnemers continu een Garmin vívosmart® 5 en vulden zij dagelijks een vragenlijst in over hun subjectieve slaapkwaliteit.
Slaapduur werd berekend met behulp van een algoritme dat zowel objectieve als subjectieve gegevens verwerkte. Op basis hiervan werden gepersonaliseerde aanbevelingen gegenereerd en aan de deelnemers verstrekt. Deze aanbevelingen bestonden uit persoonlijke slaapdoelen en algemene tips ter ondersteuning van gedragsverandering en verbetering van de slaapkwaliteit.
Aan het einde van de studie namen de deelnemers deel aan een *focusgroup*-sessie om de studie te evalueren. In combinatie met de verzamelde data kwamen hierbij meerdere beperkingen aan het licht. Het algoritme gaf in 31% van de metingen (42 van de 135) geen slaapduur terug, en slechts 40% van de gegenereerde slaapduurgegevens kwam binnen ± 45 minuten overeen met de zelfgerapporteerde slaapduur. Ondanks deze beperkingen lieten 3 van de 11 deelnemers verbeteringen zien op alle gemeten domeinen, zoals meer consistentie in slaapduur, betere subjectieve slaapkwaliteit, minder nachtelijke toiletbezoeken en vaker binnen 30 minuten in slaap vallen. Vijf andere deelnemers verbeterden op de meeste van deze domeinen. Daarnaast gaf 70% van de deelnemers de voorkeur aan gedragsgerichte aanbevelingen boven vaste slaapdoelen, en waardeerde 78% de visuele feedback, wat het potentieel benadrukt van gepersonaliseerde en motiverende aanbevelingen ter ondersteuning van gezond slaapgedrag.
Er werden ook technische problemen gemeld met de OnePlanet Research mobiele applicatie, wat leidde tot twee uitvallers en frustratie bij enkele andere deelnemers. Deze applicatie maakt gebruik van de Garmin Health SDK, die toegang geeft tot alle gezondheids- en activiteitsgegevens, zoals *heart rate (HR)*, *heart rate variability (HRV)* en stappenteller, wat voordelig is voor het, door imec NL ontwikkelde, algoritme. Een stabielere oplossing zou het gebruik van de Garmin Health API via de Garmin Connect-applicatie kunnen zijn, hoewel dit ook nadelen met zich meebrengt.
Al met al laat deze studie zien dat het concept potentie heeft om positief slaapgedrag te ondersteunen en daarmee bij te dragen aan het behoud en de verbetering van de cognitieve gezondheid van oudere volwassenen.

1 INTRODUCTION

With a prevalence of one in five, cognitive decline is becoming a larger healthcare issue. The number of people with neurodegenerative diseases has increased due to aging, with a current prevalence of 310 thousand patients in the Netherlands. The number is expected to double over the next 25 years, reaching more than 620 thousand patients with neurodegenerative diseases[1]. Dementia is an umbrella term for a particular set of symptoms characterized by difficulties with memory, language, problem solving, and other cognitive skills that interfere with daily activities. It results from changes in the brain, which can occur in various ways and lead to different forms of the condition[2]. Globally, Alzheimer's is the leading cause of dementia. As of 2019, approximately 9.8% of the European population aged 65 and older had been diagnosed with Alzheimer's disease.[3, 4]. This is just the tip of the iceberg when talking about cognitive decline[5].
Many factors play a role in the occurrence of cognitive decline, but many of them can be controlled. There are also some preventive measures for cognitive decline, such as physical activity and cognitive training[6]. Setting goals is an effective strategy to counter cognitive decline. Understanding how goal-setting works and why it is effective is crucial for developing interventions.

In the last 50 years, goal-setting theories have made their way into our daily lives. Locke and Latham developed the goal setting theory over a 25-year period based on 400 laboratory and field studies. The goal setting theory uses specific quantitative (hard) goals, in a determined time frame, to obtain higher results than telling a person to do their best[7, 8]. Nowadays, goal-setting theory is applied in nearly every aspect of life, whether it is receiving a task from your boss or being advised by your doctor to exercise more. In a society that constantly emphasizes self-improvement, setting goals is a vital first step toward changing behavior and making meaningful progress, such as improving diet and increasing physical activity[9].

It is known that bad sleep habits play an important role in neurodegeneration, and to counter cognitive decline through goal-setting in the sleep domain, different approaches to formulating goals can be applied, for example, a *gain goal frame*, which is used to improve one's resources. The goal of increasing one's physical activity to become healthier is an example of a *gain goal frame*[10, 11]. There are many ways to track your goal to become healthier, for example, tracking your weight using a scale or nutrition using a mobile application. However, a smartwatch can also be used to track your goal progress. This is because a smartwatch can measure many features that can also give indications about your health. All these data can be collected and defined as personal informatics systems, those that help people collect personally relevant information for the purpose of self-reflection and gaining self-knowledge. Personal informatics stands out as an interesting area of study within human-computer interaction. These systems are designed to help individuals gain deeper insight into their own behavior[12]. And with this insight, individuals can improve themselves.

Because goal setting and personal informatics both play a key role in encouraging self-awareness and behavior change, bringing these approaches together can strengthen efforts to support cognitive health in older adults. The MOCIA project (Maintaining Optimal Cognitive function In

Ageing) aligns with this objective, as it aims to identify individuals at increased risk of cognitive decline and reduce this risk through personalized interventions. These interventions are provided via a multi-domain mobile application, with sleep as one of the domains.

Sleep is a critical factor that influences cognitive functioning [10]. Poor sleep quality and insufficient sleep duration have been associated with decreased cognitive performance and increased risk of neurodegenerative diseases[13, 14, 15, 16]. Therefore, supporting healthy sleep habits is one of the main domains in MOCIA's approach to maintaining cognitive health.

However, motivating older adults to improve sleep behavior and adjust routines can be challenging. Personalized support and recommendations, based on each individual's needs, preferences and progress, can improve engagement. This is where the combination of of goal-setting theory and personal informatics become useful. Goal setting helps them to establish concrete goals, while personal informatics helps them to monitor progress, reflect on behavior, and adapt their goals based on feedback.

In this project, the aim is to combine these two approaches by co-developing a feature that enables users to set and track personalized sleep goals. Using both wearable data and application-based questionnaires, this feature will become part of an adaptive sleep management module within the mobile application. This feature supports MOCIA's goal of providing personalized interventions, by helping users make behavior changes that contribute to maintaining cognitive health as they age. Therefore, this study aims to identify how such an adaptive goal-setting feature can be developed, evaluated, and implemented using wearable and self-reported data to improve personalization, user engagement, and support better cognitive health in MOCIA's target population.

2 BACKGROUND

Bad sleep habits play an important role in neurodegeneration. As people age, their sleep patterns change over time. Their total sleep time, sleep efficiency, number of sleep disturbances, and sleep latency have deteriorated[10]. Furthermore, sleep is essential for cognitive processing, and disruptions can affect cognitive performance in all age groups[13, 14]. Meta-analyses have shown that individuals with sleep disturbances or insomnia have a significantly higher risk of developing cognitive diseases, including Alzheimer's disease and other forms of neurodegeneration[15, 16]. Given this strong connection, understanding and accurately measuring sleep is important when trying to improve cognitive health through behavioral change. Lifestyle-based approaches, such as physical activity, a healthy diet, cardiovascular health management, cognitive training, social engagement, stress reduction, and especially sleep management, have been shown to support cognitive functioning in older adults. Despite the known benefits of these practices, many individuals struggle to adopt and maintain them, especially in home settings where ongoing support can be limited.
One reason for this difficulty may be the daily routines that many older adults follow. For example, watching the evening news at 20:00, drinking tea at 22:30, and going to bed at 23:00 might form a consistent rhythm that is hard to break. When sleep improvement requires changes to this routine, it can be challenging to commit to new behaviors, especially when their are many options for change, which can be overwhelming. In such cases, prioritizing health goals can be of great help. Research by Conner et al.[17] suggests that focusing on one or two specific health goals can significantly increase the likelihood of behavior change. Providing older adults with concrete, personalized tips allows them to focus their efforts and gradually adjust their routine.

To effectively support behavior change and improve sleep patterns, reliable and accessible methods for monitoring sleep are essential. Understanding how sleep is measured is essential for offering useful personalized recommendations. The gold standard for assessing sleep is polysomnography (PSG). This method requires individuals to spend the night in a sleep laboratory under controlled conditions, supervised by a sleep technician. During such studies, multiple surface electrodes are attached to the body to measure various physiological indicators of sleep. These include brain activity via electroencephalography (EEG), eye movements, muscle tone, heart function, and respiratory patterns[18].

While PSG provides detailed and accurate data, it is impractical for daily use due to the complex setup. To be able to track sleep in everyday life, researchers and consumers increasingly rely on wrist-worn wearable devices, such as wrist-worn accelerometers. Where periods of minimal arm movement can be assessed as sleep[19]. Building on this, commercial wearables such as Garmin, Fitbit, and Apple Watch offer more advanced sleep tracking by combining accelerometer data with heart rate and other physiological signals. These devices provide users with accessible insights into their sleep patterns through their own algorithms.
However, a limitation of commercial wearables is that their algorithms are not publicly available, which results in less insights into how sleep durations are calculated and which factors play a more significant role than others. These parts would become a ”black box”, making it hard to understand or validate the outcomes. Besides that, access to data, such as heart rate variability

(HRV), is restricted when using their standard mobile applications, like Garmin Connect. To overcome these limitations in our study, imec NL uses a connection with the Garmin Health SDK. Through this connection, Garmin provides a JSON-formatted file which contains all data collected by the wearable device for each individual, including heart rate (HR), HRV, motion intensity, respiration, steps, stress, wellness, and zero-crossing data[20, 21]. The Garmin Health SDK connection is established via the OnePlanet Research mobile application, developed by imec NL. Because this app connects directly to the Garmin vívosmart® 5 (shown in Figure 1), it enables data transfer through the Garmin Health SDK, allowing us to collect data that can be processed by an algorithm developed by imec NL.

This access to the physiological data allows us to process the information ourselves and provides us more insight into the participant's sleep patterns. A specific valuable metric is HRV, which shows changes in the time intervals between consecutive heartbeats[22]. HRV and HR could be useful for future research within the MOCIA program, as both metrics vary across different sleep stages. Specifically, HR tends to decrease, along with reduced variability, during non-REM stages, while it increases with greater variability during REM sleep[23]. During non-REM stages the cardiovascular system is stable and parasympathetic cardiac modulation is more active, resulting in lower HR and higher HRV. During REM sleep, the cardiovascular system is unstable and sympathetic activity becomes more active, causing HR to increase and HRV to decrease [24, 25]. These physiological patterns could be useful indicators for assessing sleep quality and detecting sleep stage transitions.
This combination of wearable devices and personalized recommendations forms the basis for our approach to sleep management. By integrating these elements into a broader health-focused framework, we aim to make sleep improvement more accessible for older adults. This approach aligns with the goals of the MOCIA research program, which supports older adults by addressing key lifestyle domains associated with cognitive health. All domains, including sleep management, will be integrated into a mobile application developed by Vivica. This application will help users set personalized goals, monitor their progress, and take an active role in maintaining their cognitive health in a practical and user-friendly way.

Figure 1: Garmin vívosmart® 5 [26]

3 FRAMEWORK

To effectively integrate adaptive goal setting with personal informatics in the context of sleep management, a structured framework is necessary. This framework outlines how researchers and users can effectively utilize health-related data from wearable sensors and questionnaires to set meaningful goals. These goals are shaped by two key perspectives on well-being: hedonic and eudaimonic [27, 28]. Hedonic well-being is often related to the concept of happiness and is characterized by the presence of positive feelings, pleasure, enjoyment, satisfaction, and the absence of pain and discomfort. On the other hand, eudaimonic well-being focuses on personal growth, self-fulfillment, and a sense of meaning in life. This can relate to aspects such as autonomy, ethics, maturity, value, and relevance in an individual's life [29, 30]. By implementing both perspectives, the framework enables users to translate personal values and experiences into clear, manageable goals and to continuously refine these goals using personal data. Through this structured approach, researchers and users can simplify sleep-related challenges into actionable goals, facilitating effective behavior change and supporting cognitive health. The framework would consist of three phases in which various data is processed.

3.1 PHASE I - IDENTIFYING GOALS

In the first phase of the adaptive goal-setting framework (Figure 2), data is collected regarding the user's eudaimonic or hedonic needs. These eudaimonic or hedonic needs are often the basis of the user's motivation[31]. Users frequently express broad wishes, such as wanting to sleep more or to improve their sleep quality. However, to be able to use those needs and whishes, it needs to be transformed so that it is supported by the algorithm. Therefore, eudaimonic or hedonic needs are transformed into qualitative goals and finally into quantitative goals.
This framework builds on the goal-setting process and a different framework previously made by Nikita Sharma [32], and the initial structure used in this project was co-developed with the assistance of Nikita Sharma, who contributed plenty to the framework.

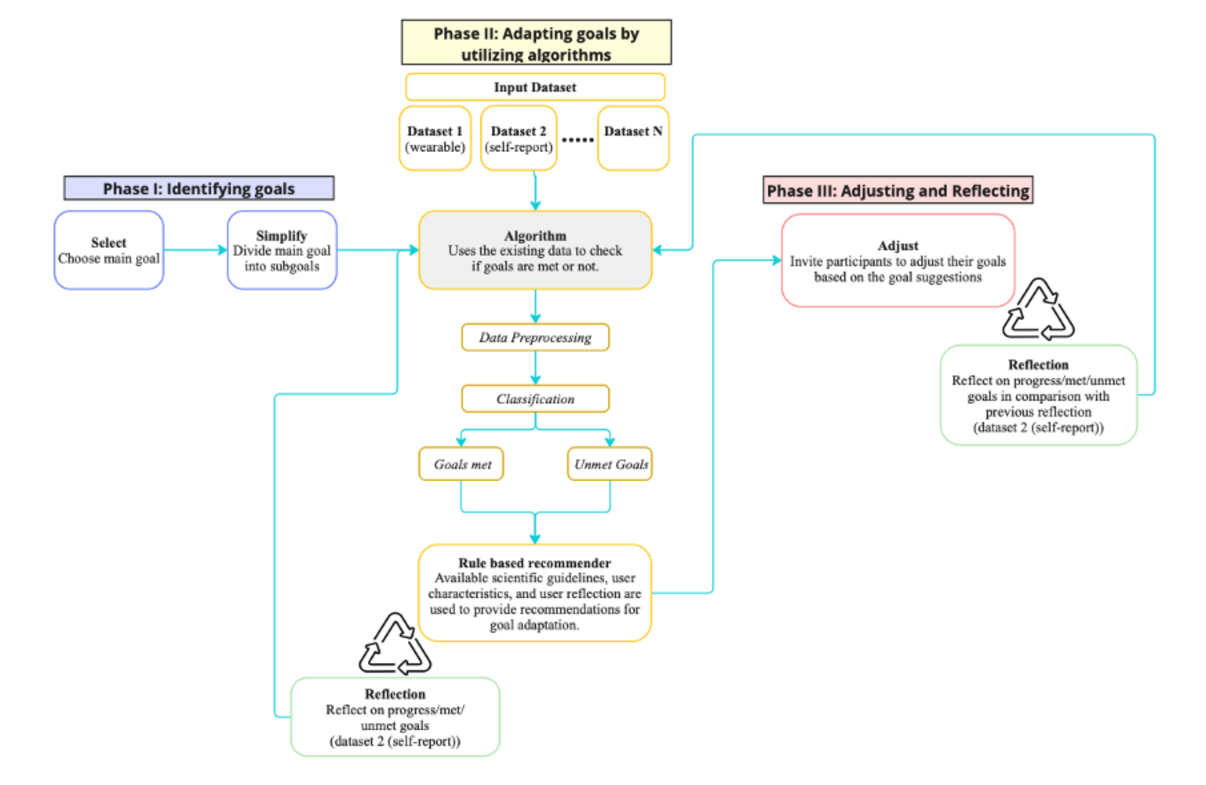


Figure 2: Adaptive goal-setting framework

3.2 PHASE II - ADAPTING GOALS BY UTILIZING ALGORITHMS

In the second phase, the algorithm processes all available data to generate a personalized recommendation. First, the wearable data is preprocessed and assigned a score that is used in the next steps. Next, the self-report data, containing information such as the participant's main goal, motivation, and perceived quality, is also scored. These two scores are weighted equally to determine which recommendations are the most suitable for the participant to sleep better or to increase the quality of their night's sleep. At the end of phase two, participants are asked to reflect on their previous progress and think more deeply about their sleep goal, including which aspects they would like to adjust or improve, and what their short- and long-term goals look like. Their responses are used to refine and adjust the recommendations moving forward.

3.2.1 DATASETS

Dataset 1 (wearable)

Dataset 1 contains the wearable data of the JSON file, used by the algorithm to calculate sleep duration. The Garmin vívosmart® 5 is connected via Bluetooth to a mobile phone. This phone has the OnePlanet Research mobile application installed. And this mobile application uses the Garmin Health API to collect data from the Garmin vívosmart® 5, such as HR, step count, and HRV. The algorithm uses these three data types to compute a sleep score.

Dataset 2 (self-report)

Dataset 2 consists of self-report data. It includes four questions about participants' perception of their sleep quality, as well as one question regarding their main goal. These questionnaires will be discussed in more detail in Chapter 4.4.3. The algorithm uses these self-report data to generate a sleep quality score. This score is weighted equally with the score from dataset 1

and combined into a total score. Based on this total score, personalized recommendations are generated to help improve both the duration and the quality of sleep.
Reflections collected from participants are not directly reintegrated into the algorithm but are used to tailor the recommendations by highlighting specific aspects that the participant wants to focus on.

3.2.2 ALGORITHM

The algorithm uses both datasets to generate personalized recommendations aimed at improving sleep and, ultimately, supporting cognitive health. To calculate the sleep score, it first estimates sleep duration using wearable data, specifically step count, HR and HRV. To enhance the accuracy of this estimate, the algorithm examines the period between the last recorded steps of the previous day and the first steps of the next morning. It checks whether the HR during this period is lower and the HRV is higher, compared to the surrounding periods. These physiological markers help validate that the identified window reflects actual rest [19]. To implement this approach, imec NL developed a script to analyze wearable data and detect sleep episodes. Below, the main steps of this data processing workflow are described.

Garmin data pre-processing algorithm (imec NL)

The algorithm checks for new Garmin data folders in the imec NL cloud for each active participant. If new data is available, the folder is unzipped, and relevant features are extracted: HR, hrvSdnn (standard deviation of all normal to normal R-R (NN) intervals), hrvRmssd (square root of the mean of the squares of successive NN interval differences), physicalActivityMets, physicalActivityClass and stepCount. If a feature is unavailable, it returns an NA value for that feature. The feature physicalActivityMets is a way to quantify the intensity of physical activity. It is determined by comparing your working metabolic rate to your resting metabolic rate [33]. The feature physicalActivityClass is a function in which a user can adjust the amount and duration of their exercises in a given week. Adjusting this setting leads to improved accuracy of burnt calories[34].
The script identifies which dates have available data for stepCount, HR, or hrvSdnn and selects those with consecutive days. For each consecutive day, data between 3 PM on that day and 3 PM on the next day is loaded. If no data is available, no sleep data is returned for that day. Otherwise, the median of each feature is calculated in intervals of 3 minutes.
If more than 16 hours of HR data are available, a 30-minute smoothed HR signal is created and interpolated using PCHIP (Piecewise Cubic Hermite Interpolating Polynomial). High-frequency noise is removed using a low-pass filter (N=1, Wn=1/3600, btype='lowpass', fs=1/180), and the first and last 30 minutes are excluded to avoid edge effects. Peaks in the stepCount signal are identified using the function scipy.signal.find_peaks. If no peaks are found, the sleep data is returned empty. If peaks are identified, each is examined:

- If HR data after the peak contains NA values, the peak is shifted to the next valid time point (max 2 hours).
- If the HR before the next peak contains NA values, the peak is shifted backward similarly.

If possible, a 2-hour HR subset is created after the first peak (subset_start) and before the last peak (subset_end). If the slope of subset_start is negative, subset_end is positive, and the interval includes any time between 00:00 and 06:00, the period is labeled as sleep. If this condition is not met, the algorithm proceeds to the next possibility. If these subsets cannot be created (e.g., due to mising data), a period is labeled as sleep if the time window includes 00:00-06:00.

In all cases, any periods with stepCount higher than zero for longer than 3 minutes are labeled as awake. The earliest and latest sleep labels define the toBedTime and getUpTime, from which the inBedDuration is calculated.
If less than 16 hours of HR data is present, the algorithm checks for at least 16 hours of stepCount data. If this is also missing, the sleep data is returned empty. If enough stepCount data is present, the same logic is applied. Peaks are identified, and those that occur between 00:00 and 06:00 are labeled as sleep. Awake periods are again defined by stepCount being higher than zero for over 3 minutes, and sleep duration is determined by the first and last sleep time points.

Algorithm developed for this study

The algorithm developed for this study is based on a previously implemented algorithm by imec NL. If no sleep data is returned through this primary method, a fallback procedure is initiated. This fallback relies on a second algorithm, also developed by imec NL, which reads the daily questionnaire responses. These responses include the participant's reported bedtime and wake-up time, used to calculate in-bed duration, as well as reported sleep quality. Additionally, this algorithm reads the participant's reported main sleep duration goals. With these features our algorithm can gives scores to these various questions.

The measured sleep duration is compared to the participant's main goal. Points are assigned as follows:

- Within ±15 minutes → 5 points
- Within ±60 minutes → 4 points
- Within ±120 minutes → 3 points
- Within ±180 minutes → 2 points
- More than ±180 minutes deviation → 1 point

The sleep quality is derived from four questionnaire items, each contributing a different number of points. Since overall perceived sleep quality was one of the primary goals for improvement, it was assigned the highest possible scores in the questionnaire. For the questions where points are subtracted, a maximum total of -2 points is spread across three questions. The yes/no question contributes up to a 0.4-point reduction, while the other two questions, with five possibilities, each have a maximum deduction of 0.8 points:

1. Overall perceived sleep quality (rated 0–100):
    - 0–19 → 1 point
    - 20–39 → 2 points
    - 40–59 → 3 points
    - 60–79 → 4 points
    - 80–100 → 5 points
2. Difficulty falling asleep: If the participant reported taking longer than 30 minutes to fall asleep, 0.4 points were subtracted.
3. Night-time awakenings (e.g., toilet visits):
    - 0 visits → 0 points deducted

- 1 visit → -0.2
- 2 visits → -0.4
- 3 visits → -0.6
- More than 3 visits → -0.8

4. Daytime functioning: If the participant reported reduced motivation or enthusiasm during daily activities:
   - No problem → 0 points deducted
   - Slight problem → -0.2
   - Somewhat of a problem → -0.4
   - Quite a problem → -0.6
   - Very problematic → -0.8

The final quality score is calculated by subtracting the penalties from the overall perceived quality score. Since the maximum total deduction is -2 points, the lowest possible final quality score is -1.
The total score for each night is calculated as the average of the sleep duration and quality scores:

$$\text{Final score} = \frac{\text{Duration score} + \text{Quality score}}{2}$$

Based on this final score, a pre-generated recommendation is selected and subsequently personalized for each participant. All extracted features and calculated scores per participant are exported to an Excel file for further analysis.

Case Example: Gijs' Night of Sleep

Gijs is a 68-year-old participant who set his main sleep goal at 7.5 hours (450 minutes). On one of the study nights, the measured sleep duration from the Garmin vívosmart® 5 was 6 hours and 40 minutes (400 minutes), resulting in a deviation of 50 minutes.

$$\text{Deviation} = |400 - 450| = 50 \text{ minutes}$$

This falls within the ± 60 minute range, and therefore Gijs receives:

$$\text{Duration score} = 4 \text{ points}$$

From the daily questionnaire, Gijs reported:

- Overall perceived sleep quality: 65/100 → 4 points
- Difficulty falling asleep: Yes → -0.4 points
- Toilet visits: 2 → -0.4 points
- Daytime functioning: "Somewhat of a problem" → -0.4 points

Total penalties:

$$\text{Penalties} = -0.4 - 0.4 - 0.4 = -1.2$$

$$\text{Quality score} = 4 - 1.2 = 2.8$$

$$\text{Final score} = \frac{\text{Duration score} + \text{Quality score}}{2} = \frac{4 + 2.8}{2} = 3.4$$

Based on this final score, a pre-generated recommendation is selected and personalized for Gijs, addressing issues such as difficulty falling asleep and night-time awakenings.

3.2.3 RULE BASED GUIDELINES

The rule-based guidelines are structured as a step-by-step flow diagram that the algorithm follows to determine appropriate recommendations. It begins by checking whether the total score has reached its maximum. If it has, the individual is considered to be doing well and no changes are recommended. If not, the algorithm evaluates the sleep score and assigns a corresponding recommendation. The self-report score is also assessed, resulting in an additional recommendation. These two recommendations, one based on the sleep score and one based on the self-report, are then combined into a single personalized recommendation for the individual.

3.2.4 REFLECTION

The final part of phase two involves the first reflection moment. In this section, the individual is asked to evaluate their progress so far. They begin by defining their sleep goal using the SMART framework (Specific, Measurable, Achievable, Relevant, and Time-based). The SMART principle can be used to create accessible and understandable goals to increase engagement and performance[35]. Next, they identify which specific aspects of their sleep they want to improve. Based on the previously calculated scores, they then receive personalized recommendations. These recommendations help guide the formulation of both short-term and long-term goals.

3.3 PHASE III - ADJUSTING AND REFLECTING

In the third and final phase, individuals have the opportunity to adjust their main goal based on the reflection of phase two. After some time has passed, they are asked to reflect once again, this time on their recent progress, their previous reflection, and whether they wish to adjust their main goal (again). As part of this reflection, they are asked whether they were able to follow through on the short-term goal they set during the previous phase, and to consider what factors contributed to their success or made it difficult to achieve. Finally, they are given the option to adjust their short- and/or long-term goals based on their experience and current needs.

4 METHODOLOGY

4.1 ETHICAL CONSIDERATION

This study was approved by the Natural Sciences & Engineering Sciences Ethics Committee of the University of Twente, reference number 250032, prior to data collection. All participants received informed consent before participating in the study and the research was conducted according to the application.

4.2 STUDY DESIGN

This study was a fifteen-day field-based measurement, with all assessments conducted outside of a laboratory setting. Participants were instructed to wear a Garmin vívosmart® 5 for the full duration of the study. At the start, they completed several questionnaires, including the Pittsburgh Sleep Quality Index (PSQI [36], which will be explained in Chapter 4.4.3), and set a personal sleep duration goal. On a daily basis, the participants had to complete the daily questionnaire in the OnePlanet Research mobile application. If the participant had an iPhone, they would have to manually synchronize, twice a day, the phone with the watch by clicking the manual synchronization button in the application. After five days, the algorithm evaluated their sleep duration and sleep quality scores and provided personalized recommendations. The participants were then asked to reflect on their initial goal and the recommendations received. Sleep tracking continued throughout the study, and on days ten and fifteen, participants were asked again to reflect on their progress, goals, and the updated recommendations. At the end of the study, participants took part in a focus group session to evaluate and discuss their experiences.

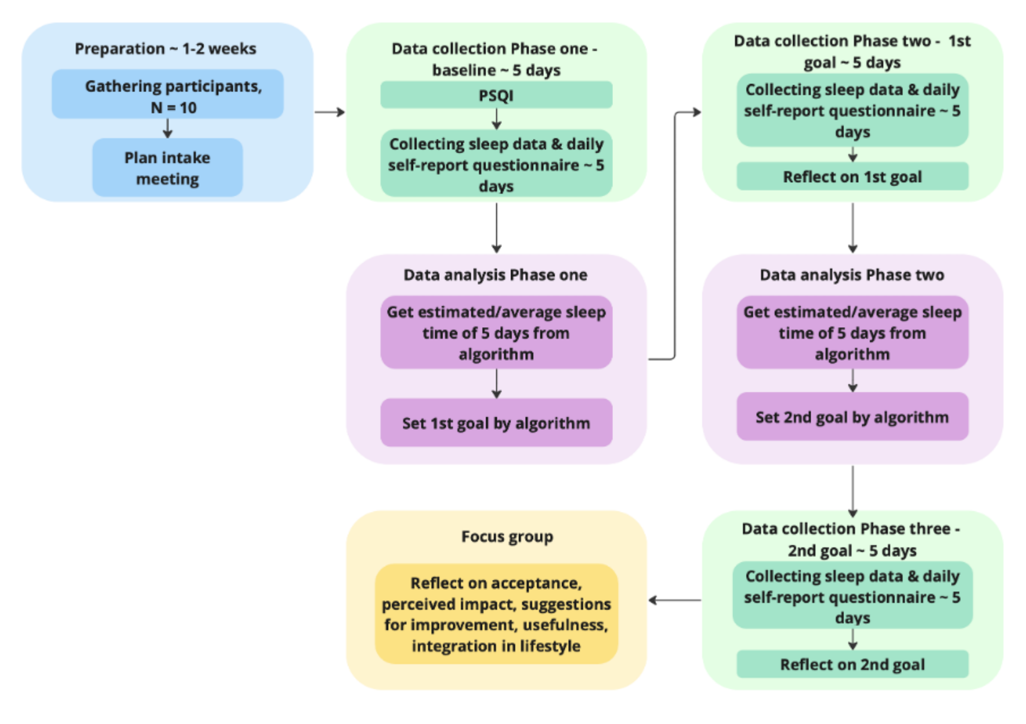


Figure 3: Study design

4.3 PARTICIPANTS

The inclusion criteria for the study were as follows: (1) male or female adults aged 60 years or older, (2) no known sleep disorders (e.g., sleep apnea or insomnia), (3) ownership of a mobile phone, (4) willingness to wear a Garmin watch throughout the study period, and (5) the ability to read and write in Dutch. Based on these criteria, eleven older adults were enrolled in the study, six males and five females. Two participants are members of a rowing association, while seven are members of an ice skating association. The final two members are not part of an association, but do exercise. Participants were recruited by sending an invitation email to the boards of these associations, who were asked to forward the invitation to their members. Existing connections within the associations were also used for recruitment. All participants were invited to an intake meeting, during which the study procedures, the Garmin device, and the mobile application were explained. They also received and signed an informed consent form prior to participation.

4.4 MATERIALS

4.4.1 WEARABLE ACTIVITY TRACKERS

An alternative method for measuring and monitoring sleep patterns is the use of wearable devices, such as smartwatches. In this study, sleep patterns were assessed using data from heart rate (HR), heart rate variability (HRV), and step count, collected via a wearable device, like the Garmin vívosmart® 5. A previous study compared HRV measurements derived from photoplethysmographic (PPG) signals of commercially available smartwatches, such as the Garmin vivoactive® 4, with those obtained from their gold standard electrocardiogram (ECG). The findings indicated that HRV values obtained through PPG closely resembled those measured via ECG recordings [37]. Given that the Garmin vivoactive® 4 was released three years prior to the Garmin vívosmart® 5 used in the current study, it is reasonable to assume that the PPG sensors in the vívosmart® 5 are at least comparable, if not improved. The vívosmart® 5 records

HR every 30 seconds and step count every minute. Every 15 minutes, this data is uploaded to the phone, as long as the phone is nearby and connected to the watch, and then uploaded to the imec NL cloud.

4.4.2 ONEPLANET RESEARCH MOBILE APPLICATION

Normally, the Garmin vívosmart® 5 connects to the Garmin Connect application, which processes the data and presents users with visual insights and graphs. However, for the purpose of this study, access to raw data was required—something not available through Garmin Connect. Therefore, the OnePlanet Research mobile application was used instead. This custom application connects to the Garmin vívosmart® 5 via Garmin Health SDK and sents the collected data to a secure server. On Android devices, the watch automatically synchronizes with the app every fifteen minutes. However, on iOS devices synchronization must be done manually by the user to ensure the data is sent to the server. Once received, the data can be extracted from the server and used by the algorithm to generate personalized sleep recommendations for each participant.

4.4.3 QUESTIONNAIRES

To improve both the quality and the duration of sleep, it is essential to measure and monitor sleep patterns. An effective method is the PSQI questionnaire. In 1988, Buysse et al. developed the PSQI questionnaire, a self-rated questionnaire designed to evaluate sleep quality and disturbances over a one-month period. The PSQI questionnaire includes 19 self-rated questions along with five questions rated by a bedpartner or roommate. These 19 questions are divided into seven components, each rated on a scale of 0-3 with equal weighting. The component scores are then combined to produce a global PSQI score ranging from 0 to 21, where higher scores reflect poorer sleep quality[36].
The PSQI questionnaire was designed to serve multiple purposes in the analysis of sleep health. Its primary goal was to establish a standardized, reliable, and valid tool for measuring sleep quality. In addition, it was designed to distinguish between people who sleep well and those who sleep worse. The ease of use was also a key consideration, ensuring that both the individuals who completed the questionnaire and the clinicians or researchers who interpret the results could do so without difficulty. Finally, the PSQI questionnaire was structured as a brief yet comprehensive tool to help identify various sleep disturbances that can affect overall sleep quality[36].

A daily questionnaire was also included in the study. In this questionnaire, participants rated the quality of their previous night's sleep on a scale from 0 to 10, where 0 indicated very poor sleep quality and 10 indicated very good sleep quality. In addition to this rating, selected items of the PSQI questionnaire were included to assess aspects such as sleep latency, sleep disturbances, and daytime dysfunction.

Two custom reflection questionnaires were developed to support participants in evaluating their sleep management and goals. These personalized questionnaires were created using Microsoft Forms and included both open-ended and structured questions focused on goal setting, perceived progress, and the intention to adjust sleep patterns.
In the first reflection, participants were asked to define a SMART sleep goal and specify which aspects of their sleep they wanted to improve. Later reflections assessed and reviewed progress toward the initial short- and long-term goals. These reflections were integrated into the intervention to promote self-awareness and personalized recommendations.

Finally, multiple focus group sessions were organized to evaluate the study from the participants' perspective. Small groups of participants came together under the guidance of the researchers to reflect on their experiences. During these sessions, participants discussed various aspects of the study, including its overall usefulness, how well it fit into their daily routines, and the perceived impact on their sleep behavior. They also provided feedback on what elements worked well, what could be improved, and how the intervention might be better integrated into everyday life. In addition, the sessions explored participants' acceptance of the technology and their suggestions for future improvements, offering valuable insights for further development and refinement of the approach. To conclude the focus group, participants were shown images of various applications in which the sleep domain is presented, such as Garmin, Fitbit, Apple Health, and others.

4.5 DATA ANALYSIS

Data analyis in this study involves several steps. First, the PSQI scores are analyzed to assess whether participants can be classified as good sleepers. This classification can be used as a reference point to validate other outcomes.

In addition to the PSQI, the measurement data contains various variables that must be processed. To provide a clear overview, a table is created to summarize trends and averages across participants. This visual representation can help identify patterns and support further interpretation.

By examining changes in time difference between participants' average sleep duration and their individual sleep goals, as well as overall sleep quality across days 1-5, 6-10, and 11-15, we can begin to identify potential improvements. Furthermore, the number of toilet visits and the number of nights in which participants took more than 30 minutes to fall asleep may also reflect sleep-related changes, which ultimately contribute to better cognitive health.

It is important to approach the measurement data with caution. Since the algorithm is newly developed, it's accuracy needs to be evaluated. One approach is to compare the sleep durations generated by the algorithm (based on sensor data) to the reported durations derived from participants' bedtimes and wake-up times. To support this comparison, the total number of recorded nights per participant is examined, along with the number of fallback durations used. For the nights with actual sensor-based measurements, it is then checked how many match the reported sleep duration within a range of $\pm$45 minutes and $\pm$60 minutes. This comparison can be summarized in a table or visualized per participant in a graph to assess reliability and guide further improvements of the algorithm.

Lastly, data from the focus group will be analyzed using deductive thematic analysis. By grouping responses into themes, the analysis can highlight shared experiences, key concerns and their perspectives on several categories. With the help of Nikita Sharma, eight themes were developed, focusing on sleep goals, recommendations, the OnePlanet Research application and the watch itself, sleep-related education, behavioral impact, participants' preferences, and examples of other applications.

5 RESULTS

5.1 DEMOGRAPHICS AND PSQI DATA

Demographic and PSQI data are shown in 1. The participants had a mean age of 66.5 (range: 61 - 83 years) and there was a slight majority of men (54.5%). None of the subjects reported a diagnosed sleep disorder, as this was one of the inclusion criteria. The sample group included individuals with various educational levels, which contributed to a good variety. Of the 11 participants, three reported previous experience with measuring sleep and using a corresponding mobile application. PSQI total scores ranged from 2 to 6 ($\leq$ 5 'Good sleep quality', $>$5 'Poor sleep quality' [36]), with 82% of the participants (9 out of 11) reporting good sleep quality.

Table 1: Participant Demographics, App Experience, and PSQI Score

Participant	Age	Gender	Education Level	Prior Sleep App Experience	PSQI Score
p020	62	Male	WO	Yes	5
p021	68	Female	HBO	No	2
p022	62	Male	MBO	No	5
p023	64	Female	MBO	No	6
p024	67	Male	MBO	Yes	3
p025	61	Male	WO	No	4
p026	71	Male	WO	No	4
p027	63	Female	MBO	Yes	3
p028	68	Female	MBO	No	2
p029	65	Female	HBO	No	2
p030	83	Male	HBO	No	6

5.2 ADAPTIVE GOAL-SETTING: QUANTITATIVE RESULTS (MEASUREMENTS OVER 15 DAYS)

Table 2: Study Engagement per Participant

Participant	**Completed 15 Days**	**Completed Questionnaire Days 5/10/15**	**Focus Group**
p020	Yes	Yes	Yes
p021	Yes	Yes	Yes
p022	Yes	Yes	Yes
p023	Yes	Yes	Yes
p024	Yes	No (2/3)	Yes
p025	Yes	Yes	Yes
p026	No (5/15)	No (1/3)	Yes
p027	Yes	Yes	Yes
p028	Yes	Yes	Yes
p029	No (9/15)	No (2/3)	No
p030	Yes	No (0/3)	Yes

During the 15-day measurement period, a few participants did not complete all 15 days or missed one or more of the questionnaires provided on days 5, 10, and 15, as shown in Table 2. This also includes information on the attendance at the focus group sessions.
The participants' main sleep duration goals were collected during the intake meeting. The first five days of data serve as a baseline. Throughout the study, changes are expected in various domains, such as the time difference between the participants' average sleep duration and their sleep duration goals, overall sleep quality, number of toilet visits, and number of nights when the participant did not fall asleep within 30 minutes. These data could give the best overview of potential improvements in sleep habits, which ultimately leads to better cognitive health.

Table 3: Sleep Metrics Overview Per Participant Across Three Periods

Participant	**Main Sleep Goal (h)**	**Average Time Diff (min)**			**Average Quality**			**No. of Toilet Visits**			**No. of 30+ min Awake**		
		Day 1-5	Day 6-10	Day 11-15	Day 1-5	Day 6-10	Day 11-15	Day 1-5	Day 6-10	Day 11-15	Day 1-5	Day 6-10	Day 11-15
p020	8	25	21	88	72.2	76.25	80	3	4	2	1	0	0
p021	8	24	45	22	71.4	87	67.5	0	0	1	1	0	1
p022	7.5	84	36	28	46.6	58.4	62.3	7	7	3	2	0	0
p023	8	60	2	15	57.4	44.4	52.2	5	9	5	1	3	2
p024	8	108	111	0	64.7	67.5	84	3	4	0	0	1	0
p025	7.5	28	33	118	69.2	72.6	77.7	0	1	1	1	0	0
p026	8	80	-	-	78.2	-	-	1	-	-	1	-	-
p027	8	39	95	66	76.8	79.4	81	5	4	3	1	2	1
p028	8.25	48	21	105	73	74	80.75	11	6	6	0	2	1
p029	8	0	20	-	54.4	75	-	4	3	-	1	0	-
p030	6	99	48	72	81.8	71.6	84	5	6	5	0	0	0
Average		54.1	43.2	48.4	67.8	70.6	74.4	4.0	4.4	2.9	0.82	0.8	0.56

When examining Table 3, it is difficult to identify clear improvements, as the data shows fluctuation across all participants. However, participants p020, p022, and p024 appear to show improvements across all domains shown in Table 3. Others, such as p023, p025, p027, p028 and p029, show improvements in some, but not all, domains. While these trends are promising it is hard to directly relate them to participants' PSQI scores. The PSQI primarily reflects subjective sleep quality and environmental factors, which may not always align with short-term, sensor-based changes in sleep behavior.

As shown in the time difference values for day 5 in Table 3, there are some outliers. In some cases, the measured sleep durations were unusually low and did not correspond with the reported bedtimes and wake-up times. This occurs in participant p024, as illustrated in Figure 4. The figure displays the metrics of that night's sleep, raw HR (red, top), filtered HR (blue), step count (black), and detected sleep (red, bottom).
At approximately 14.5 hours after 3 PM (i.e., 05:30AM) a high step count is recorded, which causes the sleep measurement to reset.

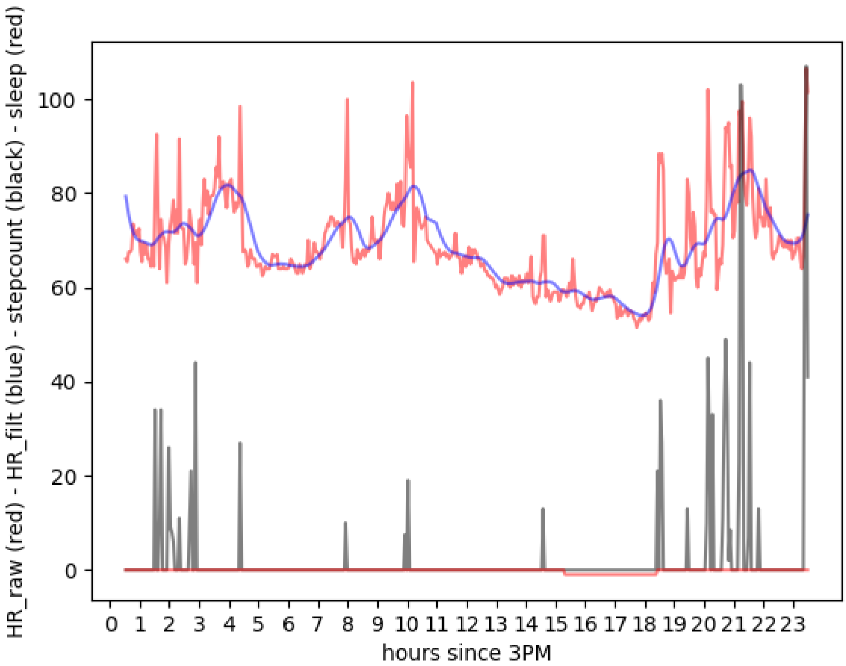
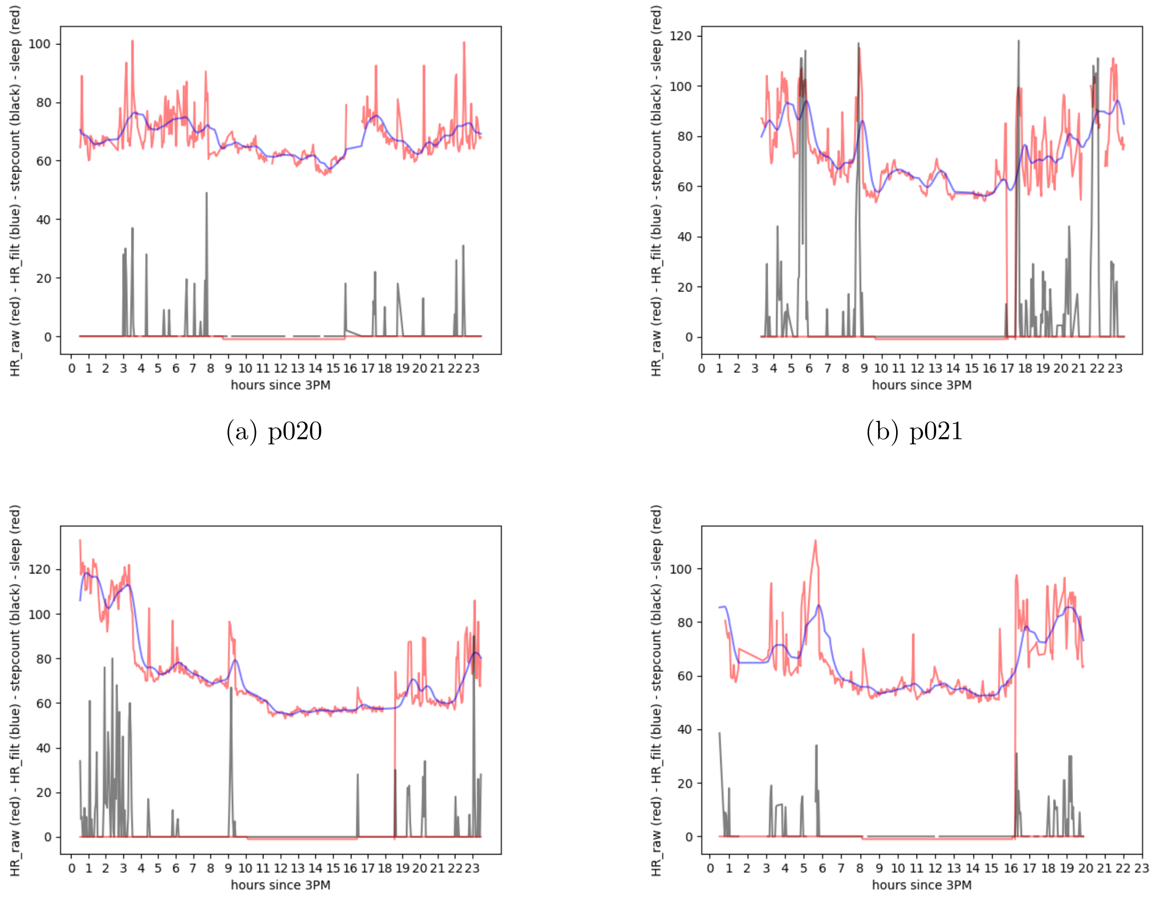


Figure 4: Misclassified Sleep in p024 Caused by Late Movement

Although some nights showed misclassified sleep, most measurements were reliable. The following graphs in Figure 5 show examples where sleep was measured accurately and correspond to the reported bedtime and wake-up time.



(a) p020 (b) p021 (c) p025 (d) p028

Figure 5: Overview of four sleep recordings (p020, p021, p025 & p028)

The subjects were provided with a questionnaire on day 5. In this questionnaire, they were asked to formulate their SMART sleep goal, and subsequently given their recommendation based on their data of the first five days.
During the formulation of their SMART goals, multiple subjects expressed similar intentions to improve their sleep quality, for example by achieving more continuous and uninterrupted sleep throughout the night.
Participants p020, p022 and p028 aimed to improve sleep continuity, with p020 specifically mentioning a desire for deeper sleep and eight hours of sleep. To achieve these goals, reducing fluid intake in the evening to minimize the number of times being awake during the night was a common strategy mentioned by p020, p023 and p028.
Caffeine reduction in the evening was also an approach mentioned by multiple participants, both p020 and p023 reported avoiding coffee in the afternoon and evening. In addition, p023 aimed to limit fluid intake during the evening but compensate by increasing hydration earlier during the day. P020 stated that maintaining consistent bedtimes was a goal, while p028 expressed the desire to reduce screen time in the evening and replace it with reading a book, an activity also planned by p020 as part of their bedtime routine.
While these participants formulated relatively concrete goals, other participants kept their goals more general, simply expressing a desire to feel more rested or to improve their sleep without specification.

As mentioned above, changes in various sleep domains can be interpreted as potential improvements in sleep habits. As shown in the time difference values for day 5 and day 10 in Table 3,

the average time difference decreased, with 50% of participants (5 out of 10; p026 dropped out after day 5) showing an improvement in sleep duration. None of these subjects had misclassified sleep durations resulting in unusually low values. However, p020 did record one unusually high sleep duration. When using the reported bedtime and wake-up time instead, the resulting time difference is 27 minutes. This means that 40% of the subjects improved their sleep duration.
An improvement in reported sleep quality was observed in 80% of the subjects (8 out of 10). On the other hand, the number of average toilet visits increased slightly over time, with only 30% of the subjects (3 out of 10) managing to reduce toilet visits at night. Lastly, 50% of the subjects (5 out of 10) consistently fell asleep within 30 minutes.

On days 10 and 15, the participants were invited to reflect on their short-term goal. Had they succeeded, and did the recommendations help motivate them? Or, if they did not succeed, what were the reasons? On day 10, 90% of the participants (9 out of 10) completed the questionnaire. Of these, 78% (7 out of 9) reported achieving their short term goal and indicated that the recommendations had motivated them. The two participants who did not achieve their short-term goal reported lifestyle-related or undisclosed limitations. At the end of the questionnaire, participants were given the option to adjust their short- or long-term goal. However, all chose to continue with their previous set goals.
On day 15, 70% of the participants (7 out of 10) answered the questionnaire. Of these, six participants managed to successfully achieve their short-term goal. Once again, all participants whishes to continue with their previous set goals.

As described previously, some of the participants improved their sleep habits after the first five days. But did this improvement continue, and what do the overall results show?
Five days after receiving the first recommendation, 40% of the participants showed an improvement in sleep duration. By day 15, five days after the second recommendation, this has increased slightly to 44% (4 out of 9 participants, note p029 unintentionally stopped uploading data due to being unable to open the application). However, once again, a misclassified sleep duration affected the results. After correction, the actual number of participants who improved increases to 55% (5 out of 9). An improvement in reported sleep quality was observed at 88% of the participants (8 out of 9), with an average total quality score of 74.4. In terms of nighttime disturbances, 66% of the participants (6 out of 9) reduced their number of toilet visits. Finally, for 88% of the participants (8 out of 9), the number of nights in which they failed to fall asleep within 30 minutes either decreased or was already at zero.

Misclassified sleep durations were mentioned on several occasions, but how many of these cases can truly be considered misclassifications? To analyze this, several categories were defined per participant: the total number of measurements, the number of times a fallback duration was used, and the number of measurements where the reported bedtime and wake-up time correspond to the measured duration within ± 45 minutes and ± 60 minutes. It is important to note that only nights with actual measurements were considered for the matched categories, fallback durations were excluded. These results are presented in Table 4.

Table 4: Sleep Duration Summary per Participant

Participant	Total Nights Recorded	Fallback Used	Total Actual Measurements	Matched ±45 min	Matched ±60 min
p020	14	4	10	1	3
p021	14	1	13	5	6
p022	13	9	4	0	0
p023	15	1	14	7	8
p024	11	2	9	2	2
p025	14	3	11	5	7
p026	5	3	2	2	2
p027	13	4	9	5	5
p028	14	4	10	5	6
p029	9	8	1	1	1
p030	13	3	10	4	5
Total	**135**	**42**	**93**	**37**	**45**

A total of 135 sleep durations were collected, which is less than the expected 165 (11 participants over 15 nights). This is because of various reasons: participant p026 voluntarily dropped out, while p029 involuntary stopped uploading data due to a malfunctioning application. Participants p022 and p024, both iOS users, experienced synchronization issues between their watch and the application, resulting in missing nights. Finally, for participants with 13 or 14 nights, the last night's data was not uploaded successfully because access to the application was prematurely revoked on the final day.

Of the 135 total sleep measurements, 42 were calculated via the fallback method. This occurred when the Garmin JSON files lacked sufficient data for the algorithm to compute the sleep duration. Either due to incomplete sensor input or no data at all, as was the case for p022's first 9 days. In such cases, the fallback method used the reported bedtime and wake-up time instead.

This leaves 93 sleep durations based on actual measurements. Of these, only 39.8% (37 out of 93) matched the reported sleep duration within ± 45 minutes. When the threshold is extended to ± 60 minutes, 48.4% (45 out of 93) can be considered accurate.

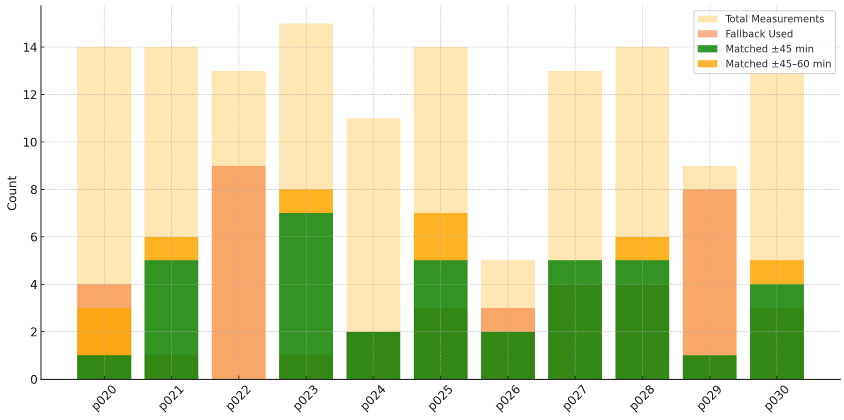


Figure 6: Sleep Measurements (Matched ±45 and ±45-60 min)

5.3 ADAPTIVE GOAL-SETTING: QUALITATIVE RESULTS (RESULTS OF FOCUS GROUP

This analysis will be grouped into several themes. This way, shared experiences, perspectives on specific categories and key concerns can be highlighted. A total of 10 participants (91%) participated in the focus group sessions. There were three sessions, with two times four participants and one session where there were two participants present.

Theme 1: Opinion on sleep goals

At the beginning of the focus group, participants were asked about their opinion on sleep goals. 70% of the participants mentioned that they did not have a specific goal, since they felt they were already sleeping well or had other reasons. For example for p021 and p030:

> *I don't think I need the tools. I don't need someone telling me: you didn't reach your goals, or you have to do this or that. No, that's not for me.*
>
> [p021]

> *Well, look. The question is whether you have a goal. I didn't have a goal at all. I had no goal to sleep longer or shorter, I just didn't have a goal.*
> *Look, I'm just participating because I want to help determine whether sleep is or isn't related to dementia or Parkinson's. That's what it's about. That was my goal. That's why I joined.*
>
> [p030]

The other 30% did report to be working on their goals. This varied from drinking less in the evenings, to reading books before bed, all to optimize the sleep conditions.
In addition to this theme, participants were asked which they found more reliable, the watch and its data, or their own perception on sleep quality. For 40%, this question remained unanswered, for 20% the watch is a significant improvement in tracking their sleep, since it knows way more then they do. But 30% classified the watch not trustworthy. Their own opinion is best, regardless of what the watch or application says.

> *If I feel that I've been lying awake for a long time and the watch says otherwise, then I don't trust the watch. My own sense of having slept poorly is more reliable.*
>
> [p028]

Theme 2: What kind of sleep goal is desired?

In the next question, participants were asked how they would prefer to set sleep related goals, whether in terms of quality, duration, long-term or short-term objectives or another domain. 60% of participants were unable to provide a specific response. The other 40% varies from setting a fixed bedtime, to avoiding the urge to fall asleep in a chair during the day, or simply improving sleep quality.
One participant made a valuable note about the use of the word 'goal', questioning whether it was the appropriate term in this context:

> *Maybe it's also a little bit semantics, because the word goal implies that you're able to organize yourself around it with the resources and to put the effort in. Sleeping in itself is not something that I can force. The timing is about pretty much what I can force. Preparation before, so, no screen time, no exciting things before bed, is something that helps me fall asleep quickly. But the sleep itself... I go down on my*

pillow and I wake up when I wake up. Pretty much. Which is normally around the same time. There's nothing I can organize, I'm sleeping. So, setting a sleep goal of 6 or 6 and a half or 7 or 7 and a half or eight, it's pretty much out of my control. Unless you've set an alarm clock that cuts off your sleep at 7:00 in the morning. But that's not really what I think you entice with setting goals. It would be better to do something about conditions instead of goals. Because this is all conditions. The condition to have a good sleep is to be healthy and not eat too late, to not do screen time before. So all those conditions you can manage. Goals, however...

[p025]

In addition, over half of the participants (70%) agreed with the statement that setting specific goals for sleep is nearly impossible. Unlike exercise, where progress can be trained, sleep is less directly controllable. Participants found it more useful to focus on aspects they can influence, such as bedtime routines, bedroom conditions, and sleep-related habits, rather than setting strict performance goals for sleep itself.

Theme 3: Recommendations

In the next section participants were asked for their opinion on the recommendations provided on days 5, 10 and 15. 60% of participants were expecting more numeric values and graphs. This would have encouraged them more to work their goals:

For me, it's important to understand what the recommendations and results are based on. And if I feel that I understand myself better than the app does, then I'm less likely to accept its advice.

[p026]

In addition, 30% had the idea that the recommendations were common formulated, it was all very logical:

Sometimes I felt that the feedback I received was just based on what I had filled in. Like, okay, great. For example, if I said that I had something to drink, then the feedback would be: you shouldn't drink so much.

[p020]

Lastly, participant p022 described the recommendations as too gentle. P022 expressed a preference for more direct and firm advice, clearly stating what should be changed, rather than beating around the bush.

The follow-up question on the recommendations focused on 5-day feedback interval. Half of the participants did not provide a specific or useful response. Among the remaining participants, opinions varied, one found the 5-day interval sufficient, two preferred a weekly interval, another would have liked daily feedback, and finally:

Well, I'd like a combination of daily feedback on your sleep, so you can see how things are going, and then, every two weeks, higher-level feedback that summarizes everything and gives advice. I'm a data person, so I enjoy seeing the details, but I'd also want to see trends and get feedback on that. Let's say a biweekly update that includes progress on your goals and how your sleep behavior over the past fourteen days has contributed, or not contributed, to achieving those goals. And maybe that feedback could even suggest whether your current goal is suitable, or whether you should consider a different one.

[p020]

Theme 4: Experience using the watch and the application

The OnePlanet Research mobile application, as well as the use of Garmin vívosmart® 5, are relatively new. Since this is a feasibility study prior to a follow-up study, it is important to identify any limitations, irritations, or possibilities for improvement in either the application or the watch.

In this study, 4 out of 11 participants used an iPhone to connect the Garmin device to the application. Manual synchronization was required for these users, which proved to be the hardest part. For 3 out of 4 iPhone users, this manual synchronization process became an obstacle, which led to one participant dropping out. The other two participants continued with the study, but had their irritations. Specifically, the app does not provide a progress bar during manual synchronizations. A pop-up appears and disappears when the synchronization is finished, but because this synchronization could take up to an hour, or just fail, this lack of feedback was undesired.

Additionally, the application's home screen displayed the time of the last synchronization, but this was often outdated, even immediately after manual syncing. This caused confusion and stress among participants, who were unsure whether the issue was due to user error and what actions, if any, they should take.

Besides that, two participants reported skin irritation of the wristband, which also occurred to some researchers prior to the start of the study. In both cases, the irritation resolved after switching the watch to the other wrist.

Additionally, it was not clear whether the questionnaires were completed or still had to be answered, as completed questionnaires did not change color, show a timestamp when finished, or disappear from the list. Furthermore, a few remarks were made on certain questions, being poorly formulated.

Finally, one participant reported that the watch screen would light up during the middle of the night due to movement, which disrupted their sleep. They resolved this by wearing a wrist sweatband over the device to block the light.

Theme 5: Provided education on sleep

During the intake meeting, participants received educational information about how sleep works and on factors influence it. However, three participants reported that they did not recall this information and had not read it. One participant found the information sufficient, and another reported it to be very useful and would have liked more detailed information. Those who did not recall the provided education suggested that it should be provided (again) as part of the first recommendation.

Theme 6: Impact on behavior

In this next section the participants are asked whether the recommendations and measurements had impact on their behavior, did it alter their sleep habits or not? Two participants were convinced the recommendations could alter someones habits, if you put the work in it:

> *For me, I think I have to answer this more hypothetically. I didn't really have any specific goals, so setting goals wasn't really a result of receiving feedback. But hypothetically, I do see the potential: when you combine actual measurements with your own perception, you can receive quite personalized advice. Like, "This is what you think, and this is what we observe." Then you can try adjusting things—conditions like stress, light, noise, or other factors—and see how that affects your sleep. That's the kind of insight you could get from an individual Garmin device, but also through the collective app. The idea is that personal sleep outcomes might improve by making small changes. Like going to bed a bit earlier, for example.*

[p025]

On the other hand, one participant expressed doubts about the effectiveness of the recommendations, noting that sleep habits have developed over many years. Finally, two participants showed their interest in changing their behavior if necessary for their health, but noted that they currently do not feel the urgency to do so.

Theme 7: Preference in sort of app

In this section, participants were asked wether they would prefer an application that automatically adjusts their sleep goals, one that allows them to set their own goals based on suggestions from the application, or one that lets them set goals based on their own experiences.
Two participants indicated a preference for an application that automatically adapts sleep goals. However, these were not the same participants who trusted the application and watch to determine their sleep quality. But p022, who previously described the recommendations as too gentle, stated a preference for the app to take full control, as this would result in more direct and firm advice.

> *I'd be interested in an app that tells me what's best for me, based on both the data it collects and my own perception. For example, it could suggest, "It's best for you to go to bed at this time." Whether I follow that advice or not is up to me, the app doesn't control me. Maybe it says I should go to bed by midnight, but I stay up until 2:00 AM because the movie was interesting. I'll feel the consequences of that choice the next day by being more tired. For me, optimal sleep might mean going to bed around 11:30 or 12:00 and waking up around 7:00 or 7:30. But maybe that changes, maybe it shifts by two hours depending on the season, like winter versus summer. I don't know, it could depend on sunlight. What I'd really like is an app that combines personal data with general knowledge about healthy sleep, and tailors recommendations to me.*
>
> [p025]

More than half of the participants (70%) preferred to set their own sleep goals, based on their own experience and the recommendations of the application. While most of the participants gave similar reasons for this preference, participant p020 provided a unique motivation:

> *In my own research, my motto is: "Machine control, human command." That's why I believe in setting goals myself, based on suggestions from the app. The machine measures everything, it knows much more than I do, it can detect things I can't. And based on that, it gives suggestions. But in the end, I'm the one who decides what to do.*
>
> [p020]

Theme 8: Examples of other applications

In the final part of the focus group, participants reviewed example screenshots from various health applications, showing different ways sleep data can be visualized. They commented on features such as graphs of sleep stages and duration, sleep scores and long-term overviews (weekly, monthly or yearly). Based on these examples, participants were then asked to name three features they would most like to see implemented in the future versions of the application. This section is answered by 9 participants, since one participant was unable to attend the full focus group. Of these, participant p030 stated that he had no idea or preference in either one of the features.

It's really hard to answer. I don't know what I find important in the different apps. I don't want to brush it off by saying I don't want anything or something like that—but I'm just not sure.

[p030]

Of the remaining eight participants, seven participants expressed their preference for graphs showing the different sleep stages (REM, light, deep, and awake), along with a textual summary of this information, such as shown in Figure 7 from the Garmin Connect App.

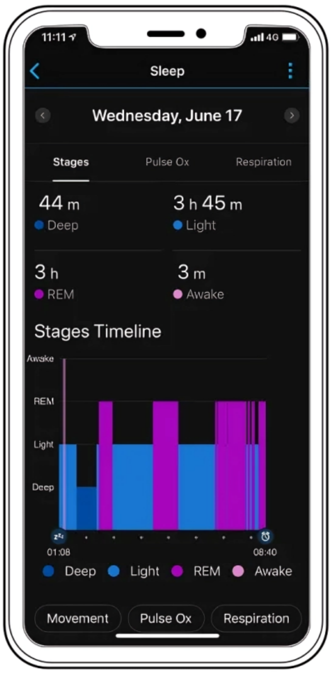


Figure 7: Example of Sleep Domain in Garmin Connect App [20]

Additionally, five participants expressed their preference for a white interface layout, similar to that of the Fitbit application. Some participants also mentioned preferences in features such as HR, (continuity in) breathing, and sleep scores, especially when combined with self-reported sleep quality, as in this study.

6 DISCUSSION

This feasibility study aimed to assess the practical implementation of an adaptive goal-setting feature for sleep in a mobile application designed for older adults. The study investigated both technical and user-experience aspects of the application, as well as the implementation of the adaptive goal-setting feature in the sleep domain. Overall, the findings suggest that the study procedures were mostly feasible, and that the adaptive goal-setting feature holds promise for promoting healthier sleep behavior. However, the results also revealed several challenges that must be resolved before proceeding to a follow-up study. The focus group sessions made clear that the majority preferred behavior-based recommendations over rigid sleep duration goals. But only a minority of the participants actively engaged with the recommendations, likely due to limited motivation to change their sleep habits, which highlights the need for more targeted inclusion criteria.
Additionally, multiple usability issues emerged, such as synchronization fails and interface confusion, which negatively influenced the user experience. Finally, algorithmic limitations and data mismatches led to potential benefits of switching from Garmin Health SDK to a possibly more reliable system like the Garmin Health API, although this would entail trade-offs regarding data transparency and the benefit of custom algorithms. Taken together, these insights offer valuable input to refine both the technological implementation and the adaptive goal-setting feature in future work.

To start, recruitment went relatively well. We aimed to recruit 15 participants and initially succeeded in doing so. However, due to the postponement of the study, four participants withdrew because of holiday plans. Recruitment through local sports associations with large adult memberships proved the most effective strategy, as all 15 participants were recruited through this method, no other methods were used. Therefore, it is unknown whether alternative recruitment strategies can be effective. Furthermore, the inclusion criteria required the participants to be healthy, in terms of quality, duration, or habits. As a result, 70% of the participants (7 out of 10) mentioned during the focus group that they did not have a specific sleep-related goal or intrinsic motivation to change their habits. This lack of engagement may had impact on the effectiveness and relevance of the study. Therefore, adding a criterion that participants should show willingness or intention to improve their sleep could help the overall impact of the follow-up study.
Retention during the study was acceptable, with 82% of enrolled participants (9 out of 11, excluding the four who withdrew before the start) completing the study. This suggests an acceptable level of engagement and acceptability. The two dropouts were due to technical difficulties with the application. In addition to these two, two others also reported frustration when using the application on multiple domains, which highlights the importance of improving the application for the follow-up study.

The purpose of this study was to evaluate the practical implementation of the adaptive goal-setting feature within the sleep domain, focusing on both the functioning of the algorithm and the user experience with the application. As discussed in Chapter 5.3, 70% of the participants (7 out of 10) expressed their preference for receiving recommendations to adjust their sleep-related

habits and conditions, rather than being given direct goals such as a specific sleep duration. This suggests that implementing the adaptive goal-setting feature to directly increase sleep duration may be less effective. Instead, a more feasible approach may be to use the feature to slowly adapt sleep-related behavior to improve sleep quality over time.
Although the recommendations provided to the participants did include sleep duration as a goal, they also provided generic advice to improve sleep-related habits and conditions as well. For some participants (27%, 3 out of 11), the recommendations helped, as they actively made changes to their routines and habits to improve their sleep. This indicates that behavior-based recommendations could be helpful for sleep improvement.
On the other hand, the ultimate aim of the MOCIA project is to maintain or enhance cognitive health, which is linked to sleep duration. Adjusting your sleep conditions and habits may result in falling asleep more easily and have fewer nightly awakenings, but the overall improvement in total sleep duration may be limited (e.g. around 30 minutes). This could be a first step, but to effectively support cognitive health, an increase in overall sleep duration may still be necessary, assuming it is not already at an acceptable level. Therefore, it may be important to keep sleep duration as part of the recommendations.

During this study, multiple unforeseen usability issues became clear, that provide important insights for improving. One participant, for example, reported multiple disturbances during their first night wearing the watch due to the screen lighting up during movement. This issue was resolved by covering the watch with a wrist sweatband, a simple but effective solution. Therefore, wrist sweatbands could be given to participants during intake meetings in the follow-up study.
It was already known that iPhone users were required to manually synchronize the watch. However, an unexpected issues presented itself. In some cases, synchronizing took several hours, as mentioned in Chapter 5.3. This needs to be resolved, and an additional improvement would be to add a progress bar to the synchronization pop-up.
The application's home screen also displayed the time of the last synchronization. This was often outdated, and caused stress and confusion among participants. Therefore, it is recommended to resolve this issue or remove this feature.
Another usability issue problem involed the questionnaire overview. As discussed in Chapter 5.3, completed questionnaires remained visible and unchanged in the list. This caused confusion among participants, and should be resolved.
All together, these issues highlight a significant number of required improvements for the application. The OnePlanet Research mobile application connects to the Garmin vívosmart® 5 via Garmin Health SDK, allowing the use of a custom made algorithm developed by imec NL and providing access to sensor data. However, as mentioned above, this application and its algorithm still have some flaws that need to be resolved before continuing with the follow-up study.
A different approach would be to replace the Garmin Health SDK with the Garmin Health API. This way the participants connect their Garmin vívosmart® 5 to the Garmin Connect application. The data would then become accessible through the Health API, removing the need for the OnePlanet Research application for synchronization. This would enhance the user experience and reduce technical issues.
With this alternative, the algorithm developed by imec NL for this study would no longer be strictly necessary, but it could still be useful. When using the Garmin Health API, HRV data is no longer included in the JSON files extracted from the watch. Since the algorithm relies on HRV as one of its inputs, this would be a limitation. However, the algorithm is designed to function even without HRV, using HR ad step data instead. Therefore, it could still be used, but with less detailed input.
Alternatively, sleep data collected via Garmin Connect could also be processed by a new or mod-

ified custom algorithm, which combines sleep metrics with reported sleep quality and provides recommendations through a separate application developed by Vivica. The questionnaires used to collect perceived sleep quality could be implemented into this same application, making the use of the OnePlanet Research mobile application unnecessary.
An additional benefit of using Garmin Connect is its interface and features. As mentioned in group, Chapter 5.3, 78% of the participants (7 out of 9) expressed their preference for graphs and interfaces, such as shown in Figure 7.
However, there are potential downsides to this solution. Participants might have to install and manage three different mobile applications, which could negatively impact user experience and engagement. Besides that, fully using the Garmin Health API and their algorithms for the Garmin Connect application would result in less insights into how sleep durations are calculated. These parts would become a "black box", making it hard to understand or validate the outcomes.
Issues regarding the accuracy of the custom algorithm also became clear. In this study, whilst using the Garmin Health SDK and the custom algorithm, only 39.8% of the algorithm-generated sleep durations matched participants' self-reported durations within ±45 minutes. Several factors may explain this low correspondence, including false answers in self-reports and flaws in the algorithm. Additionally, the algorithm failed to return sleep data in 31.1% of the measurements (42 out of 135), which is most likely due to insufficient data in the JSON files. However, at this point, it is unknown what causes the absence of required data in the JSON files, and whether this solely affects the Garmin Health SDK, or the Garmin Health API as well. Additionally, for the 56 sensor-based measurements that fell outside the ± 45 minutes margin, the exact reasons for these mismatches remain unclear. Understanding these mismatches would require a detailed analysis of each individual night, as well as the use of a more reliable validation method than self-reported bedtimes and wake-up times. This would allow for a proper validation of the algorithm's performance and help determine whether it remains suitable when using the Garmin Health API, or if switching to Garmin's own algorithm would be more beneficial despite the potential disadvantages.
If the current setup with the custom algorithm and Garmin Health SDK is maintained and its technical issues resolved, future work could also focus on enhancing the algorithm by implementing machine learning. This could improve both the accuracy of sleep detection and personalized recommendations [38].

Finally, the final sample size of 11 participants was sufficient for the initial feasibility study, but smaller than the intended 15. A larger sample would have reduced the relative impact of dropouts on the overall results. On the other hand, they did report useful insights for further improvements. Additionally, the study duration was shortened from the originally planned four weeks to 15 days. This limited time frame may have limited the participants' ability to adjust their behavior or to observe trends in the data. Several participants reported during the focus group sessions that a longer period, and a weekly interval for receiving recommendations, would have been better for the study.

For future work, it is important to distinguish between short-term improvements that can be implemented immediately and long-term tasks that require more extensive improvements. One immediate improvement is to refine the inclusion criteria to better target participants who are motivated to change their sleep habits. Additionally, several factors of the intake meeting can be improved. For example, participants could be provided with wrist sweatbands to block light emitted by the watch at night whilst moving, and the introduction to the provided sleep education could be improved. This education could also be provided multiple times throughout the study.
The duration of a feasibility study should be extended to at least four weeks, with eight weeks being preferable, with a weekly recommendation and daily insights in their progress. For a larger

follow-up study, an even longer duration is recommended to allow enough time for behavioral changes to occur and trends to emerge.
In short term, the application could be improved by integrating more visual and numerical elements, such as graphs and data visualizations, which are highly recommended by participants. Moreover, the recommendations provided by the algorithm should be made more personalized and less repetitive to maintain user engagement.
If the OnePlanet Research mobile application continues to be used, it may be advisable to exclude iPhone users or provide them with Android devices, since this is presumably a difficult issue to resolve. This would help reduce technical issues and allow participants to focus on the study rather than on technical difficulties.
In the longer term, further investigation is needed to validate the algorithm developed by imec NL if it continues to be used. Additionally, the potential of using the Garmin Health API and Garmin's algorithm should be evaluated with a similar feasibility study. It is important to assess whether the potential benefits of switching to the Garmin Health API and their algorithm outweigh the limitations or whether continued use of the imec NL algorithm in combination with the Garmin Health API represents the optimal solution.

7 CONCLUSION

This feasibility study assessed the practical implementation of an adaptive goal-setting feature for sleep improvement within a mobile application for older adults. Overall, the results suggest that it is feasible, but various improvements are necessary before continuing with a follow-up study. Recruitment was initially successful, although the reduced sample size and study duration limited the ability to observe behavioral change. Participants reported a preference for sleep recommendations focused on habits and conditions, rather than fixed goals like sleep duration. This highlights the importance of personalized feedback to what users can actually control.
Analysis of the Garmin vívosmart® 5 data and the custom algorithm developed by imec NL showed that 69% of the nights (93 out of 135) were based on actual sensor-based measurements. However, only 40% of these measurements matched the individual sleep duration goals within a $\pm$ 45 minute margin, emphasizing the need for further validation of the algorithm. On a more positive note, three participants showed improvement across all assessed domains, including increased sleep consistency, better perceived sleep quality, fewer nightly awakenings, and a higher rate of falling asleep within 30 minutes. An additional five participants showed improvements in most, but not all, domains.
Qualitative feedback from the focus group also provided valuable insights into participant preferences, particularly regarding desired sleep duration, user experience with the application, data visualizations, and the effectiveness of the recommendations.
Future work should focus on several main domains. First, inclusion criteria should better target participants who are motivated to improve their sleep. Second, usability can be improved by offering wrist sweatbands, improving sleep education and extending the study duration to at least four to eight weeks. Short-term improvements include adding more personalized recommendations and integrating visualizations in the application. Finally, further validation of the imec NL algorithm is needed, as well as a evaluation of the Garmin Health API to determine the most effective and user-friendly setup.

REFERENCES

[1] Alzheimer Nederland. Feiten en cijfers over dementie, 3 2025.

[2] 2024 Alzheimer's disease facts and figures. *Alzheimer's & Dementia*, 20(5):3708–3821, 5 2024.

[3] Christophe' 'Bintener, Owen' 'Miller, and Jean' 'Georges. Dementia in Europe Yearbook 2019. Technical report, Alzheimer Europe, Luxembourg, 2019.

[4] Eurostat. Ageing Europe, 4 2025.

[5] Liesi E. Hebert, Jennifer Weuve, Paul A. Scherr, and Denis A. Evans. Alzheimer disease in the United States (2010–2050) estimated using the 2010 census. *Neurology*, 80(19):1778–1783, 5 2013.

[6] David B. Reuben, Sarah Kremen, and Donovan T. Maust. Dementia Prevention and Treatment. *JAMA Internal Medicine*, 184(5):563, 5 2024.

[7] Edwin A. Locke. Toward a theory of task motivation and incentives. *Organizational Behavior and Human Performance*, 3(2):157–189, 5 1968.

[8] Edwin A. Locke and Gary P. Latham. Building a practically useful theory of goal setting and task motivation: A 35-year odyssey. *American Psychologist*, 57(9):705–717, 9 2002.

[9] Edwin A Locke and Gary P Latham. *A theory of goal setting & task performance.* Prentice-Hall, Inc, Englewood Cliffs, NJ, US, 1990.

[10] Maurice M. Ohayon, Mary A. Carskadon, Christian Guilleminault, and Michael V. Vitiello. Meta-Analysis of Quantitative Sleep Parameters From Childhood to Old Age in Healthy Individuals: Developing Normative Sleep Values Across the Human Lifespan. *Sleep*, 27(7):1255–1273, 10 2004.

[11] Siegwart Lindenberg and Linda Steg. Normative, Gain and Hedonic Goal Frames Guiding Environmental Behavior. *Journal of Social Issues*, 63(1):117–137, 3 2007.

[12] Ian Li, Anind Dey, and Jodi Forlizzi. A stage-based model of personal informatics systems. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 557–566, New York, NY, USA, 4 2010. ACM.

[13] Julia F Dewald, Anne M Meijer, Frans J Oort, Gerard A Kerkhof, and Susan M Bögels. The influence of sleep quality, sleep duration and sleepiness on school performance in children and adolescents: A meta-analytic review. *Sleep Medicine Reviews*, 14(3):179–189, 2010.

[14] Émilie Fortier-Brochu, Simon Beaulieu-Bonneau, Hans Ivers, and Charles M Morin. Insomnia and daytime cognitive performance: A meta-analysis. *Sleep Medicine Reviews*, 16(1):83–94, 2012.

[15] Omonigho M. Bubu, Michael Brannick, James Mortimer, Ogie Umasabor-Bubu, Yuri V. Sebastião, Yi Wen, Skai Schwartz, Amy R. Borenstein, Yougui Wu, David Morgan, and William M. Anderson. Sleep, Cognitive impairment, and Alzheimer's disease: A Systematic Review and Meta-Analysis. *Sleep*, 40(1), 1 2017.

[16] Katie Moraes de Almondes, Mônica Vieira Costa, Leandro Fernandes Malloy-Diniz, and Breno Satler Diniz. Insomnia and risk of dementia in older adults: Systematic review and meta-analysis. *Journal of Psychiatric Research*, 77:109–115, 2016.

[17] Mark Conner, Sarah Wilding, Andrew Prestwich, Russell Hutter, Robert Hurling, Frenk van Harreveld, Charles Abraham, and Paschal Sheeran. Goal prioritization and behavior change: Evaluation of an intervention for multiple health behaviors. *Health Psychology*, 41(5):356–365, 5 2022.

[18] Miguel Marino, Yi Li, Michael N. Rueschman, J. W. Winkelman, J. M. Ellenbogen, J. M. Solet, Hilary Dulin, Lisa F. Berkman, and Orfeu M. Buxton. Measuring Sleep: Accuracy, Sensitivity, and Specificity of Wrist Actigraphy Compared to Polysomnography. *Sleep*, 36(11):1747–1755, 11 2013.

[19] Vincent T. van Hees, Séverine Sabia, Kirstie N. Anderson, Sarah J. Denton, James Oliver, Michael Catt, Jessica G. Abell, Mika Kivimäki, Michael I. Trenell, and Archana Singh-Manoux. A Novel, Open Access Method to Assess Sleep Duration Using a Wrist-Worn Accelerometer. *PLOS ONE*, 10(11):e0142533, 11 2015.

[20] Garmin. Garmin Health Science Advanced Sleep Monitoring.

[21] Garmin. Garmin Connect Developer Program.

[22] Rollin Mccraty and Fred Shaffer. Heart Rate Variability: New Perspectives on Physiological Mechanisms, Assessment of Self-regulatory Capacity, and Health Risk. *Global Advances in Health and Medicine*, 4(1):46–61, 1 2015.

[23] Phyllis K. Stein and Yachuan Pu. Heart rate variability, sleep and sleep disorders. *Sleep Medicine Reviews*, 16(1):47–66, 2 2012.

[24] Philippe Boudreau, Wei-Hsien Yeh, Guy A. Dumont, and Diane B. Boivin. Circadian Variation of Heart Rate Variability Across Sleep Stages. *Sleep*, 36(12):1919–1928, 12 2013.

[25] Danguolė ŽEmaitytė, Giedrius Varoneckas, and Eugene Sokolov. Heart Rhythm Control During Sleep. *Psychophysiology*, 21(3):279–289, 5 1984.

[26] Garmin. Garmin vívosmart 5.

[27] Richard M. Ryan and Edward L. Deci. On Happiness and Human Potentials: A Review of Research on Hedonic and Eudaimonic Well-Being. *Annual Review of Psychology*, 52(1):141–166, 2 2001.

[28] Alan S. Waterman. Two conceptions of happiness: Contrasts of personal expressiveness (eudaimonia) and hedonic enjoyment. *Journal of Personality and Social Psychology*, 64(4):678–691, 4 1993.

[29] Antonella Delle Fave, Ingrid Brdar, Teresa Freire, Dianne Vella-Brodrick, and Marié P. Wissing. The Eudaimonic and Hedonic Components of Happiness: Qualitative and Quantitative Findings. *Social Indicators Research*, 100(2):185–207, 1 2011.

[30] Veronika Huta. An overview of hedonic and eudaimonic well-being concepts. 4 2015.

[31] Jasmin Niess and Paweł W. Woźniak. Supporting Meaningful Personal Fitness. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–12, New York, NY, USA, 4 2018. ACM.

[32] Nikita Sharma. *Sensing the care: advancing unobtrusive sensing solutions to support informal caregivers of older adults with cognitive impairment.* PhD thesis, University of Twente, Enschede, The Netherlands, 2 2024.

[33] Sang-Woo Jeong, Sun-Hwa Kim, Si-Hyuck Kang, Hee-Jun Kim, Chang-Hwan Yoon, Tae-Jin Youn, and In-Ho Chae. Mortality reduction with physical activity in patients with and without cardiovascular disease. *European Heart Journal*, 40(43):3547–3555, 11 2019.

[34] Garmin. Garmin Support.

[35] Jared Weintraub, David Cassell, and Thomas P. DePatie. Nudging flow through 'SMART' goal setting to decrease stress, increase engagement, and increase performance at work. *Journal of Occupational and Organizational Psychology*, 94(2):230–258, 6 2021.

[36] Daniel J. Buysse, Charles F. Reynolds, Timothy H. Monk, Susan R. Berman, and David J. Kupfer. The Pittsburgh sleep quality index: A new instrument for psychiatric practice and research. *Psychiatry Research*, 28(2):193–213, 5 1989.

[37] Fabian Theurl, Michael Schreinlechner, Nikolay Sappler, Michael Toifl, Theresa Dolejsi, Florian Hofer, Celine Massmann, Christian Steinbring, Silvia Komarek, Kurt Mölgg, Benjamin Dejakum, Christian Böhme, Rudolf Kirchmair, Sebastian Reinstadler, and Axel Bauer. Smartwatch-derived heart rate variability: a head-to-head comparison with the gold standard in cardiovascular disease. *European Heart Journal - Digital Health*, 4(3):155–164, 6 2023.

[38] Elliot G. Mitchell, Elizabeth M. Heitkemper, Marissa Burgermaster, Matthew E. Levine, Yishen Miao, Maria L. Hwang, Pooja M. Desai, Andrea Cassells, Jonathan N. Tobin, Esteban G. Tabak, David J. Albers, Arlene M. Smaldone, and Lena Mamykina. From Reflection to Action: Combining Machine Learning with Expert Knowledge for Nutrition Goal Recommendations. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, New York, NY, USA, 2021. Association for Computing Machinery.

A DAILY QUESTIONNAIRE (ENGLISH)

1. How would you rate the quality of your sleep during the past 24 hours? Consider how many hours you slept, how easily you fell asleep, how often you woke up during the night, whether you woke up too early, and how refreshing your sleep was.
   (Visual Analog Scale: 0–10, where 0 = very poor, 10 = very good)

2. Did you have trouble falling asleep because you were awake for more than 30 minutes before falling asleep?
   (Options: Yes / No)

3. How much trouble did you have today with having enough motivation or enthusiasm to do things?
   (Options: No problem at all / A little problem / Somewhat of a problem / Quite a problem / A major problem)

4. What time did you go to bed?
   (Time input)

5. What time did you get out of bed?
   (Time input, between 03:00 and 15:00)

6. Did you have trouble sleeping last night because you woke up during the night or early morning (e.g., to use the bathroom)? If yes, how many times?
   (Options: Not at all / Once / Twice / Three times / More than three times)

7. How much time were you awake in total after waking up during the night (e.g., to use the bathroom)?
   (Numeric input in minutes)

B DAILY QUESTIONNAIRE (NEDERLANDS)

1. Hoe zou u de kwaliteit van uw slaap in de afgelopen 24 uur beoordelen?
   (VAS 0–10, 0 = zeer slecht, 10 = zeer goed)

2. Had u afgelopen nacht moeite met slapen omdat u langer dan 30 minuten wakker lag voordat u in slaap viel?
   (Keuze: Ja / Nee)

3. Hoeveel problemen ervaarde u de afgelopen dag met genoeg zin / enthousiasme te hebben om dingen te doen?
   (Keuze: Helemaal geen probleem / Een klein beetje een probleem / Enigszins een probleem / Behoorlijk een probleem / Een groot probleem)

4. Hoe laat bent u in bed gaan liggen?
   (Tijdsinvoer)

5. Hoe laat bent u uit bed gegaan?
   (Tijdsinvoer tussen 03:00 en 15:00)

6. Had u afgelopen nacht moeite met slapen omdat u 's nachts of in de vroege ochtend wakker werd (bijvoorbeeld om naar het toilet te gaan)? Zo ja, hoe vaak?
   (Keuze: Niet / Een keer / Twee keer / Drie keer / Meer dan drie keer)

7. Hoe lang bent u in totaal 's nachts wakker geweest nadat u wakker bent geworden, bijvoorbeeld om naar de wc te gaan?
   (Numerieke invoer in minuten)

C PSQI QUESTIONNAIRE (ENGLISH)

1. During the past month, what time have you usually gone to bed?
   (Time input)

2. How long (in minutes) has it usually taken you to fall asleep?
   (Numeric input)

3. What time have you usually gotten up in the morning?
   (Time input between 03:00 and 15:00)

4. How many hours of actual sleep did you get at night? (This may differ from the number of hours spent in bed.)
   (Numeric input)

a–j) How often during the past month have you had trouble sleeping because of the following reasons?
(Options: Not during the past month / Less than once a week / Once or twice a week / Three or more times a week)

   a) Could not fall asleep within 30 minutes
   b) Woke up in the middle of the night or early morning
   c) Had to get up to use the bathroom
   d) Could not breathe comfortably
   e) Coughed or snored loudly
   f) Felt too cold
   g) Felt too hot
   h) Had bad dreams
   i) Had pain
   j) Other reason (optional: specify)

5. During the past month, how would you rate your overall sleep quality?
   (Options: Very good / Fairly good / Fairly bad / Very bad)

6. How often did you take medicine (prescribed or over-the-counter) to help you sleep?
   (Same frequency options as above)

7. How often have you had trouble staying awake while driving, eating meals, or engaging in social activity?
   (Same frequency options as above)

8. How much trouble did you have with enthusiasm or motivation to get things done?
   (Options: No problem / A little problem / Somewhat of a problem / A big problem)

9. Do you have a bed partner or roommate?
(Options: No / Partner in another room / Partner in same room, different bed / Partner in same bed)

10a–10e) If yes: Ask your partner or roommate how often you have...

a) Snored loudly
b) Had long pauses between breaths while sleeping
c) Had leg movements or jerking during sleep
d) Appeared confused or disoriented during sleep
e) Shown other types of restlessness (optional: describe)

D PSQI QUESTIONNAIRE (NEDERLANDS)

1. Hoe laat bent u gewoonlijk naar bed gegaan in de afgelopen maand?
   (Tijdsinvoer)

2. Hoe lang (in minuten) duurde het meestal voordat u in slaap viel?
   (Numerieke invoer)

3. Hoe laat bent u gemiddeld opgestaan in de ochtend?
   (Tijdsinvoer tussen 03:00 en 15:00)

4. Hoeveel uur heeft u gemiddeld daadwerkelijk geslapen per nacht? (Dit kan verschillen van het aantal uren dat u in bed lag.)
   (Numerieke invoer)

a–j) Hoe vaak had u in de afgelopen maand moeite met slapen vanwege de volgende redenen?
(Opties: Niet gedurende deze maand / Minder dan 1 keer per week / 1–2 keer per week / 3 of meer keren per week)

- a) Kon niet binnen 30 minuten in slaap vallen
- b) Werd wakker midden in de nacht of te vroeg in de ochtend
- c) Moest naar het toilet
- d) Kon niet goed ademhalen
- e) Hoestte of snurkte luid
- f) Had het te koud
- g) Had het te warm
- h) Had nare dromen
- i) Had pijn
- j) Andere reden (optioneel: specificeren)

5. Hoe zou u uw algemene slaapkwaliteit beoordelen over de afgelopen maand?
   (Opties: Heel goed / Redelijk goed / Redelijk slecht / Heel slecht)

6. Hoe vaak heeft u slaapmedicatie genomen om in slaap te komen? (Voorgeschreven of zelf gekocht bij apotheek/drogist)
   (Zelfde antwoordopties als hierboven)

7. Hoe vaak had u moeite om wakker te blijven tijdens autorijden, eten of sociale activiteiten?
   (Zelfde antwoordopties als hierboven)

8. Hoeveel moeite had u met enthousiasme of motivatie om dingen te doen gedurende de dag?
   (Opties: Helemaal geen probleem / Klein probleem / Enigszins een probleem / Groot probleem)

9. Heeft u een bedpartner of kamergenoot?
   (Opties: Geen / Partner in andere kamer / Partner in zelfde kamer, ander bed / Partner in zelfde bed)

10a–10e) Indien ja: Vraag uw partner of kamergenoot hoe vaak u...

   a) Luid snurkte
   b) Pauzes tussen ademhalingen had tijdens het slapen
   c) Bewegingen of schokken met de benen had tijdens het slapen
   d) Gedesoriënteerd of verward leek tijdens het slapen
   e) Andere vormen van onrust vertoonde tijdens het slapen (optioneel: beschrijven)

E DAY 5 QUESTIONNAIRE (ENGLISH)

1. Do you use digital lifestyle apps? Think of apps that help with exercise, mindfulness, nutrition, sleep, etc.
   (Open-ended)
2. Have you previously set goals using such an app? If so, in which domains?
   (Open-ended)
3. Specify: What would you like to improve about your sleep?
   (Open-ended)
4. Measurable: How will you track your progress?
   (Open-ended)
5. Achievable: What small steps will help you reach this goal?
   (Open-ended)
6. Realistic: Why is this goal important to you?
   (Open-ended)
7. Time-bound: When do you want to achieve this goal?
   (Open-ended)
8. Which specific aspects of your sleep quality would you like to improve?
   (Open-ended)
9. Write down your own short-term sleep goal(s).
   (Open-ended)
10. Write down your own long-term sleep goal(s).
    (Open-ended)

F DAY 5 QUESTIONNAIRE (NEDERLANDS)

1. Gebruikt u digitale leefstijlapps? Denk hierbij aan apps die u helpen om te sporten, mindfulness, voeding, slaap, etc.
   (Open vraag)
2. Heeft u eerder doelen gesteld met zo'n app? Zo ja, in welke gebieden?
   (Open vraag)
3. Specificeer: Wat wilt u verbeteren aan uw slaap?
   (Open vraag)
4. Meetbaar: Hoe houdt u uw voortgang bij?
   (Open vraag)
5. Acceptabel: Welke kleine stappen gaan u helpen dit doel te behalen?
   (Open vraag)
6. Realistisch: Waarom is dit doel belangrijk voor u?
   (Open vraag)
7. Tijdgebonden: Wanneer wilt u dit doel bereiken?
   (Open vraag)
8. Welke specifieke aspecten van uw slaapkwaliteit wilt u verbeteren?
   (Open vraag)
9. Schrijf uw eigen korte termijn doel(en) op.
   (Open vraag)
10. Schrijf uw eigen lange termijn doel(en) op.
    (Open vraag)

G DAY 10 QUESTIONNAIRE (ENGLISH)

1. Were you able to maintain your short-term sleep goal?
   (Open-ended)
2. Did tracking your progress help you stay motivated?
   (Open-ended)
3. What prevented you from maintaining your goal?
   (Open-ended)
4. Would you like to adjust your short- and/or long-term sleep goals?
   (Open-ended)
5. Write your new short-term sleep goal(s). (Leave blank if no changes)
   (Open-ended)
6. Write your new long-term sleep goal(s). (Leave blank if no changes)
   (Open-ended)

H DAY 10 QUESTIONNAIRE (NEDERLANDS)

1. Is het u gelukt om uw korte termijn slaapdoel vol te houden?
   (Open vraag)

2. Heeft het bijhouden van uw voortgang u geholpen om gemotiveerd te blijven?
   (Open vraag)

3. Wat heeft u ervan weerhouden om uw doel vol te houden?
   (Open vraag)

4. Wilt u uw korte en/of lange termijn slaapdoel aanpassen?
   (Open vraag)

5. Schrijf uw nieuwe korte termijn doel(en) op. (Indien u uw korte termijn doel niet wilt aanpassen, hoeft u niets in te vullen)
   (Open vraag)

6. Schrijf uw nieuwe lange termijn doel(en) op. (Indien u uw lange termijn doel niet wilt aanpassen, hoeft u niets in te vullen)
   (Open vraag)

I DAY 15 QUESTIONNAIRE (ENGLISH)

1. Were you able to reach your short-term sleep goal?
   (Open-ended)
2. Would you be interested in continuing to practice sleep goals after the study?
   (Open-ended)
3. Would you like to set a new or updated goal?
   (Open-ended)
4. Write your new short-term sleep goal(s). (Leave blank if no changes)
   (Open-ended)
5. Write your new long-term sleep goal(s). (Leave blank if no changes)
   (Open-ended)

J DAY 15 QUESTIONNAIRE (NEDERLANDS)

1. Is het u gelukt om uw korte termijn slaapdoel te bereiken?
   (Open vraag)
2. Zou u geïnteresseerd zijn om na het onderzoek door te gaan met het oefenen van slaapdoelen?
   (Open vraag)
3. Wilt u een nieuw of aangepast doel stellen?
   (Open vraag)
4. Schrijf uw nieuwe korte termijn doel(en) op. (Indien u uw korte termijn doel niet wilt aanpassen, hoeft u niets in te vullen)
   (Open vraag)
5. Schrijf uw nieuwe lange termijn doel(en) op. (Indien u uw lange termijn doel niet wilt aanpassen, hoeft u niets in te vullen)
   (Open vraag)

K ALGORITHM - PYTHON SCRIPT - GARMIN DATA AND SUBJECTIVE DATA

Listing K.1: HRV analysis code

```python
#Copyright (c), 2025, OnePlanet Research Center & University of Twente

# Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

# The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.


import json
import math
import os
import shutil
from datetime import datetime, timedelta
from io import BytesIO
from zipfile import ZipFile
import numpy as np
import pandas as pd
import pytz
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import scipy.signal
from azure.storage.blob import BlobServiceClient
from scipy.signal import butter, filtfilt


class Garmin:
    def __init__(self, account_url, sas_token):
        self.blob_service_client = BlobServiceClient(account_url=account_url, credential=sas_token)

    def getAllBlobNames(self, containerSubject="participants"):
```

```
        container_client = self.blob_service_client.get_container_client(container=containerSubject)
        blob_names_list = list(container_client.list_blob_names())
        sensordata_blob_names = [i for i in blob_names_list if "sensordata" in i]
        garmin_blob_names = [i for i in blob_names_list if "garmindata" in i]
        return sensordata_blob_names, garmin_blob_names

    def getNewBlobNames(self, sensordata_blob_list, garmin_blob_list):
        sensordata_folders = list(set([i.split("/")[-1].split(".")[0] for i in sensordata_blob_list]))
        garmindata_folders = list(set([i.split("/")[-2] for i in garmin_blob_list]))
        newFolders = list(set(sensordata_folders) - set(garmindata_folders))
        newBlobs = [i for i in sensordata_blob_list if i.split("/")[-1].split(".")[0] in newFolders]
        return newBlobs

    def unzipGarminFiles(self, blob_names_list, subjectID, containerSubject="participants"):
        # This gets all the blob names from Azure
        container_client = self.blob_service_client.get_container_client(container=containerSubject)
        # temporary_local_path = os.getcwd() + "/zips"
        # os.makedirs(temporary_local_path)
        for blob_name in blob_names_list:
            blob_client = self.blob_service_client.get_blob_client(container=containerSubject,
                                                                   blob=blob_name)
            with BytesIO() as input_blob: # Writes data to a temporary file
                blob_client.download_blob().readinto(input_blob)
                input_blob.seek(0) # This can, I think, be removed...
                with ZipFile(input_blob, 'r') as zipObj:
                    zip_names = zipObj.namelist()
                    zip_names = [i for i in zip_names if ".json" in i]
                    print(zip_names)
                    for zipname in zip_names:
                        data = zipObj.read(zipname)
                        filename = subjectID + "/garmindata/" + zipname
                        blob_client = self.blob_service_client.get_blob_client(container=containerSubject,
                                                                               blob=filename)
                        blob_client.upload_blob(data, overwrite=True)

                    # zipObj.extractall(temporary_local_path)
            # datafolder = os.listdir(temporary_local_path)[0]
            # folder_path = temporary_local_path + "/" + datafolder
            # datafile_list = os.listdir(folder_path)
            # for datafile in datafile_list:
            #   file_path = folder_path + "/" + datafile
            #   datafile_name = "/" + datafile
            #   new_blob_name = blob_name.replace("sensordata", "garmindata")
            #   new_blob_name = new_blob_name.replace(".zip", datafile_name)
            #   with open(file=file_path, mode="rb") as data:
            #       container_client.upload_blob(name=new_blob_name, data=data, overwrite=True)
            # shutil.rmtree(folder_path)
        # shutil.rmtree(temporary_local_path)
        garmin_blob_names = []
        for folder in [i.split("/")[-1].split(".")[0] for i in blob_names_list]:
```

```
            start_string = subjectID + "/garmindata/" + folder
            garmin_blob_names = (garmin_blob_names +
                                 list(container_client.list_blob_names(name_starts_with=start_string)))
        return garmin_blob_names

    def getOperatingSystem(self, blobName):
        filename = blobName.split('/')[-1]
        if filename[0].isnumeric():
            return "android"
        elif not filename[0].isnumeric():
            return "ios"
        else:
            return "unknown_os"

    def getMetsIos(self, activityList):
        METS_dict = {"sedentary": 1.4,
                     "standing": 1.6,
                     "generic": 1.6,
                     "walking": 3.5,
                     "cycling": 7.0,
                     "running": 9.0}

        metsList = [METS_dict[i] for i in activityList]
        return metsList

    def getMetsAndroid(self, activityList):
        METS_dict = {"SEDENTARY": 1.4,
                     "GENERIC": 1.6,
                     "WALKING": 3.5,
                     "CYCLING": 7.0,
                     "RUNNING": 9.0}

        metsList = [METS_dict[i] for i in activityList]
        return metsList

    def readWellnessIos(self, inputBlob):
        wellnessData = json.load(inputBlob)
        checks = ['heartRate', 'steps', 'distance', 'ascent', 'descent', 'moderateActivityMinutes',
                  'vigorousActivityMinutes', 'intensity', 'totalCalories', 'activeCalories']
        tsList = [i['startTime'] for i in wellnessData if all([x in i for x in checks])]
        hrList = [i['heartRate'] for i in wellnessData if all([x in i for x in checks])]
        stepsList = [i['steps'] for i in wellnessData if all([x in i for x in checks])]
        descriptionList = [i['description'] for i in wellnessData if all([x in i for x in checks])]
        activityListRaw = [i.split("activityType: ")[-1] for i in descriptionList]
        activityListRaw = [i.split(" ")[0] for i in activityListRaw]
        if any(["\n" in i for i in activityListRaw]):
            activityList = [i.replace("\n", "") for i in activityListRaw if "\n" in i]
        else:
            activityList = activityListRaw
        cumStepsList = np.cumsum(np.array(stepsList)).tolist()
        physicalActivityMetsList = self.getMetsIos(activityList)
        heartRate = dict(map(lambda i, j: (str(i), j), tsList, hrList))
```

```python
        physicalActivityClass = dict(map(lambda i, j: (str(i), j), tsList, activityList))
        physicalActivityMets = dict(map(lambda i, j: (str(i), j), tsList, physicalActivityMetsList))
        stepCount = dict(map(lambda i, j: (str(i), j), tsList, stepsList))
        cumulativeSteps = dict(map(lambda i, j: (str(i), j), tsList, cumStepsList))

        return heartRate, physicalActivityClass, physicalActivityMets, stepCount, cumulativeSteps

    def readSleepIos(self, inputBlob):
        sleepData = json.load(inputBlob)
        checks = ['startTimestamp', 'endTimestamp']
        toSleepList = [i['startTimestamp'] for i in sleepData if all([x in i for x in checks])]
        getUpList = [i['endTimestamp'] for i in sleepData if all([x in i for x in checks])]
        sleep_dict = {"sleepDuration": getUpList[0] - toSleepList[0],
                      "toBedTime": toSleepList[0],
                      "getUpTime": getUpList[0]}
        return sleep_dict

    def computeHrvFeatures(self, tsList, bbiList):
        hrvDf = pd.DataFrame({"ts": tsList, "bbi": bbiList})
        hrvDf["bbi_diff_squared"] = hrvDf["bbi"].diff() ** 2
        hrvDf.insert(0, "minutes_unix", [math.floor(i / 60) for i in hrvDf['ts']])
        hrvDf = hrvDf.groupby("minutes_unix").agg(timestamp=('ts', 'min'),
                                                  hrv_sdnn=('bbi', 'std'),
                                                  hrv_mssd=('bbi_diff_squared', 'mean'),
                                                  ibi_mean=('bbi', 'mean'))
        hrvDf = hrvDf.reset_index(drop=True)
        hrvDf['hrv_rmssd'] = np.sqrt(hrvDf['hrv_mssd'])
        hrvDf['heart_rate'] = [60000 / x for x in hrvDf['ibi_mean']]

        hrvSdnn = dict(map(lambda i, j: (str(int(i)), j), hrvDf['timestamp'].tolist(), hrvDf['hrv_sdnn'].tolist()))
        hrvRmssd = dict(map(lambda i, j: (str(int(i)), j), hrvDf['timestamp'].tolist(), hrvDf['hrv_rmssd'].tolist()))
        hrvHr = dict(map(lambda i, j: (str(int(i)), j), hrvDf['timestamp'].tolist(), hrvDf['heart_rate'].tolist()))
        return hrvSdnn, hrvRmssd, hrvHr

    def readHrIos(self, inputBlob):
        hrData = json.load(inputBlob)
        checks = ['heartRate', 'timestamp']
        hrList = [i['heartRate'] for i in hrData if all([x in i for x in checks])]
        tsList = [i['timestamp'] for i in hrData if all([x in i for x in checks])]
        heartrate = dict(map(lambda i, j: (str(int(i)), j), tsList, hrList))
        return heartrate

    def readStepsIos(self, inputBlob):
        stepData = json.load(inputBlob)
        checks = ['stepCount', 'startTimestamp']
        stepsList = [i['stepCount'] for i in stepData if all([x in i for x in checks])]
        cumStepsList = [i['totalSteps'] for i in stepData if all([x in i for x in checks])]
        tsList = [i['startTimestamp'] for i in stepData if all([x in i for x in checks])]
```

```
        stepCount = dict(map(lambda i, j: (str(int(i)), j), tsList, stepsList))
        cumStepsCount = dict(map(lambda i, j: (str(int(i)), j), tsList, cumStepsList))
        return stepCount, cumStepsCount

    def readHrvIos(self, inputBlob):
        hrvData = json.load(inputBlob)
        checks = ['interval', 'timestamp']
        bbiList = [i['interval'] for i in hrvData if all([x in i for x in checks])]
        tsList = [i['timestamp'] for i in hrvData if all([x in i for x in checks])]
        hrvSdnn, hrvRmssd, hrvHr = self.computeHrvFeatures(tsList, bbiList)
        return hrvSdnn, hrvRmssd, hrvHr

    def readAllJsonIos(self, blobNamesList, containerSubject="participants"):
        folderList = list(set([i.split("/")[-2] for i in blobNamesList]))
        folder_dict_list = []
        for folder in folderList:
            blobNames = [i for i in blobNamesList if folder in i]
            wellnessBlobNames = [i for i in blobNames if "welness" in i]
            if len(wellnessBlobNames) > 0:
                blobClient = self.blob_service_client.get_blob_client(container=
containerSubject,
                                                                   blob=wellnessBlobNames[0])
                with (BytesIO() as inputBlob):
                    blobClient.download_blob().readinto(inputBlob)
                    inputBlob.seek(0)
                    heartRate, physicalActivityClass, physicalActivityMets, stepCount,
cumulativeSteps = self.readWellnessIos(
                        inputBlob)
                if len(list(heartRate.keys())) > 0:
                    startTimeInSeconds = int(list(heartRate.keys())[0])
                    endTimeInSeconds = int(list(heartRate.keys())[-1]) + 60
                else:
                    heartRate = {}
                    physicalActivityClass = {}
                    physicalActivityMets = {}
                    stepCount = {}
                    cumulativeSteps = {}
                    startTimeInSeconds = 0
                    endTimeInSeconds = 0
            else:
                heartRate = {}
                physicalActivityClass = {}
                physicalActivityMets = {}
                stepCount = {}
                cumulativeSteps = {}
                startTimeInSeconds = 0
                endTimeInSeconds = 0

            print('startTimeInSeconds:')
            print(startTimeInSeconds)
            print('endTimeInSeconds:')
            print(endTimeInSeconds)
            print('physicalActivityClass:')
            print(physicalActivityClass)
            print('physicalActivityMets:')
            print(physicalActivityMets)

            heartrateBlobNames = [i for i in blobNames if "loggedHeartRate" in i]
```

```
        if len(heartrateBlobNames) > 0:
            blobClient = self.blob_service_client.get_blob_client(container=containerSubject,
                                                                  blob=heartrateBlobNames[0])
            with BytesIO() as inputBlob:
                blobClient.download_blob().readinto(inputBlob)
                inputBlob.seek(0)
                heartRate = self.readHrIos(inputBlob)

        print('heartRate:')
        print(heartRate)

        stepBlobNames = [i for i in blobNames if "loggedStep" in i]
        if len(stepBlobNames) > 0:
            blobClient = self.blob_service_client.get_blob_client(container=containerSubject,
                                                                  blob=stepBlobNames[0])
            with BytesIO() as inputBlob:
                blobClient.download_blob().readinto(inputBlob)
                inputBlob.seek(0)
                stepCount, cumulativeSteps = self.readStepsIos(inputBlob)

        print('stepCount:')
        print(stepCount)

        sleepBlobNames = [i for i in blobNames if "sleep" in i]
        if len(sleepBlobNames) > 0:
            blobClient = self.blob_service_client.get_blob_client(container=containerSubject,
                                                                  blob=sleepBlobNames[0])
            with BytesIO() as inputBlob:
                blobClient.download_blob().readinto(inputBlob)
                inputBlob.seek(0)
                sleepData = self.readSleepIos(inputBlob)
            hasSleep = 1
        else:
            sleepData = {"sleepDuration": 0, "toBedTime": 0, "getUpTime": 0}
            hasSleep = 0

        print('hasSleep:')
        print(hasSleep)
        print('sleepData:')
        print(sleepData)

        hrvBlobNames = [i for i in blobNames if "loggedBBI" in i]
        if len(hrvBlobNames) > 0:
            blobClient = self.blob_service_client.get_blob_client(container=containerSubject,
                                                                  blob=hrvBlobNames[0])
            with BytesIO() as inputBlob:
                blobClient.download_blob().readinto(inputBlob)
                inputBlob.seek(0)
                hrvSdnn, hrvRmssd, hrvHr = self.readHrvIos(inputBlob)
                if startTimeInSeconds == 0:
                    startTimeInSeconds = int(list(hrvSdnn.keys())[0])
                    endTimeInSeconds = int(list(hrvSdnn.keys())[-1]) + 60
        else:
```

```
                hrvSdnn = {}
                hrvRmssd = {}
                hrvHr = {}

            print('hrvSdnn:')
            print(hrvSdnn)
            print('hrvRmssd:')
            print(hrvRmssd)

            if hrvHr != {}:
                if heartRate == {}:
                    heartRate = hrvHr
                elif len(heartRate.keys()) < len(hrvHr.keys()):
                    heartRate = hrvHr

            folder_dict = {"startTimeInSeconds": startTimeInSeconds,
                           "endTimeInSeconds": endTimeInSeconds,
                           "heartRate": heartRate,
                           "hrvSdnn": hrvSdnn,
                           "hrvRmssd": hrvRmssd,
                           "physicalActivityMets": physicalActivityMets,
                           "physicalActivityClass": physicalActivityClass,
                           "stepCount": stepCount}

            folder_dict_list = folder_dict_list + [folder_dict]
        print('number of new dicts:')
        print(len(folder_dict_list))
        print('folder_dict_list:')
        print(folder_dict_list)

        return folder_dict_list

    def readHrAndroid(self, inputBlob):
        heartrateData = json.load(inputBlob)
        heartrateList = [i['heartRate'] for i in heartrateData if 'heartRate' in i]
        # statusList = [i['status'] for i in heartrateData if 'heartRate' in i]
        localTimeList = [datetime(i['timestamp']['date']['year'],
                                  i['timestamp']['date']['month'],
                                  i['timestamp']['date']['day'],
                                  i['timestamp']['time']['hour'],
                                  i['timestamp']['time']['minute'],
                                  i['timestamp']['time']['second'],
                                  tzinfo=pytz.utc) for i in heartrateData if 'heartRate' in i]
        localTimeList = [i.astimezone(pytz.timezone('Europe/Amsterdam'))
                         for i in localTimeList]
        tsList = [int(datetime.timestamp(i)) for i in localTimeList]
        heartRate = dict(map(lambda i, j: (str(i), j), tsList, heartrateList))
        return heartRate

    def readHrvAndroid(self, inputBlob):
        hrvData = json.load(inputBlob)
        bbiList = [i['bbi'] for i in hrvData if 'bbi' in i]
        localTimeList = [datetime(i['timestamp']['date']['year'],
                                  i['timestamp']['date']['month'],
                                  i['timestamp']['date']['day'],
                                  i['timestamp']['time']['hour'],
                                  i['timestamp']['time']['minute'],
```

```
                                  i['timestamp']['time']['second'],
                                  tzinfo=pytz.utc) for i in hrvData if 'bbi' in i]
        localTimeList = [i.astimezone(pytz.timezone('Europe/Amsterdam'))
                         for i in localTimeList]
        tsList = [int(datetime.timestamp(i)) for i in localTimeList]
        hrvSdnn, hrvRmssd, hrvHr = self.computeHrvFeatures(tsList, bbiList)
        return hrvSdnn, hrvRmssd, hrvHr

    def readStepsAndroid(self, inputBlob):
        stepsData = json.load(inputBlob)
        stepCountList = [i['stepCount'] for i in stepsData if 'stepCount' in i]
        totalStepsList = [i['totalSteps'] for i in stepsData if 'stepCount' in i]
        localTimeList = [datetime(i['startTimestamp']['date']['year'],
                                  i['startTimestamp']['date']['month'],
                                  i['startTimestamp']['date']['day'],
                                  i['startTimestamp']['time']['hour'],
                                  i['startTimestamp']['time']['minute'],
                                  i['startTimestamp']['time']['second'],
                                  tzinfo=pytz.utc) for i in stepsData if 'stepCount' in i]
        localTimeList = [i.astimezone(pytz.timezone('Europe/Amsterdam'))
                         for i in localTimeList]
        tsList = [int(datetime.timestamp(i)) for i in localTimeList]
        cumStepsList = np.cumsum(np.array(stepCountList)).tolist()
        stepCount = dict(map(lambda i, j: (str(i), j), tsList, stepCountList))
        cumulativeSteps = dict(map(lambda i, j: (str(i), j), tsList, cumStepsList))
        return stepCount, cumulativeSteps

    def readMotionAndroid(self, inputBlob):
        motionData = json.load(inputBlob)
        activityList = [i['activityType'] for i in motionData if 'activityType' in i]
        tsList = [i['timestamp']['begin_timestamp'] for i in motionData if 'timestamp' in i]
        physicalActivityMetsList = self.getMetsAndroid(activityList)
        physicalActivityClass = dict(map(lambda i, j: (str(i), j), tsList, activityList))
        physicalActivityMets = dict(map(lambda i, j: (str(i), j), tsList, physicalActivityMetsList))
        return physicalActivityClass, physicalActivityMets

    def readAllJsonAndroid(self, blobNamesList, containerSubject="participants"):
        folderList = list(set([i.split("/")[-2] for i in blobNamesList]))
        folder_dict_list = []
        for folder in folderList:
            blobNames = [i for i in blobNamesList if folder in i]
            heartrateBlobNames = [i for i in blobNames if "heartrate" in i]
            if len(heartrateBlobNames) > 0:
                blobClient = self.blob_service_client.get_blob_client(container=containerSubject,
                                                                      blob=heartrateBlobNames[0])
                with BytesIO() as inputBlob:
                    blobClient.download_blob().readinto(inputBlob)
                    inputBlob.seek(0)
                    heartRate = self.readHrAndroid(inputBlob)
                startTimeInSeconds = int(list(heartRate.keys())[0])
                endTimeInSeconds = int(list(heartRate.keys())[-1]) + 60
            else:
                heartRate = {}
```

```
            startTimeInSeconds = 0
            endTimeInSeconds = 0

        hrvBlobNames = [i for i in blobNames if "hrv" in i]
        if len(hrvBlobNames) > 0:
            blobClient = self.blob_service_client.get_blob_client(container=
containerSubject,
                                                                  blob=hrvBlobNames[0])
            with BytesIO() as inputBlob:
                blobClient.download_blob().readinto(inputBlob)
                inputBlob.seek(0)
                hrvSdnn, hrvRmssd, hrvHr = self.readHrvAndroid(inputBlob)
            if startTimeInSeconds == 0:
                startTimeInSeconds = int(list(hrvSdnn.keys())[0])
                endTimeInSeconds = int(list(hrvSdnn.keys())[-1]) + 60
        else:
            hrvSdnn = {}
            hrvRmssd = {}
            hrvHr = {}

        stepsBlobNames = [i for i in blobNames if "steps" in i]
        if len(stepsBlobNames) > 0:
            blobClient = self.blob_service_client.get_blob_client(container=
containerSubject,
                                                                  blob=stepsBlobNames[0])
            with BytesIO() as inputBlob:
                blobClient.download_blob().readinto(inputBlob)
                inputBlob.seek(0)
                stepCount, cumulativeSteps = self.readStepsAndroid(inputBlob)
            if startTimeInSeconds == 0:
                startTimeInSeconds = int(list(stepCount.keys())[0])
                endTimeInSeconds = int(list(stepCount.keys())[-1]) + 60
        else:
            stepCount = {}
            cumulativeSteps = {}

        motionBlobNames = [i for i in blobNames if "motion" in i]
        if len(motionBlobNames) > 0:
            blobClient = self.blob_service_client.get_blob_client(container=
containerSubject,
                                                                  blob=motionBlobNames[0])
            with BytesIO() as inputBlob:
                blobClient.download_blob().readinto(inputBlob)
                inputBlob.seek(0)
                physicalActivityClass, physicalActivityMets = self.readMotionAndroid(
inputBlob)
            if startTimeInSeconds == 0:
                startTimeInSeconds = int(list(physicalActivityClass.keys())[0])
                endTimeInSeconds = int(list(physicalActivityClass.keys())[-1]) + 60
        else:
            physicalActivityClass = {}
            physicalActivityMets = {}

        print('startTimeInSeconds:')
        print(startTimeInSeconds)
        print('endTimeInSeconds:')
        print(endTimeInSeconds)
        print('heartRate:')
```

```
            print(heartRate)
            print('physicalActivityClass:')
            print(physicalActivityClass)
            print('physicalActivityMets:')
            print(physicalActivityMets)
            print('stepCount:')
            print(stepCount)
            print('hrvSdnn:')
            print(hrvSdnn)
            print('hrvRmssd:')
            print(hrvRmssd)

            hasSleep = 0
            sleepData = {"sleepDuration": 0, "toBedTime": 0, "getUpTime": 0}

            if hrvHr != {}:
                if heartRate == {}:
                    heartRate = hrvHr
                elif len(heartRate.keys()) < len(hrvHr.keys()):
                    heartRate = hrvHr

            folder_dict = {"startTimeInSeconds": startTimeInSeconds,
                         "endTimeInSeconds": endTimeInSeconds,
                         "heartRate": heartRate,
                         "hrvSdnn": hrvSdnn,
                         "hrvRmssd": hrvRmssd,
                         "physicalActivityMets": physicalActivityMets,
                         "physicalActivityClass": physicalActivityClass,
                         "stepCount": stepCount}

            folder_dict_list = folder_dict_list + [folder_dict]
        print('number of new dicts:')
        print(len(folder_dict_list))
        print('folder_dict_list:')
        print(folder_dict_list)

        return folder_dict_list

    def readExistingData(self, subject, containerOutput="vivicadata"):
        container_client = self.blob_service_client.get_container_client(container=
containerOutput)
        blob_names_list = list(container_client.list_blob_names())
        blob_names_subject_list = [i for i in blob_names_list if subject in i]
        if len(blob_names_subject_list) > 0:
            print(blob_names_subject_list[0])
            blobClient = self.blob_service_client.get_blob_client(container=
containerOutput,
                                                        blob=blob_names_subject_list
[0])
            with BytesIO() as inputBlob:
                blobClient.download_blob().readinto(inputBlob)
                inputBlob.seek(0)
                existingDataAll = json.load(inputBlob)
                existingData = existingDataAll['garminData']
        else:
            existingData = {}
        return existingData
```

```
    def addNewData(self, existing_data, garmin_dict_list, new_zip_blobs):
        dict_keys = [i.split("/")[-1].split(".")[0] for i in new_zip_blobs]
        newData = dict(map(lambda i, j: (i, j), dict_keys, garmin_dict_list))
        existing_data.update(newData)
        return existing_data

    def computeSleep(self, dates, data, subject):
        sleepData = {}
        start_offset = 15 * 60 * 60
        print("available dates:")
        print(dates)
        if len(dates) > 1:
            timestamps = [datetime.strptime(x, "%Y-%m-%d").timestamp() for x in dates]
            print("timestamps derived from available dates:")
            print(timestamps)
            ts_diff = list(np.diff(timestamps))
            ts_diff.insert(len(ts_diff), 0)
            if any([x == 24 * 60 * 60 for x in ts_diff]):
                timestamps = [timestamps[i] for i in range(len(ts_diff)) if ts_diff[i] == 24 * 60 * 60]
                print("timestamps derived from available dates that have a consecutive day after it:")
                print(timestamps)
                folders = list(data.keys())
                print("all garmin data folders available:")
                print(folders)
                folders_ts = [x.split('_')[1] for x in folders]
                # This starts a for loop over nights that we can compute features of.
                # All features can best be computed within this for-loop
                for ts in timestamps:
                    ts_start = ts + start_offset
                    ts_end = ts + 24 * 60 * 60 + start_offset
                    folders_oi = [folders[x] for x in range(len(folders_ts)) if
                                  ts_start < float(folders_ts[x])]
                    print("garmin data folders available that could have date of this night:")
                    print(folders_oi)
                    data_hr = pd.DataFrame(columns=["ts", "heartRate"])
                    data_hrv = pd.DataFrame(columns=["ts", "hrvSdnn"])
                    data_stepcount = pd.DataFrame(columns=["ts", "stepCount"])
                    if len(folders_oi) > 0:
                        for folder in folders_oi:
                            print("active folder:")
                            print(folder)
                            ts_hr_oi = [x for x in list(data[folder]['heartRate'].keys()) if ts_start < int(x) < ts_end]
                            if len(ts_hr_oi) > 0:
                                datapart_hr = pd.DataFrame.from_dict(
                                    {'ts': [int(x) for x in ts_hr_oi if x in list(data[folder]['heartRate'].keys())],
                                     'heartRate': [data[folder]['heartRate'][x] for x in ts_hr_oi]})
                            else:
                                datapart_hr = pd.DataFrame(columns=["ts", "heartRate"])

                            ts_hrv_oi = [x for x in list(data[folder]['hrvSdnn'].keys()) if ts_start < int(x) < ts_end]
                            if len(ts_hrv_oi) > 0:
```

```
                            datapart_hrv = pd.DataFrame.from_dict(
                                {'ts': [int(x) for x in ts_hrv_oi if x in list(data[folder]['hrvSdnn'].keys())],
                                 'hrvSdnn': [data[folder]['hrvSdnn'][x] for x in ts_hrv_oi]})
                        else:
                            datapart_hrv = pd.DataFrame(columns=["ts", "hrvSdnn"])

                        ts_sc_oi = [x for x in list(data[folder]['stepCount'].keys()) if
                                  ts_start < int(x) < ts_end]
                        if len(ts_sc_oi) > 0:
                            datapart_stepcount = pd.DataFrame.from_dict(
                                {'ts': [int(x) for x in ts_sc_oi if x in list(data[folder]['stepCount'].keys())],
                                 'stepCount': [data[folder]['stepCount'][x] for x in ts_sc_oi]})
                        else:
                            datapart_stepcount = pd.DataFrame(columns=["ts", "stepCount"])

                        data_hr = pd.concat([data_hr, datapart_hr])
                        data_hrv = pd.concat([data_hrv, datapart_hrv])
                        data_stepcount = pd.concat([data_stepcount, datapart_stepcount])

                    df = pd.merge(data_hr, data_hrv, how="outer", on="ts")
                    df = pd.merge(df, data_stepcount, how="outer", on="ts")
                    df = df.drop_duplicates(subset="ts")
                    df = df.sort_values("ts").reset_index(drop=True)
                    df['timegroup'] = [math.floor((x - min(df['ts'])) / (3 * 60)) for x in df['ts']]

                    # df = df.set_index('datetime')
                    print("df:")
                    print(df)
                    df_sum = pd.DataFrame(df.groupby('timegroup').median()).reset_index()
                    df_sum['ts'] = [math.floor(x) for x in df_sum['ts']]
                    df_sum['datetime'] = [datetime.utcfromtimestamp(x).strftime('%Y-%m-%d %H:%M:%S') for x in
                                       df_sum["ts"]]
                    df_sum['plot_time'] = [(x - ts_start) / 3600 for x in df_sum['ts']]
                    print("df_sum:")
                    print(df_sum)

                    if (len(df_sum.loc[df_sum['heartRate'].notna(), 'heartRate']) * 60) > 16:
                        df_sum['HR_smoothed'] = df_sum['heartRate'].rolling(window=10).mean()
                        df_sum['HR_smoothed_ip'] = df_sum['HR_smoothed'].interpolate('pchip',
                                                                                     limit_direction="both")
                        x = np.asarray(df_sum['HR_smoothed_ip'])
                        b, a = butter(N=1, Wn=1 / 3600, btype='lowpass', fs=1 / 180)
                        x = filtfilt(b, a, x)
```

```
                            df_sum['HR_filt'] = x.tolist()
                            df_sum = df_sum.iloc[10:-10]
                            df_sum = df_sum.reset_index(drop=True)
                            df_sum['sleep'] = 0

                            peaks_original, _ = scipy.signal.find_peaks(np.asarray(df_sum['stepCount']))
                            if len(peaks_original) > 0:
                                for i_p in range(0, len(peaks_original) - 1, 1):
                                    peaks = peaks_original
                                    if any(df_sum.loc[peaks[i_p]:peaks[i_p] + 40, "heartRate"].isna()):
                                        peaks[i_p] = df_sum.loc[peaks[i_p]:peaks[i_p] + 40, "heartRate"].isna()[
                                                     ::-1].idxmax()
                                    if any(df_sum.loc[peaks[i_p + 1]:peaks[i_p + 1] - 40, "heartRate"].isna()):
                                        peaks[i_p + 1] = df_sum.loc[peaks[i_p + 1]:peaks[i_p + 1] - 40,
                                                         "heartRate"].isna()[
                                                         ::-1].idxmin()
                                    subset_start = df_sum.iloc[
                                                   max([peaks[i_p] - 40, 0]):min([peaks[i_p] + 40, peaks[i_p + 1],
                                                                                  len(df_sum['heartRate'])])]
                                    subset_end = df_sum.iloc[
                                                 max([peaks[i_p + 1] - 40, peaks[i_p]]):min([peaks[i_p + 1] + 40,
                                                                                              len(df_sum['heartRate'])])]
                                    if all(v > 2 for v in [subset_start['HR_filt'].count(),
                                                           subset_end['HR_filt'].count()]):
                                        try:
                                            HR_start_slope, _ = np.polyfit(np.arange(len(subset_start["HR_filt"])),
                                                                           np.asarray(subset_start["HR_filt"]), 1)
                                            HR_end_slope, _ = np.polyfit(np.arange(len(subset_end["HR_filt"])),
                                                                         np.asarray(subset_end["HR_filt"]), 1)
                                            # if HR_start_slope < 0 < HRV_start_slope and HR_end_slope > 0 > HRV_end_slope:
                                            if HR_start_slope < 0 < HR_end_slope:
                                                if any([0 < df_sum.loc[peaks[i_p + 1], "plot_time"] - x < df_sum.loc[
                                                    peaks[i_p + 1], "plot_time"] - df_sum.loc[peaks[i_p], "plot_time"] for x in
                                                        [9, 10, 11, 12, 13, 14, 15]]):
                                                    df_sum.loc[peaks_original[i_p] + 15: peaks[i_p + 1], 'sleep'] = 1
                                        except:
                                            print("defining slopes in HR at start and end of sleep failed, " +
                                                  "so we will only use the stepcount data for the night of " +
```

```
                                           str(datetime.utcfromtimestamp(ts).strftime('%Y-%m-%d')))
                                      if any([0 < df_sum.loc[peaks[i_p + 1], "plot_time"] - x < df_sum.loc[
                                         peaks[i_p + 1], "plot_time"] - df_sum.loc[peaks[i_p], "plot_time"] for x
                                             in
                                             [9, 10, 11, 12, 13, 14, 15]]):
                                         df_sum.loc[peaks_original[i_p] + 15: peaks[i_p + 1], 'sleep'] = 1
                               else:
                                   print("not sufficient HR data around start and end of sleep, " +
                                        "so we will only use the stepcount data for the night of " +
                                        str(datetime.utcfromtimestamp(ts).strftime('%Y-%m-%d')))
                                   if any([0 < df_sum.loc[peaks[i_p + 1], "plot_time"] - x < df_sum.loc[
                                      peaks[i_p + 1], "plot_time"] - df_sum.loc[peaks[i_p], "plot_time"] for x in
                                          [9, 10, 11, 12, 13, 14, 15]]):
                                      df_sum.loc[peaks_original[i_p] + 15: peaks[i_p + 1], 'sleep'] = 1

                            df_sum.loc[df_sum['stepCount'] > 0, 'sleep'] = 0
                            df_sum.loc[df_sum['heartRate'].isna(), 'sleep'] = 0
                            sleepdiff = list(np.diff(df_sum['sleep']))
                            sleepdiff_min = [0] + sleepdiff
                            sleepdiff_plus = sleepdiff + [0]
                            sleepdiff_result = [abs(x) + abs(y) for x, y in zip(sleepdiff_min, sleepdiff_plus)]
                            df_sum['sleep_corr'] = sleepdiff_result
                            # sleepdiff_min.insert(len(sleepdiff_min), 0)
                            # sleepdiff_plus.insert(0, 0)
                            # sleepdiff_result2 = [abs(x) + abs(y) for x, y in zip(sleepdiff_min, sleepdiff_plus)]

                            df_sum.loc[df_sum['sleep_corr'] > 1, 'sleep'] = 1 - df_sum.loc[
                                df_sum['sleep_corr'] > 1, 'sleep']
                            # df_sum.loc[sleepdiff_result2 > 1, 'sleep'] = 1 - df_sum.loc[sleepdiff_result2 > 1, 'sleep']

                            if len(df_sum.loc[df_sum['sleep'] == 1, 'sleep']) > 1:
                                sleepDataInstance = {str(max(df_sum.loc[df_sum['sleep'] == 1, 'ts'])): {
                                    "toBed": min(df_sum.loc[df_sum['sleep'] == 1, 'ts']),
                                    "getUp": max(df_sum.loc[df_sum['sleep'] == 1, 'ts']),
                                    "inBedDuration": (max(df_sum.loc[df_sum['sleep'] == 1, 'ts']) -
                                                      min(df_sum.loc[df_sum['sleep'] == 1, 'ts'])) / 60}}
                                sleepData.update(sleepDataInstance)
                            else:
                                print("no sleep detected for night of " +
```

```python
                                    str(datetime.utcfromtimestamp(ts).strftime('%Y-%m-%d')))
                                # sleepDataInstance = {str(ts): {
                                #    "to_bed": np.nan,
                                #    "get_up": np.nan,
                                #    "sleep_duration": np.nan}}

                        else:
                            print("no step count peaks detected around night of " +
                                  str(datetime.utcfromtimestamp(ts).strftime('%Y-%m-%d')))
                            # sleepDataInstance = {str(ts): {
                            #    "to_bed": np.nan,
                            #    "get_up": np.nan,
                            #    "sleep_duration": np.nan}}

                        plt.figure()
                        plt.plot(df_sum['plot_time'], df_sum['heartRate'], 'r', alpha=0.5)
                        plt.plot(df_sum['plot_time'], df_sum['HR_filt'], 'b', alpha=0.5)
                        #plt.plot(df_sum['plot_time'], df_sum['HR_smoothed_ip'], 'k', alpha=0.5)
                        plt.plot(df_sum['plot_time'], df_sum['stepCount'], 'k', alpha=0.5)
                        plt.plot(df_sum['plot_time'], df_sum['sleep'].multiply(-1), 'r', alpha=0.5)
                        plt.xlabel('hours since 3PM')
                        plt.xticks(np.arange(0, 24, 1))
                        plt.ylabel('HR_raw (red) - HR_filt (blue) - stepcount (black) - sleep (red)')
                        plt.savefig(
                            subject + "_HR_" + str(datetime.utcfromtimestamp(ts).strftime('%Y-%m-%d')) + ".png")
                        plt.close()

                    elif (len(df_sum.loc[df_sum['hrvSdnn'].notna(), 'hrvSdnn']) * 60) > 16:
                        peaks, _ = scipy.signal.find_peaks(np.asarray(df_sum['hrvSdnn']))
                        valleys, _ = scipy.signal.find_peaks(np.asarray(df_sum['hrvSdnn'].multiply(-1)))
                        df_sum['AC'] = np.nan
                        df_sum.loc[peaks, 'AC'] = df_sum.loc[peaks, 'hrvSdnn']
                        df_sum['DC'] = np.nan
                        df_sum.loc[valleys, 'DC'] = df_sum.loc[valleys, 'hrvSdnn']
                        df_sum['AC'] = df_sum['AC'].interpolate('pchip', limit_direction="both")
                        df_sum['DC'] = df_sum['DC'].interpolate('pchip', limit_direction="both")
                        df_sum['AC_smoothed'] = df_sum['AC'].rolling(window=20).mean()
                        df_sum['DC_smoothed'] = df_sum['DC'].rolling(window=20).mean()
                        df_sum['HRV_spread'] = df_sum['AC_smoothed'] - df_sum['DC_smoothed']
                        df_sum['HRV_smoothed'] = df_sum['hrvSdnn'].rolling(window=10).mean()
                        df_sum['HRV_smoothed_ip'] = df_sum['HRV_smoothed'].interpolate('pchip',
```

```
                                                                                     limit_direction
="both")
                            x = np.asarray(df_sum['HRV_smoothed_ip'])
                            b, a = butter(N=1, Wn=1 / 3600, btype='lowpass', fs=1 / 180)
                            x = filtfilt(b, a, x)
                            df_sum['HRV_filt'] = x.tolist()

                            df_sum = df_sum.iloc[10:-10]
                            df_sum = df_sum.reset_index(drop=True)
                            df_sum['sleep'] = 0

                            peaks_original, _ = scipy.signal.find_peaks(np.asarray(df_sum['stepCount']))
                            if len(peaks_original) > 0:
                                for i_p in range(0, len(peaks_original) - 1, 1):
                                    peaks = peaks_original
                                    if any(df_sum.loc[peaks[i_p]:peaks[i_p] + 40, "hrvSdnn"].isna()):
                                        peaks[i_p] = df_sum.loc[peaks[i_p]:peaks[i_p] + 40, "hrvSdnn"].isna()[
                                                     ::-1].idxmax()
                                    if any(df_sum.loc[peaks[i_p + 1]:peaks[i_p + 1] - 40, "hrvSdnn"].isna()):
                                        peaks[i_p + 1] = df_sum.loc[peaks[i_p + 1]:peaks[i_p + 1] - 40,
                                                         "hrvSdnn"].isna()[
                                                         ::-1].idxmin()
                                    subset_start = df_sum.iloc[
                                                   max([peaks[i_p] - 40, 0]):min([peaks[i_p] + 40, peaks[i_p + 1],
                                                                                  len(df_sum['hrvSdnn'])])]
                                    subset_end = df_sum.iloc[
                                                 max([peaks[i_p + 1] - 40, peaks[i_p]]):min([peaks[i_p + 1] + 40,
                                                                                             len(df_sum['hrvSdnn'])])]
                                    if all(v > 2 for v in [subset_start['HRV_filt'].count(),
                                                           subset_end['HRV_filt'].count()]):
                                        try:
                                            HRV_start_slope, _ = np.polyfit(np.arange(len(subset_start["HRV_filt"])),
                                                                            np.asarray(subset_start["HRV_filt"]), 1)
                                            HRV_spread_start_slope, _ = np.polyfit(
                                                np.arange(len(subset_start["HRV_spread"])),
                                                np.asarray(subset_start["HRV_spread"]), 1)
                                            HRV_end_slope, _ = np.polyfit(np.arange(len(subset_end["HRV_filt"])),
                                                                          np.asarray(subset_end["HRV_filt"]), 1)
                                            HRV_spread_end_slope, _ = np.polyfit(
                                                np.arange(len(subset_end["HRV_spread"])),
                                                np.asarray(subset_end["HRV_spread"]), 1)
                                            # if HRV_end_slope < 0 < HRV_start_slope:
                                            if any([0 < df_sum.loc[peaks[i_p + 1], "plot_time"] - x < df_sum.loc[
```

```
                        peaks[i_p + 1], "plot_time"] - df_sum.loc[peaks[i_p], "plot_time"] for x in
                          [9, 10, 11, 12, 13, 14, 15]]):
                        df_sum.loc[peaks_original[i_p] + 15: peaks[i_p + 1], 'sleep'] = 1
                    except:
                      print("defining slopes in HRV at start and end of sleep failed, " +
                         "so we will only use the stepcount data for the night of " +
                         str(datetime.utcfromtimestamp(ts).strftime('%Y-%m-%d')))
                      if any([0 < df_sum.loc[peaks[i_p + 1], "plot_time"] - x < df_sum.loc[
                        peaks[i_p + 1], "plot_time"] - df_sum.loc[peaks[i_p], "plot_time"] for x
                          in
                          [9, 10, 11, 12, 13, 14, 15]]):
                        df_sum.loc[peaks_original[i_p] + 15: peaks[i_p + 1], 'sleep'] = 1
                  else:
                    print("not sufficient HRV data around start and end of sleep, " +
                       "so we will only use the stepcount data for the night of " +
                       str(datetime.utcfromtimestamp(ts).strftime('%Y-%m-%d')))
                    if any([0 < df_sum.loc[peaks[i_p + 1], "plot_time"] - x < df_sum.loc[
                      peaks[i_p + 1], "plot_time"] - df_sum.loc[peaks[i_p], "plot_time"] for x in
                        [9, 10, 11, 12, 13, 14, 15]]):
                      df_sum.loc[peaks_original[i_p] + 15: peaks[i_p + 1], 'sleep'] = 1

                df_sum.loc[df_sum['stepCount'] > 0, 'sleep'] = 0
                df_sum.loc[df_sum['hrvSdnn'].isna(), 'sleep'] = 0
                sleepdiff = list(np.diff(df_sum['sleep']))
                sleepdiff_min = [0] + sleepdiff
                sleepdiff_plus = sleepdiff + [0]
                sleepdiff_result = [abs(x) + abs(y) for x, y in zip(sleepdiff_min, sleepdiff_plus)]
                df_sum['sleep_corr'] = sleepdiff_result
                # sleepdiff_min.insert(len(sleepdiff_min), 0)
                # sleepdiff_plus.insert(0, 0)
                # sleepdiff_result2 = [abs(x) + abs(y) for x, y in zip(sleepdiff_min, sleepdiff_plus)]

                df_sum.loc[df_sum['sleep_corr'] > 1, 'sleep'] = 1 - df_sum.loc[
                  df_sum['sleep_corr'] > 1, 'sleep']
                # df_sum.loc[sleepdiff_result2 > 1, 'sleep'] = 1 - df_sum.loc[sleepdiff_result2 > 1, 'sleep']

                if len(df_sum.loc[df_sum['sleep'] == 1, 'sleep']) > 1:
                  sleepDataInstance = {str(max(df_sum.loc[df_sum['sleep'] == 1, 'ts'])): {
```

```
                    "toBed": min(df_sum.loc[df_sum['sleep'] == 1, 'ts']),
                    "getUp": max(df_sum.loc[df_sum['sleep'] == 1, 'ts']),
                    "inBedDuration": (max(df_sum.loc[df_sum['sleep'] == 1, 'ts']) -
                                             min(df_sum.loc[df_sum['sleep'] == 1, 'ts'])) / 60}}
                  sleepData.update(sleepDataInstance)
                else:
                  print("no sleep detected for night of " +
                     str(datetime.utcfromtimestamp(ts).strftime('%Y-%m-%d')))
                  # sleepDataInstance = {str(ts): {
                  #    "to_bed": np.nan,
                  #    "get_up": np.nan,
                  #    "sleep_duration": np.nan}}

              else:
                print("no step count peaks detected around night of " +
                   str(datetime.utcfromtimestamp(ts).strftime('%Y-%m-%d')))
                # sleepDataInstance = {str(ts): {
                #    "to_bed": np.nan,
                #    "get_up": np.nan,
                #    "sleep_duration": np.nan}}

              plt.figure()
              plt.plot(df_sum['plot_time'], df_sum['hrvSdnn'], 'r', alpha=0.5)
              plt.plot(df_sum['plot_time'], df_sum['HRV_filt'], 'b', alpha=0.5)
              #plt.plot(df_sum['plot_time'], df_sum['AC_smoothed'], 'm', alpha=0.5)
              #plt.plot(df_sum['plot_time'], df_sum['DC_smoothed'], 'y', alpha=0.5)
              #plt.plot(df_sum['plot_time'], df_sum['HRV_smoothed_ip'], 'k', alpha=0.5)
              #plt.plot(df_sum['plot_time'], df_sum['HRV_spread'], 'b', alpha=0.5)
              plt.plot(df_sum['plot_time'], df_sum['stepCount'], 'k', alpha=0.5)
              plt.plot(df_sum['plot_time'], df_sum['sleep'].multiply(-1), 'r', alpha=0.5)
              plt.xlabel('hours since 3PM')
              plt.xticks(np.arange(0, 24, 1))
              plt.ylabel('SDNN_raw (red) - SDNN_filt (blue) - stepcount (black) - sleep (red)')
              plt.savefig(
                subject + "_HRV_" + str(datetime.utcfromtimestamp(ts).strftime('%Y-%m-%d')) + ".png")
              plt.close()

            else:
              print("there are less than 16 hours of hrv / hr data available for the night of " +
                  str(datetime.utcfromtimestamp(ts).strftime('%Y-%m-%d')))
              df_sum = df_sum.iloc[10:-10]
```

```
df_sum = df_sum.reset_index(drop=True)
df_sum['sleep'] = 0

if (len(df_sum.loc[df_sum['stepCount'].notna(), 'stepCount']) * 60) > 16:
    peaks_original, _ = scipy.signal.find_peaks(np.asarray(df_sum['stepCount']))
    if len(peaks_original) > 0:
        activity_times = df_sum.loc[peaks_original, "plot_time"]
        max_rest_duration = np.max(np.diff(activity_times))
        for i_p in range(0, len(peaks_original) - 1, 1):
            peaks = peaks_original
            if df_sum.loc[peaks[i_p + 1], "plot_time"] - df_sum.loc[
                peaks[i_p], "plot_time"] == max_rest_duration:
                if any([0 < df_sum.loc[peaks[i_p + 1], "plot_time"] - x < df_sum.loc[
                    peaks[i_p + 1], "plot_time"] - df_sum.loc[peaks[i_p], "plot_time"] for x in
                        [9, 10, 11, 12, 13, 14, 15]]):
                    df_sum.loc[peaks_original[i_p] + 15: peaks[i_p + 1], 'sleep'] = 1

        df_sum.loc[df_sum['stepCount'] > 0, 'sleep'] = 0
        if len(df_sum.loc[df_sum['sleep'] == 1, 'sleep']) > 1:
            sleepDataInstance = {str(max(df_sum.loc[df_sum['sleep'] == 1, 'ts'])): {
                "toBed": min(df_sum.loc[df_sum['sleep'] == 1, 'ts']),
                "getUp": max(df_sum.loc[df_sum['sleep'] == 1, 'ts']),
                "inBedDuration": (max(df_sum.loc[df_sum['sleep'] == 1, 'ts']) -
                                  min(df_sum.loc[df_sum['sleep'] == 1, 'ts'])) / 60}}
            sleepData.update(sleepDataInstance)
        else:
            print("no sleep detected for night of " +
                  str(datetime.utcfromtimestamp(ts).strftime('%Y-%m-%d')))
    else:
        print("no step count peaks detected around night of " +
              str(datetime.utcfromtimestamp(ts).strftime('%Y-%m-%d')))
else:
    print("there are less than 16 hours of stepcount data available for the night of " +
          str(datetime.utcfromtimestamp(ts).strftime('%Y-%m-%d')))

plt.figure()
plt.plot(df_sum['plot_time'], df_sum['stepCount'], 'k', alpha=0.5)
plt.plot(df_sum['plot_time'], df_sum['sleep'].multiply(-1), 'r', alpha=0.5)
plt.xlabel('hours since 3PM')
plt.xticks(np.arange(0, 24, 1))
```

```python
                            plt.ylabel('stepcount (black) - sleep (red)')
                            plt.savefig(
                                subject + "_steps_" + str(datetime.utcfromtimestamp(ts).strftime('%Y-%m-%d')) + ".png")
                            plt.close()

                        df_sum.to_csv(subject + "_" + str(datetime.utcfromtimestamp(ts).strftime('%Y-%m-%d')) + ".csv")

                    else:
                        print("there are no related datafolders detected for the night of " +
                              str(datetime.utcfromtimestamp(ts).strftime('%Y-%m-%d')))
                        # sleepDataInstance = {str(ts): {
                        #    "to_bed": np.nan,
                        #    "get_up": np.nan,
                        #    "sleep_duration": np.nan}}

                    print("sleepData")
                    print(sleepData)

                    # TODO: add sleep quality estimation code based on heartRate, hrvSdnn, stepCount:
                    # TODO: - Look for 15-min periods with very high sd in heartRate / stable drop in hrvSDnn and/or stepCount>0 -> awake periods
                    # TODO: - Compute following features to link to reported sleep quality:
                    # TODO:       - Ratio awake (sum of awake periods / Sleep Duration)
                    # TODO:       - Average / Median / sd of 1-min SDNN from periods asleep
                    # TODO:       - total power of (ultradian) frequency spectrum in 1-min HR and SDNN from periods asleep (stretch goal)
                    # TODO: - write away features to a dict and make sure that the dicts in else statements match

            else:
                print("There are no consecutive dates")
        else:
            print("There are no 2 dates or more")

        return sleepData

    def getSleepData(self, data, subject):
        keys = list(data.keys())
        ts = []
        for key in keys:
            new_hr_ts = list(data[key]['heartRate'].keys())
            new_hrv_ts = list(data[key]['hrvSdnn'].keys())
            new_steps_ts = list(data[key]['stepCount'].keys())
            ts = ts + new_hr_ts + new_hrv_ts + new_steps_ts
        ts = [int(x) for x in ts]
        ts.sort()
        dates = [datetime.utcfromtimestamp(x).strftime('%Y-%m-%d') for x in ts]
        dates_set = list(set(dates))
        dates_set.sort()
        print("dates:")
        print(dates_set)
        sleepDf = self.computeSleep(dates_set, data, subject)
```

```python
        return sleepDf


def process_blob():
    # The script is based on generating a SAS token to use for data access.
    # first "get shared access signature", then select the options you desire for permissions and other parameters and click "create".
    # In the connection string field, you will find a part that says "BlobEndpoint=https....". Everything from the "https:" till ".net/" is put below as account_url.
    # For filling in the sas_token variable below you should use the copy button below the "SAS token" field.
    # Check if your account_url indeed starts with "https://" and ends with ".net/"
    # Check if your sas_token starts with a ?.
    # You are good to go and use this script.
    # STUDY ENVIRONMENT PARAMETERS


# Om de account_url en de sas_token te krijgen moet je Alex van Kraaij (imec) <Alex.vanKraaij@imec.nl> mailen
    account_url = "" # Mail Alex
    sas_token = "" # Mail Alex
    garmin = Garmin(account_url, sas_token)


    # list all blobs per specific folder "garmindata" and "sensordata"
    sensordata_blob_list_all, garmin_blob_list_all = garmin.getAllBlobNames()
    # Check which subjects are in the azure container "participants" and have sensordata
    subject_list = list(set([i.split("/")[0] for i in sensordata_blob_list_all]))
    in_bed_by_subject={}
    for subject in subject_list:
        sensordata_blob_list = [i for i in sensordata_blob_list_all if subject in i]
        garmin_blob_list = [i for i in garmin_blob_list_all if subject in i]
        # Check which blobs are in sensordata_blob_list and not in garmindata_blob_list
        new_zip_blobs = garmin.getNewBlobNames(sensordata_blob_list, garmin_blob_list)
        print("subject:")
        print(subject)
        print("new_zip_blobs:")
        print(new_zip_blobs)
        if len(new_zip_blobs) > 0:
            new_blobs = garmin.unzipGarminFiles(new_zip_blobs, subject)
            operating_system = garmin.getOperatingSystem(new_blobs[0])
            if operating_system == "ios":
                garmin_dict_list = garmin.readAllJsonIos(new_blobs)
            elif operating_system == "android":
                garmin_dict_list = garmin.readAllJsonAndroid(new_blobs)
            else:
                raise OSError("The operating system is unknown. The options are android or ios.")
            existing_data = garmin.readExistingData(subject)
            print("existing_data:")
            print(existing_data)
            garmin_data = garmin.addNewData(existing_data, garmin_dict_list, new_zip_blobs)
            print("final_data:")
            print(garmin_data)
            sleep_dict = garmin.getSleepData(garmin_data, subject)
            print(sleep_dict)
```

```
            triggered_events = {"1728546000": {"referenceId": "questionnaire_a"},
                                "1828546000": {"referenceId": "questionnaire_b"}}
            subject_in_bed = {}
            for ts_key, entry in sleep_dict.items():
                try:
                    date = datetime.utcfromtimestamp(int(ts_key)).strftime("%Y-%m-%d")
                    subject_in_bed[date] = entry["inBedDuration"]
                except:
                    continue
            if subject_in_bed:
                in_bed_by_subject[subject] = subject_in_bed
            last_updated = list(garmin_data.keys())[-1].split('_')[1]

            final_data = {'garminData': garmin_data,
                          'sleepData': sleep_dict,
                          'triggeredEvents': triggered_events,
                          'lastUpdated': last_updated}
            print(final_data)

            filename = subject + "_data.json"
            final_json = json.dumps(final_data)
            blob_client = garmin.blob_service_client.get_blob_client(container="vivicadata", blob=filename)
            blob_client.upload_blob(final_json, overwrite=True)
    return in_bed_by_subject

            # sleep_json = json.dumps(sleep_dict)
            # filename = subject + "_sleep.json"
            # blob_client = garmin.blob_service_client.get_blob_client(container="vivicadata", blob=filename)
            # blob_client.upload_blob(sleep_json, overwrite=True)

            # TODO: test script in AzureML environment [Alex]
##### Eind garmin script
##### Begin questionnaire script



# The script is based on generating a SAS token to use for data access.
# Below are the parameters you can use to access the data.

# Om de account_url en de sas_token te krijgen moet je Alex van Kraaij (imec) <Alex.vanKraaij@imec.nl> mailen
account_url_questionnaire = "" # Mail Alex, dit is zelfde account_url als voorheen in het script
sas_token_questionnaire = "" # Mail Alex, dit is zelfde sas token als voorheen in het script
blobServiceClient = BlobServiceClient(account_url=account_url_questionnaire, credential=sas_token_questionnaire)

# Here you can specify which subject you are interested in
subjects = ["p024", "p029"]

def get_questionnaire_of_subject(blob_service_client, subject, questionnaire, container="participants"):
    container_client = blob_service_client.get_container_client(container=container)
    blob_names_list = list(container_client.list_blob_names())
    subject_blob_list = [i for i in blob_names_list if subject in i]
```

```
    questionnaire_blob_name = [i for i in subject_blob_list if questionnaire in i][0]
    blobClient = blob_service_client.get_blob_client(container=container, blob=questionnaire_blob_name)

    with BytesIO() as inputBlob:
        blobClient.download_blob().readinto(inputBlob)
        inputBlob.seek(0)
        questionnaire_data = inputBlob.readlines()

        questionnaire_dict_list = []
        for i, line in enumerate(questionnaire_data):
            try:
                obj = json.loads(line.decode('utf-8'))
                ts = obj.get("timestamps", {})
                if all(k in ts for k in ["opened", "submitted", "uploaded"]):
                    entry_date = datetime.fromtimestamp(ts["submitted"], tz=pytz.UTC).strftime("%Y-%m-%d")
                    obj["date"] = entry_date
                    questionnaire_dict_list.append(obj)

                else:
                    print(f"  Skipped incomplete entry at line {i}: missing timestamp fields")
            except Exception as e:
                print(f"  Error parsing line {i}: {e}")

    return questionnaire_dict_list


def extract_combined_answers(subject):
    combined_answers = []

    # Get the most recent PSQI answer (last entry's last answer)
    try:
        psqi_data = get_questionnaire_of_subject(blobServiceClient, subject, "PSQI")
        if psqi_data:
            latest_psqi_entry = psqi_data[-1]
            last_answer = latest_psqi_entry['answers'][-1]['answer']
            combined_answers.append(last_answer)
    except Exception as e:
        print(f"[{subject}] Failed to process PSQI: {e}")

    # Get all dailySleepQ answers with complete timestamps
    try:
        sleep_data = get_questionnaire_of_subject(blobServiceClient, subject, "dailySleepQ")
        for entry in sleep_data:
            if all(k in entry.get("timestamps", {}) for k in ["opened", "submitted", "uploaded"]):
                answers = [a['answer'] for a in entry['answers']]
                combined_answers.extend(answers)
    except Exception as e:
        print(f"[{subject}] Failed to process dailySleepQ: {e}")

    return combined_answers

def build_goal_timeline_for_subject(subject):
    try:
```

```
        goals_data = get_questionnaire_of_subject(blobServiceClient, subject, "Goals")
        if goals_data:
            timeline = []
            for entry in goals_data:
                try:
                    date = datetime.fromtimestamp(entry["timestamps"]["submitted"], tz=pytz.UTC).date()
                    goal_value = float(entry["answers"][0]["answer"].replace(",", "."))
                    timeline.append((date, goal_value))
                except Exception as e:
                    print(f"[{subject}] Failed parsing Goals entry: {e}")
            timeline.sort()
            return timeline
    except Exception as e:
        print(f"[{subject}] Failed to get goal data: {e}")
    return []

def get_goal_for_date(goal_timeline, target_date):
    applicable_goals = [g for (d, g) in goal_timeline if d <= target_date]
    return applicable_goals[-1] if applicable_goals else None

def calculate_score(duration, main_goal):
    try:
        duration = float(duration)
    except:
        return 0
    time_diff = abs(duration - (main_goal * 60))
    if time_diff <= 15:
        return 5
    elif time_diff <= 60:
        return 4
    elif time_diff <= 120:
        return 3
    elif time_diff <= 180:
        return 2
    else:
        return 1


def score_daily_sleep_answers(answers):
    score_daily_questionnaire = 0

    try:
        val = int(answers[0])
        if 0 <= val <= 19:
            score_daily_questionnaire += 1
        elif 20 <= val <= 39:
            score_daily_questionnaire += 2
        elif 40 <= val <= 59:
            score_daily_questionnaire += 3
        elif 60 <= val <= 79:
            score_daily_questionnaire += 4
        elif 80 <= val <= 100:
            score_daily_questionnaire += 5
    except:
        pass

    if answers[1].strip().lower() == "ja":
```

```
        score_daily_questionnaire -= 0.4

    q3_map = {
        "ja, een keer": -0.2,
        "ja, twee keer": -0.4,
        "ja, drie keer": -0.6,
        "ja, meer dan drie keer": -0.8,
        "niet gedurende afgelopen nacht": 0
    }
    score_daily_questionnaire += q3_map.get(answers[5].strip().lower(), 0)

    q4_map = {
        "een klein beetje een probleem": -0.2,
        "enigszins een probleem": -0.4,
        "behoorlijk een probleem": -0.6,
        "een groot probleem": -0.8,
        "helemaal geen probleem": 0
    }
    score_daily_questionnaire += q4_map.get(answers[2].strip().lower(), 0)

    return score_daily_questionnaire

def calculate_in_bed_from_questionnaire(answers):
    try:
        # Parse Q4 en Q5 als tijd (formaat bijv. '23:45' of '07:15')
        to_bed_raw = datetime.strptime(answers[3], "%H:%M")
        # Adjust if someone enters something between 6:00 and 15:00 as 'to bed' (likely meant PM)
        if 6 <= to_bed_raw.hour <= 15:
            to_bed = to_bed_raw - timedelta(hours=12) # e.g. 10:55 becomes 22:55
        else:
            to_bed = to_bed_raw
        get_up = datetime.strptime(answers[4], "%H:%M")

        if get_up <= to_bed:
            get_up += timedelta(days=1)

        in_bed_duration = (get_up - to_bed).total_seconds() / 60 # in minuten

        # Check op Q7 (alleen als Q6 != 'niet gedurende afgelopen nacht')
        if answers[5].strip().lower() != "niet gedurende afgelopen nacht" and len(answers) >= 7:
            try:
                awake_minutes = int(answers[6])
                in_bed_duration -= awake_minutes
            except:
                pass

        return max(in_bed_duration, 0)
    except Exception as e:
        print(f"Fallback berekening mislukt: {e}")
        return None

def generate_recommendation(avg_sleep_score, avg_daily_score, avg_total_score):
    recs = []

    # Sleep duration part
    if avg_total_score == 5:
```

```python
        return ("You're doing an excellent job with your sleep! Your sleep duration and quality are in a great place. "
                "Keep up your healthy sleep habits to maintain this balance! If you're feeling confident, you could even "
                "challenge yourself by setting a slightly more ambitious sleep goal to explore your optimal sleep and see "
                "how it feels. Keep it up!")

    if avg_total_score <= 4:
        # Sleep duration recommendation
        if avg_sleep_score < 2:
            rec = ("It seems like you are facing some sleep issues, and it can impact how you feel overall. "
                   "It might be a good idea to talk to a professional to understand what's going on. In the meantime, try to set either a realistic sleep duration goal or adjust your bedtime or wake-up time gradually to align better with your goal.")
        elif avg_sleep_score == 2:
            rec = ("Your sleep 'isnt quite on track right now. Over the last five days, your sleep has been about two hours over or under your goal, which can impact how you feel during the day. Try gradually adjusting your bedtime or wake-up time to better match your sleep goal. If needed, consider adjusting your target sleep duration goal to find what works best for you!!")
        elif avg_sleep_score == 3:
            rec = ("You're on the right track, but 'theres still a little room to improve your sleep duration. Over the last five days, your sleep has been about an hour over or under your goal, which can affect how you feel during the day. Hitting the perfect sleep schedule every night 'isnt easy, but small -adjustmentslike gradually shifting your bedtime or wake-up time over the -weekscan help you stay on track"!")
        elif avg_sleep_score < 4:
            rec = ("Over the last five days, your sleep has been slightly outside your ideal duration. To improve consistency, try adjusting your bedtime or wake-up time by 30 to 60 minutes. ")
        else:
            rec = "Your sleep duration currently aligns well with your stated goal. This consistency indicates effective time management and attention to healthy sleep habits. Continuing to monitor and maintain this pattern may contribute positively to your overall well-being."
        recs.append(rec)

        # Sleep quality recommendation
        if avg_daily_score < 2:
            rec = ("It looks like your sleep quality needs some major improvements. To improve your sleep quality, try turning off screens before bed, making your bedroom more comfortable, and going to bed at the same time each night. If you wake up often during the night, try drinking less water before going to bed and cutting out caffeine late in the day. A relaxing bedtime routine, like reading or deep breathing, can also help you sleep better. If poor sleep continues, it might be a good idea to talk to a professional for more help".")
        elif avg_daily_score == 2:
            rec = ("Your sleep isn't as restful as it could be. To improve your sleep quality, try turning off screens before bed, making your bedroom more comfortable, and going to bed at the same time each night. If you wake up often during the night, try drinking less water before going to bed and cutting out caffeine late in the day. A relaxing bedtime routine, like reading or deep breathing, can also help you sleep better and wake up feeling more refreshed! ")
        elif avg_daily_score == 3:
```

```
            rec = ("Your sleep quality is decent, but a few small changes could make it even better. Try limiting screen time before bed, adjusting your sleep environment for comfort, or cutting back on fluids in the evening if you wake up frequently at night. Stay consistent, make small changes, and over time, you'll find the routine that works best for you!")
        elif avg_daily_score == 4:
            rec = ("Your sleep quality is good, but there is some room for improvement! Keep up a consistent bedtime and try incorporating a relaxing activity before sleep, like reading or deep breathing, to wake up feeling even more refreshed.")
        else:
            rec = "Your reported sleep quality is at a desirable level. This outcome suggests that current sleep-related behaviors and environmental factors may be supporting restful sleep. Ongoing attention to these elements can help preserve this favorable pattern over time."
        recs.append(rec)

    return " ".join(recs)



if __name__ == "__main__":
    all_subjects_answers = {}
    main_sleep_goal_by_subject = {}
    for subject in subjects:
        answers = extract_combined_answers(subject)
        all_subjects_answers[subject] = answers

    in_bed_durations_per_subject = process_blob()

    # Store scores for later matching
    sleep_scores_by_subject = {}
    questionnaire_scores_by_subject = {}

    for subject in subjects:
        sleep_scores = []
        daily_scores = []
        last_answer_value = None


            # Main sleep goal (from Goals questionnaire)
        goal_timeline = build_goal_timeline_for_subject(subject)
        main_sleep_goal_by_subject[subject] = goal_timeline[-1][1] if goal_timeline else None # voor referentie


        # Sleep durations
        durations = in_bed_durations_per_subject.get(subject, [])
        for duration in durations:
            if last_answer_value is not None:
                score = calculate_score(duration, last_answer_value)
                sleep_scores.append(score)

        # Questionnaire scores
        try:
            sleep_data = get_questionnaire_of_subject(blobServiceClient, subject, "dailySleepQ")
            for entry in sleep_data:
```

```python
                if not all(k in entry.get("timestamps", {}) for k in ["opened", "submitted", "uploaded"]):
                    continue # Skip incomplete entries
                answers = [a['answer'] for a in entry['answers']]
                if len(answers) >= 4:
                    score = score_daily_sleep_answers(answers)
                    daily_scores.append(score)

        except Exception as e:
            print(f"[{subject}] Failed to get dailySleepQ data: {e}")

        sleep_scores_by_subject[subject] = sleep_scores
        questionnaire_scores_by_subject[subject] = daily_scores

    # --- Combine latest unused sleep & questionnaire scores per subject ---
    combined_scores_per_subject = {}

    for subject in subjects:
        sleep_data = in_bed_durations_per_subject.get(subject, {})
        daily_entries = []
        try:
            raw_daily_data = get_questionnaire_of_subject(blobServiceClient, subject, "dailySleepQ")
            daily_entries = [
                entry for entry in raw_daily_data
                if all(k in entry.get("timestamps", {}) for k in ["opened", "submitted", "uploaded"])
            ]
        except Exception as e:
            print(f"[{subject}] Failed to load dailySleepQ entries: {e}")

        try:
            goals_data = get_questionnaire_of_subject(blobServiceClient, subject, "Goals")
            if goals_data:
                latest_goal_entry = goals_data[-1]
                raw_goal = latest_goal_entry['answers'][0]['answer'].replace(",", ".")
                main_goal_value = float(raw_goal)
                main_sleep_goal_by_subject[subject] = main_goal_value
        except Exception as e:
            print(f"[{subject}] Failed to get sleep goal from 'Goals' questionnaire: {e}")
            main_goal_value = None

        combined_scores = []
        goal_timeline = build_goal_timeline_for_subject(subject)
        main_sleep_goal_by_subject[subject] = goal_timeline[-1][1] if goal_timeline else None # laatste voor referentie
        combined_scores = []

        for entry in daily_entries:
            if not all(k in entry.get("timestamps", {}) for k in ["opened", "submitted", "uploaded"]):
                continue

            entry_date_str = entry.get("date")
            entry_date = datetime.strptime(entry_date_str, "%Y-%m-%d").date()

```

```
            #  Haal juiste doel op voor deze datum
            main_goal_value = get_goal_for_date(goal_timeline, entry_date)

            answers = [a["answer"] for a in entry["answers"]]
            daily_score = score_daily_sleep_answers(answers)
            note = ""
            in_bed = None

            #  Gebruik juiste doel bij berekenen van slaapscore
            if entry_date_str in sleep_data:
                in_bed = sleep_data[entry_date_str]
                sleep_score = calculate_score(in_bed, main_goal_value)
            else:
                fallback = calculate_in_bed_from_questionnaire(answers)
                if fallback:
                    in_bed = fallback
                    sleep_score = calculate_score(fallback, main_goal_value)
                    note = f"Fallback sleep duration used: {round(fallback)} min ({round(fallback/60,2)} hrs)"
                else:
                    sleep_score = 0
                    note = "No sleep data available"

            total_score = (sleep_score * 0.5) + (daily_score * 0.5)

            combined_scores.append({
                "date": entry_date_str,
                "sleep_score": sleep_score,
                "daily_score": daily_score,
                "total_score": total_score,
                "in_bed_duration": in_bed,
                "note": note,
                "goal_used": main_goal_value #  optioneel voor in Excel
            })

        combined_scores_per_subject[subject] = combined_scores

        # for entry in daily_entries:
        #     entry_date = entry.get("date")
        #     answers = [a["answer"] for a in entry["answers"]]
        #     daily_score = score_daily_sleep_answers(answers)
        #     note = ""
        #     in_bed = None

        #     if entry_date in sleep_data:
        #         in_bed = sleep_data[entry_date]
        #         sleep_score = calculate_score(in_bed, main_goal_value)
        #     else:
        #         fallback = calculate_in_bed_from_questionnaire(answers)
        #         if fallback:
        #             in_bed = fallback
        #             sleep_score = calculate_score(fallback, main_goal_value)
        #             note = f"Fallback sleep duration used: {round(fallback)} min ({round(fallback/60,2)} hrs)"
        #         else:
        #             sleep_score = 0
        #             note = "No sleep data available"

```

```
        #    total_score = (sleep_score * 0.5) + (daily_score * 0.5)

        #    combined_scores.append({
        #        "date": entry_date,
        #        "sleep_score": sleep_score,
        #        "daily_score": daily_score,
        #        "total_score": total_score,
        #        "in_bed_duration": in_bed,
        #        "note": note
        #    })

        # combined_scores_per_subject[subject] = combined_scores


    # --- Output ---
        for subject, scores in combined_scores_per_subject.items():
            print(f"\n--- {subject} Combined Daily Scores ---")
            total_so_far = 0

        # Laad alle dagelijkse antwoorden vooraf
            try:
                daily_data = get_questionnaire_of_subject(blobServiceClient, subject, "dailySleepQ")
            except Exception as e:
                print(f"[{subject}] Failed to load dailySleepQ entries for printing: {e}")
                daily_data = []

            print(f"\n--- {subject} Combined Daily Scores ---")
            total_so_far = 0

            # Print per dag
            # Maak mapping per datum
            score_by_date = {entry["date"]: entry for entry in scores}

            # Combineer alle datums
            dates_questionnaire = [e["date"] for e in daily_data]
            dates_sleep = list(in_bed_durations_per_subject.get(subject, {}).keys())
            all_dates = sorted(set(dates_questionnaire + dates_sleep))

            print(f"\n--- {subject} Combined Daily Scores (by date) ---")

            for i, date in enumerate(all_dates):
                entry = score_by_date.get(date, {})
                sleep_score = entry.get("sleep_score", "")
                daily_score = entry.get("daily_score", "")
                total_score = entry.get("total_score", "")
                in_bed = entry.get("in_bed_duration", "")
                note = entry.get("note", "No questionnaire data available for this day")

                print(f"\n === {date} ===")
                print(f"  Sleep score: {sleep_score if sleep_score != '' else -''}")
                print(f"  Daily questionnaire score: {daily_score if daily_score != '' else -''}")
                print(f"  Total score: {total_score if total_score != '' else -''}")

                if in_bed != "":
                    hours = round(in_bed / 60, 2)
```

```
                    print(f" In-bed duration: {round(in_bed)} minutes ({hours} hours)")
                else:
                    print(" In-bed duration: Not available")

                print(f" Note: {note}")

                # Toon vragen en antwoorden als beschikbaar
                daily_entry = next((e for e in daily_data if e["date"] == date), None)
                if daily_entry:
                    answers = daily_entry["answers"]
                    questions = {
                        q["id"]: q["description"]
                        for q in daily_entry.get("questionnaire", [{}])[0].get("questions", [])
                    }

                    print("\n Daily questionnaire and answers:")
                    for qa in answers:
                        q_id = qa.get("id")
                        question = questions.get(q_id, f"Unknown question (id: {q_id})")
                        answer = qa.get("answer", "No answer")
                        print(f"  - {question}: {answer}")
                else:
                    print("  No questionnaire available for this day.")

            # Print aanbeveling op basis van gemiddelden
            if scores:
                avg_sleep_score = sum([entry["sleep_score"] for entry in scores if "sleep_score" in entry]) / len(scores)
                avg_daily_score = sum([entry["daily_score"] for entry in scores if "daily_score" in entry]) / len(scores)
                avg_total_score = sum([entry["total_score"] for entry in scores if "total_score" in entry]) / len(scores)

                recommendation = generate_recommendation(avg_sleep_score, avg_daily_score, avg_total_score)

                print("\n === Recommendations based on your averages ===")
                print(recommendation)
                print("  ==============================================\n")

                for entry in scores:
                    entry["recommendation"] = recommendation

            # Voeg vragen en antwoorden toe als beschikbaar
                if i < len(daily_data):
                    try:
                        entry = daily_data[i]
                        answers = entry["answers"]
                        question_map = {q["id"]: q["description"] for q in entry.get("questionnaire", [{}])[0].get("questions", [])}

                        print("\n Daily questionnaire and answers:")
                        for qa in answers:
                            q_id = qa.get("id")
                            question = question_map.get(q_id, f"Unknown question (id: {q_id})")
                            answer = qa.get("answer", "No answer")
```

```python
                            print(f"  - {question}: {answer}")

                    except Exception as e:
                        print(f"   Error while retrieving questions/answers: {e}")
                else:
                    print(" No daily questionnaire available for this day.")
                print("  ------------------------------\n")




# Maak ExcelWriter om meerdere tabs/sheets aan te maken
output_file = "combined_sleep_scores_per_subject_day15_all.xlsx"
with pd.ExcelWriter(output_file, engine='openpyxl') as writer:
    for subject, scores in combined_scores_per_subject.items():
        if not scores:
            continue

        subject_rows = []

        # Stap 1: Laad vragenlijstdata + dates
        daily_data_raw = get_questionnaire_of_subject(blobServiceClient, subject, "dailySleepQ")
        daily_data = [
            e for e in daily_data_raw
            if all(k in e.get("timestamps", {}) for k in ["opened", "submitted", "uploaded"])
        ]
        dates_questionnaire = [e["date"] for e in daily_data]

        # Stap 2: Haal alle dagen met slaapmeting (die dict is {date: duration})
        sleep_data_dict = in_bed_durations_per_subject.get(subject, {})

        # Stap 3: Combineer alle dagen waarin iets zit
        all_dates = sorted(set(dates_questionnaire + list(sleep_data_dict.keys())))

        # Stap 4: Bouw scores per datum (inclusief alleen-metingsdagen)
        score_by_date = {}

        for e in scores:
            score_by_date[e["date"]] = e

        for date, duration in sleep_data_dict.items():
            if date not in score_by_date:
                sleep_score = calculate_score(duration, main_sleep_goal_by_subject.get(subject, 8))
                score_by_date[date] = {
                    "date": date,
                    "sleep_score": sleep_score,
                    "daily_score": "",
                    "total_score": round(sleep_score * 0.5, 2),
                    "note": "No questionnaire data available for this day",
                    "in_bed_duration": duration,
                    "recommendation": "",
                }

        # Stap 5: Maak per dag de rijen aan
        for date in all_dates:
```

```
            entry = score_by_date.get(date, {})
            row = {
                "Goal used (hours)": entry.get("goal_used", ""),
                "Subject": subject,
                "Main Goal (hours)": main_sleep_goal_by_subject.get(subject, None),
                "Date": date,
                "Sleep Score": entry.get("sleep_score", ""),
                "Daily Score": entry.get("daily_score", ""),
                "Total Score": entry.get("total_score", ""),
                "Average Total Score": "",
                "In-bed (minutes)": round(entry["in_bed_duration"], 2) if entry.get("in_bed_duration") else "",
                "In-bed (hours)": round(entry["in_bed_duration"] / 60, 2) if entry.get("in_bed_duration") else "",
                "Note": entry.get("note", "No questionnaire data available for this day"),
                "Recommendation": entry.get("recommendation", "")
            }

            # Voeg vragen en antwoorden toe
            daily_entry = next((e for e in daily_data if e["date"] == date), None)
            if daily_entry:
                answers = daily_entry["answers"]
                questions = {
                    q["id"]: q["description"]
                    for q in daily_entry.get("questionnaire", [{}])[0].get("questions", [])
                }
                for i, qa in enumerate(answers, start=1):
                    q_id = qa.get("id")
                    row[f"Q{i}"] = questions.get(q_id, f"Unknown question (id: {q_id})")
                    row[f"A{i}"] = qa.get("answer", "No answer")

            # Bereken voortschrijdend gemiddelde
            valid_totals = [
                r["Total Score"] for r in subject_rows
                if isinstance(r.get("Total Score", None), (int, float, float))
            ]
            if isinstance(row["Total Score"], (int, float)):
                valid_totals.append(row["Total Score"])
            if valid_totals:
                row["Average Total Score"] = round(sum(valid_totals) / len(valid_totals), 2)

            subject_rows.append(row)

        if subject_rows:
            df = pd.DataFrame(subject_rows)
            df.to_excel(writer, sheet_name=subject, index=False)

print(f"\n Excel file saved as: {output_file} (met correcte matching en lege cellen waar nodig)")
```

L ALGORITHM - PYTHON SCRIPT - PSQI SCORE

Listing L.1: PSQI Scoring Algorithm

```python
#Copyright (c), 2025, OnePlanet Research Center & University of Twente

# Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

# The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.


#Script from Alex originally, adjusted to give scores to PSQI questionnaire
import json
from azure.storage.blob import BlobServiceClient
from io import BytesIO
import pandas as pd
from datetime import datetime, timedelta

# Azure instellingen
account_url = "" #mail Alex
sas_token = "" #mail Alex
blobServiceClient = BlobServiceClient(account_url=account_url, credential=sas_token)
# Om de account_url en de sas_token te krijgen moet je Alex van Kraaij (imec) <Alex.vanKraaij@imec.nl> mailen

# Subjectlijst
deelnemers = ["p001", "p002"]

# Mappingfuncties
def freq_to_score(answer):
    return {
        "niet gedurende deze maand": 0,
        "minder dan 1 keer per week": 1,
        "1 tot 2 keer per week": 2,
        "3 of meer keren per week": 3
    }.get(answer, 0)

def quality_to_score(answer):
    return {
        "heel goed": 0,
```

```
        "redelijk goed": 1,
        "redelijk slecht": 2,
        "heel slecht": 3
    }.get(answer, 0)

def problem_to_score(answer):
    return {
        "helemaal geen probleem": 0,
        "een klein probleem": 1,
        "enigszins een probleem": 2,
        "een groot probleem": 3
    }.get(answer, 0)

# PSQI-scoreberekening
def calculate_psqi_score(answers):
    answer_dict = {a['id']: a['answer'] for a in answers}

    comp1 = quality_to_score(answer_dict.get("6", ""))

    # Component 2 - slaaplatentie
    try:
        latentie = int(answer_dict.get("2", "0"))
    except ValueError:
        latentie = 0
    comp2a = 0 if latentie <= 15 else 1 if latentie <= 30 else 2 if latentie <= 60 else 3
    comp2b = freq_to_score(answer_dict.get("5a", ""))
    comp2 = round((comp2a + comp2b) / 2)

    # Component 3 - slaapduur
    try:
        slaapduur = float(answer_dict.get("4", "0").replace(",", "."))
    except ValueError:
        slaapduur = 0
    comp3 = 0 if slaapduur >= 7 else 1 if slaapduur >= 6 else 2 if slaapduur >= 5 else 3

    # Component 4 - Slaapefficiëntie
    try:
        slaapduur = float(answer_dict.get("4", "0").replace(",", "."))

        bed_str = answer_dict.get("1", "00:00")
        wake_str = answer_dict.get("3", "00:00")

        bed_time = datetime.strptime(bed_str, "%H:%M").time()
        wake_time = datetime.strptime(wake_str, "%H:%M").time()

        # Forceer bedtijd als avondtijd indien nodig
        if bed_time.hour < 12:
            bed_time = (datetime.combine(datetime.today(), bed_time) + timedelta(hours=12)).time()

        bed_dt = datetime.combine(datetime.today(), bed_time)
        wake_dt = datetime.combine(datetime.today(), wake_time)

        if wake_dt <= bed_dt:
            wake_dt += timedelta(days=1)

        time_in_bed = (wake_dt - bed_dt).total_seconds() / 3600
        efficiency = (slaapduur / time_in_bed) * 100 if time_in_bed > 0 else 0
        comp4 = 0 if efficiency >= 85 else 1 if efficiency >= 75 else 2 if efficiency >= 65 else 3

```

```python
    except Exception as e:
        print(f" Fout bij berekenen Component 4: {e}")
        comp4 = 0



    # Component 5 - slaapproblemen
    keys = ["5b", "5c", "5d", "5e", "5f", "5g", "5h", "5i", "5j1"]
    comp5_sum = sum([freq_to_score(answer_dict.get(k, "")) for k in keys])
    comp5 = 0 if comp5_sum == 0 else 1 if comp5_sum <= 9 else 2 if comp5_sum <= 18 else 3

    # Component 6 - medicatie
    comp6 = freq_to_score(answer_dict.get("7", ""))

    # Component 7 - functioneren overdag
    comp7a = freq_to_score(answer_dict.get("8", ""))
    comp7b = problem_to_score(answer_dict.get("9", ""))
    comp7 = round((comp7a + comp7b) / 2)

    totaal = comp1 + comp2 + comp3 + comp4 + comp5 + comp6 + comp7

    return {
        "Component 1": comp1,
        "Component 2": comp2,
        "Component 3": comp3,
        "Component 4": comp4,
        "Component 5": comp5,
        "Component 6": comp6,
        "Component 7": comp7,
        "Totaalscore": totaal
    }

# Questionnaire ophalen
def get_questionnaire(subject, questionnaire="PSQI", container="participants"):
    container_client = blobServiceClient.get_container_client(container)
    blobs = list(container_client.list_blob_names())
    relevant = [b for b in blobs if subject in b and questionnaire in b]
    if not relevant:
        return []
    blob = blobServiceClient.get_blob_client(container=container, blob=relevant[0])
    with BytesIO() as inputBlob:
        blob.download_blob().readinto(inputBlob)
        inputBlob.seek(0)
        lines = inputBlob.readlines()
        return [json.loads(line.decode("utf-8")) for line in lines]

# Hoofdscript
if __name__ == "__main__":
    rows = []
    for subject in deelnemers:
        try:
            entries = get_questionnaire(subject)
            for entry in entries:
                scores = calculate_psqi_score(entry["answers"])
                timestamp = entry.get("timestamps", {}).get("submitted", 0)
                datum = datetime.fromtimestamp(timestamp).strftime("%Y-%m-%d %H:%M:%S")
                scores["Subject"] = subject
                scores["Datum"] = datum
                rows.append(scores)
        except Exception as e:
            print(f"Fout bij {subject}: {e}")
```

```

    df = pd.DataFrame(rows)
    kolomvolgorde = ["Subject", "Datum"] + [f"Component {i}" for i in range(1, 8)] +
        ↪  ["Totaalscore"]
    df = df[kolomvolgorde]
    df.to_excel("psqi_scores.xlsx", index=False)
    print(" PSQI-scores opgeslagen in psqi_scores.xlsx")
```