Repairing R code to work with non-integer frequencies and calculating burstiness of posture changes.

Submitted by Archie Howard Norman to the University of Exeter

as a dissertation for the MSc in Applied Data Science and Statistics

in August 2024

Abstract

This dissertation aims to address limitations in the sed-up.R code used for processing accelerometer data from GENEActiv devices, focusing on classifying postures and introducing new analytical features. The initial code faced challenges with handling non-integer frequencies and lacked features such as burstiness scoring and sleep detection. To overcome these limitations, the code was examined and enhanced to handle fractional sampling frequencies and introduce a burstiness measure. Burstiness, which characterises the variability in movement patterns, offers deeper insights into periods of intense activity and inactivity. Additionally, a sleep detection feature was implemented to improve the classification of nighttime postures. The project integrated these new features while maintaining the accuracy of posture classification, as validated by comparison with data from the activPAL device which saw a 6% increase in accuracy.

Keywords: accelerometer data, posture classification, GENEActiv, burstiness.

Table of Contents

1 Introduction

Accelerometer data has become an essential tool in health and activity monitoring due to devices like the Fitbit, Apple Watch, and WHOOP. These devices, equipped with accelerometers, measure motion across three axes: vertical, horizontal, and perpendicular to capture detailed data on an individual's movements. This data is transformed into activity counts, enabling the analysis of physical activity, sedentary behaviour, and sleep patterns.

Understanding activity patterns provides insights into the links between behaviour and health outcomes. For example, prolonged sedentary behaviour is associated with a higher risk of chronic diseases such as cardiovascular disease and diabetes, while regular physical activity offers numerous health benefits, including improved cardiovascular fitness and mental health (Owen et al., 2010; Biswas et al., 2015). Accurately categorising these activities informs public health policies and personal interventions aimed at reducing sedentary time and increasing physical activity. Advancements in accelerometer technology have enhanced the accuracy and reliability of activity categorisation. Algorithms achieve accuracy rates of up to 90% for differentiating activities such as sitting and standing (Zablocki et al., 2024).

The Department of Public Health and Sport Sciences at the University of Exeter faces challenges with implementing its accelerometer data categorisation process. With the categorised data, the department aims to explore the relationship between the burstiness of posture changes and health. The existing R code `sedup`, which processes accelerometer data from GENEActiv devices doesn't work. The code includes functions such as data calibration, extraction of specific intervals, and the identification of non-wear periods. The main aim is to fix this code however, the department has requested additional features, including Burstiness scoring to provide deeper insights into the variability and intensity of physical activity, and sleep detection for more comprehensive assessments of activity patterns. Burstiness refers to the irregular occurrence of events in bursts rather than evenly distributed over time, providing valuable insights into physical activities.

The current R code faces challenges that limit its effectiveness and applicability. First, the code lacks features such as burstiness scoring and sleep detection. Burstiness scoring measures the irregular occurrence of events in bursts, revealing information about the intensity and variability of physical activity. This is particularly relevant, as traditional analyses often overlook the significance of activity bursts, potentially underestimating their impact on metabolic and cardiovascular health (Barabási, 2005).

The main issue is that the existing code is not operational, further compounding the other issues. Specifically, the current code's limitations in handling non-integer frequencies pose technical challenges in accurately categorising activities. Addressing these limitations is essential for enhancing the code's robustness and ensuring that it provides reliable, in-depth analysis of accelerometer data.

Objectives

This project aims to achieve the following objectives:

1. **Enhance Code Robustness**: Improve the existing R code to increase its reliability and prevent future errors. This includes optimising code efficiency, improving error handling, and ensuring compatibility with diverse datasets. The goal is to create a robust codebase that can handle real-world data variability without crashing or producing incorrect outputs.

2. **Develop Burstiness Scoring**: Integrate a burstiness scoring algorithm to provide insights into the variability and intensity of physical activity. Burstiness, which refers to the irregular occurrence of events in bursts rather than being evenly distributed over time, offers valuable insights into activity patterns (Barabási, 2005).

3. **Implement Sleep Detection**: Introduce a method for detecting sleep periods from accelerometer data. This is especially key due to the code's weakness in correctly categorising periods of sleep as laying/sitting bouts.

By addressing these objectives, the project seeks to improve the reliability and depth of accelerometer data analysis. This will contribute to better health monitoring and research, enabling more precise identification of activity patterns and their health implications. The success of these objectives will be evaluated through analysis and code benchmarks to ensure that the improvements meet the desired accuracy and robustness standards.

Methodology Overview

First, the existing R code responsible for sitting and standing categorisation will be thoroughly analysed and troubleshooted. This process involves breaking down the code into individual functions to identify specific points of failure or inaccuracies. Each function will be tested and debugged to ensure proper operation and alignment with the objectives.

Next, a burstiness scoring algorithm will be integrated into the code while maintaining the original structure as closely as possible. This will ensure that the burstiness measure is incorporated without disrupting the existing functionality, preserving the code's integrity and flow.

Finally, sleep detection will be implemented to identify sleep periods. This will overcome the limitations of the current categorisation algorithm, which struggles with detecting sleep patterns reliably.

Given the absence of a dataset with labelled posture changes, ensuring that the modified code maintains the same level of accuracy as the original research presents a challenge. The accuracy of the updated algorithm will be validated by comparing its output against data from their ActivePAl device, which serves as the benchmark for performance.

This dissertation aims to advance the analysis of posture data by incorporating fractional frequencies and burstiness measures, providing a more accurate and comprehensive understanding of the data. The following chapters will explore these ideas in greater detail, starting with a review of the existing literature on posture detection and related methodologies.

2 Literature Review

2.1 Literature Review on Categorising Posture in Accelerometer Data

With the advent of wearable technology, particularly accelerometers, researchers can collect and analyse extensive datasets to classify different postures and activities. This review aims to summarise the current methodologies, challenges, and advancements in the field of posture classification using accelerometer data, focusing on both wrist and thigh-worn devices.

Theoretical Background

Accelerometers measure acceleration forces in three axes (x, y, and z) to detect movement and orientation changes. The raw accelerometer data is processed using algorithms and models to classify postures accurately. These sensors are commonly used in wearable devices to monitor various physical activities and postures, such as sitting, standing, lying down, and walking.

Methodologies for Posture Classification

Machine learning approaches have been widely adopted for posture classification. Traditional methods like logistic regression models and decision trees have been effective. For instance, Trost et al(2005). implemented logistic regression models to classify physical activities using data from wrist-worn and hip-worn accelerometers, with a classification accuracy of up to 85% for distinguishing between walking, running, and sedentary behaviours.

Similarly, decision trees, noted for their simplicity and interpretability, have been successfully utilised. Mannini et al (2013) used a support vector machine to classify activities in youth with a single wrist-worn accelerometer, achieving an accuracy of 80% for activities like sitting, standing, and walking. However, while these traditional models are straightforward and interpretable, their performance may be limited in more complex activity scenarios.

Advanced machine learning techniques have further improved classification accuracy. Shoaib et al, (2021) employed neural networks along with other classifiers to distinguish between various activities such as sitting, walking, and running, reporting high classification accuracies of over 84.89%. While neural networks provide powerful tools for pattern recognition, they also introduce complexity and a lack of interpretability.

Deep learning techniques, particularly convolutional neural networks (CNNs) and recurrent neural networks (RNNs), have demonstrated superior performance in activity classification. Sani et al (2017) explored deep learning models, including CNNs and hybrid approaches like CNN-SVM, using wrist and thigh accelerometer data, achieving high F1-scores of 0.92. The efficacy of deep recurrent models in handling time-series data further underscores their value in enhancing classification accuracy. However, these models often act as "black boxes," posing challenges for interpretability and practical application.

Innovative hybrid models have also emerged. Rowlands et al (2016) introduced the Sedentary Sphere (SS) method, which uses arm inclination and motion intensity for posture classification, showing moderate accuracy of around 70% in free-living environments. Straczkiewicz et al (2020). developed SedUp, combining wrist inclination and motion variability features with a logistic regression classifier, resulting in accuracy of over 85%.

Challenges in Posture Classification

Despite advancements, several challenges persist in posture classification using accelerometer data. Data variability, due to the complexity of human motion and diversity of activities, poses a significant challenge. Wrist-worn devices capture a wide range of hand movements, complicating the distinction between similar activities such as writing and eating or using a phone. This variability necessitates sophisticated algorithms capable of distinguishing subtle differences in movement patterns.

Wrist-worn accelerometers, while convenient and exhibiting high compliance, may not be as accurate as hip-worn devices for certain activities due to arm movement variability. For example, Karas et al (2018). found that hip-worn accelerometers achieved higher accuracy compared to wrist-worn devices in classifying sitting and standing postures. The placement of the accelerometer significantly influences the data collected and, consequently, the classification accuracy. Researchers must carefully consider sensor placement to balance convenience and data fidelity.

Algorithm complexity, particularly with advanced machine learning models, presents another challenge. Many of these models function as "black boxes," making it difficult to interpret results and understand the underlying decision-making process. This lack of transparency can hinder the acceptance and practical implementation of these models.

Advancements and Applications

Recent advancements have prioritised validating posture classification models in free-living conditions, moving beyond controlled laboratory settings. This shift is crucial for developing algorithms that can be reliably applied in real-world scenarios. Integrating data from multiple sensors, such as wrist and thigh accelerometers, improves accuracy and robustness, offering a more comprehensive view of physical activity and sedentary behaviour. However, this multi-sensor integration also increases the complexity of data processing and analysis.

Evaluation of the SedUp Algorithm for Posture Classification

It is to be noted that the sedup algorithm was preselected for the code.

The SedUp algorithm, developed by Straczkiewicz et al. (2020), presents an approach for classifying sedentary and upright postures using wrist-worn accelerometers. However, while SedUp has several strengths, it also has notable limitations that warrant further consideration.

This section evaluates both the contributions and shortcomings of the SedUp approach, reflecting on its broader implications.

Strengths of the SedUp Algorithm

A primary strength of the SedUp function lies in its simplicity. By using only two key features median acceleration and the standard deviation of acceleration the function effectively classifies body posture with minimal computational complexity. This straightforward approach enhances its applicability, especially in long-term studies where computational efficiency and ease of implementation are essential. SedUp's reliance on wrist-worn accelerometers, which are more user-friendly than devices placed on the hip or thigh, further contributes to its effectiveness, as it promotes higher user compliance in prolonged monitoring scenarios. This simplicity makes SedUp a practical tool for both research and broader commercial applications, such as in wearable fitness trackers and health monitoring devices.

Another important strength is SedUp’s simplicity and interpretability. Unlike many advanced machine learning models that function as "black boxes," SedUp’s reliance on two simple features median acceleration and the median standard deviation of acceleration making it easier to understand and interpret. This transparency is good in health research, where the ability to explain the model's decision-making process is important for gaining trust from practitioners, patients, and stakeholders. By offering a more interpretable model, SedUp circumvents a common problem in advanced analytics: the disconnect between the accuracy of the model and its practical usability.

Moreover, SedUp delivers high classification accuracy. With a true positive rate (TPR) of 0.83 and a true negative rate (TNR) of 0.91 from left wrist data, the method demonstrates a clear improvement over alternative methods, such as the Sedentary Sphere (SS). This strong performance suggests that SedUp effectively captures the nuances of postural changes, making it a valuable in studies where accurate detection of posture transitions is needed. Additionally, the open-source nature of the algorithm promotes transparency and accessibility, allowing other researchers to validate or extend its application in various domains. This openness enhances the credibility of the model, ensuring that the results can be scrutinised and built upon by others in the field.

Furthermore, SedUp’s validation in free-living environments is a strength. Many algorithms developed for posture classification are tested in controlled laboratory settings, where the predictability of movements limits real-world applicability. In contrast, SedUp was validated using data from community-dwelling older adults, which adds credibility to its potential application in everyday settings. This real-world testing enhances the ecological validity of the method, suggesting that it may perform well outside the controlled environments typical of previous studies. It highlights SedUp’s potential to move beyond academic applications and into practical, large-scale epidemiological research.

Weaknesses of the SedUp Algorithm

Despite its strengths, the SedUp algorithm faces several limitations. One significant weakness is its limited generalisability. The model was tested exclusively on older adults, who tend to have more predictable, slow-moving activity patterns compared to younger, more active individuals (Straczkiewicz et al., 2020). Younger individuals or those engaged in more dynamic activities may exhibit more varied movement patterns that could challenge SedUp's classification accuracy. This limitation is compounded by the fact that SedUp relies on static thresholds for classifying postures. These predefined thresholds do not adapt to individual variations in movement patterns, making it difficult for the algorithm to accurately classify postures in populations with more dynamic or irregular activities. Static thresholds are effective for simpler, more predictable movements, but they may not capture the full complexity of posture transitions in younger, athletic, or mobility-impaired populations. Without further validation in these groups and a more adaptive thresholding mechanism, the generalisability of the algorithm remains in question, limiting its broader applicability.

The large time window (up to 90 seconds) used in SedUp also introduces a issue: the omission of short-duration posture transitions, such as sit-to-stand or stand-to-sit movements. While this improves the model’s accuracy for sustained postures, it undermines its sensitivity to short-term events, which could be particularly important in health contexts, such as monitoring mobility or fall risk in older adults. In rehabilitation, for instance, the ability to detect quick posture changes could provide valuable insights into a patient’s progress. SedUp's inability to capture these transitions limits its utility in such scenarios, suggesting that future iterations of the model may need to integrate more sensitive metrics to address this gap.

Additionally, SedUp’s reliance on a single axis for measurement (the y-axis) is a notable limitation. While this simplifies the model, it may reduce its accuracy in capturing more complex postures and movements, particularly during activities involving lateral or rotational motions. Movements such as twisting, turning, or lateral bending are common in daily activities but may be poorly captured by focusing on a single axis. This limitation raises issues regarding the robustness of the model when applied to more dynamic environments or when classifying activities that involve a broader range of motions, such as sports or physical therapy settings.

Another point of reflection is the trade-off between simplicity and performance. While SedUp’s logistic regression model offers interpretability, it is sensitive to tuning parameters, particularly the window size (τ). The authors note that optimising this parameter is essential to balancing accuracy with responsiveness, but the choice of a large window might smooth out important short-term events, while smaller windows might introduce. This inherent trade-off highlights a broader challenge in posture classification: the need to balance model simplicity and interpretability with the need for accurate detection of postural transitions. As wearable technology and data collection methods evolve, future research should focus on developing adaptive models that can dynamically adjust to different activity levels and environments, without compromising on interpretability.

Finally, SedUp’s validation process lacks direct observation as the ActivePal will make some mistakes. The study relies on activPAL data for validation, which, while widely used, does not provide the same level of accuracy as direct observation. The absence of direct observational data raises concerns about potential biases in the validation process, as errors inherent in the activPAL device could be mirrored in SedUp's reported accuracy. For a more robust validation, future studies should include direct observational measures as the ground truth to ensure the highest level of accuracy. This limitation points to a broader issue in the field: the reliance on sensor-to-sensor validation, which may mask underlying inaccuracies shared across devices.

Conclusion

The SedUp algorithm represents a meaningful advancement in posture classification using wrist-worn accelerometers, particularly in free-living environments. Its strengths lie in its simplicity, high accuracy, and ease of implementation, making it a valuable tool for health research and large-scale epidemiological studies. The algorithm's ability to maintain user compliance through its use of wrist-worn devices, along with its interpretability and transparency, further enhance its practical application.

However, SedUp also has notable limitations. Its reliance on data from older adults, the omission of short-duration posture transitions, and its focus on a single axis reduce its generalisability and sensitivity to more dynamic movements. To broaden its applicability, future research should validate SedUp across more diverse populations, optimise its ability to detect short-term postural changes, and consider multi-axis or multi-sensor integration to capture more complex movements.

2.2 Literature Review on Measuring Burstiness

Burstiness, characterised by periods of high activity followed by long intervals of inactivity, is a common phenomenon observed in various natural and social systems. This pattern can be found in diverse domains such as email communication, web browsing, earthquake occurrences, and more. To analyse the underlying mechanisms of these systems and for predicting and controlling their behaviours the burstiness measures must be understood. This literature review aims to provide a comprehensive overview of the measures used to quantify burstiness, particularly focusing on finite event sequences, and to discuss the limitations and advancements in this field. Additionally, the review will explore the application of burstiness measures to accelerometer data for analysing posture changes.

Theoretical Background

Key terms in this context include interevent time (the time interval between consecutive events) and the coefficient of variation (a measure of the relative variability of interevent times).

Historically, the study of burstiness has evolved from simple observations of event patterns to sophisticated mathematical modelling. Early work focused on identifying and describing bursty patterns in specific datasets, while more recent research has developed quantitative measures to compare and analyse burstiness across different systems.

Traditional Measures of Burstiness

One of the most widely used measures of burstiness is the burstiness parameter (B), introduced by Goh and Barabási, (2008). The burstiness parameter is defined as:

$$B = \frac{\sigma - \mu}{\sigma + \mu} = \frac{r-1}{r+1}$$

where sigma is the standard deviation and mu is the mean of the interevent times, and r equals sigma divided by mu. This parameter effectively captures the degree of burstiness, with B=−1 indicating regular, periodic sequences, B=0 for Poisson (random) sequences, and B≈1 for highly bursty sequences.

Finite-Size Effects on Burstiness Measures

One major challenge with the burstiness parameter is its sensitivity to the number of events in a sequence. When the number of events *n* is small, the maximum value of *B* is constrained, potentially leading to underestimation of burstiness. This issue is particularly pronounced in empirical datasets where the number of events varies significantly across different sequences (Kim & Jo, 2016). In such cases, finite-size effects distort the true burstiness of a system, especially in datasets that involve rare or infrequent events.

Alternative Measures to Address Finite-Size Effects

It is to be noted that this burstiness measure was preselected for the code

To mitigate the finite-size effects, Kim and Jo proposed a novel burstiness measure, An(r), which adjusts for the sample size n. This measure is defined as:

$$A_n(r) = \frac{\sqrt{n+1}r - \sqrt{n-1}}{\left(\sqrt{n+1}-2\right)r + \sqrt{n-1}}$$

$r = \frac{\sigma}{\mu}$ is the coefficient of variation, with σ being the standard deviation of the interevent times and μ being the mean and *n* is the number of events. This formulation ensures that An(r) does not have an upper bound due to finite n, providing a more robust measure of burstiness across sequences with varying event counts. The robustness of An(r) against finite-size effects makes it a more reliable tool for analysing burstiness in datasets with varying numbers of events. For example, in social media data, where some users are highly active and others post infrequently, An(r) allows for a more accurate comparison of burstiness levels.

Kim and Jo tested their burstiness measure An(r) on a large-scale Twitter dataset, where users' tweeting behaviour varied significantly in frequency. They found that the traditional burstiness parameter b exhibited an upper bound for users with fewer tweets, while An(r) provided more accurate representations of bursty dynamics. This improved accuracy highlights the utility of An(r) in capturing the intrinsic bursty behaviour across users with different activity levels.

In addition to Twitter, burstiness measures have been applied to various domains such as financial market data, where irregular trading patterns can indicate market volatility, and biological systems, where bursty behaviour is observed in neuron firing patterns.

Incorporating Additional Temporal Features

The minimum interevent time captures the intrinsic timescale of the system, such as the refractory period in neurons or the minimum time required for a meaningful posture change. Adjusting for τmin, another measure An,y~(r) was proposed defined as:

$$A_{n,\tilde{y}}(r) = \frac{(n-2)\left[\sqrt{n+1}r - \sqrt{n-1}(1-n\tilde{y})\right]}{\left[n\sqrt{n+1} - 2(n-1)\right] r + \sqrt{n-1}(n-2\sqrt{n+1})(1-n\tilde{y})}$$

$\tilde{y} = \frac{\tau_{\min}}{T}$ is the ratio of the minimum interevent time τmin to the total observation period *T*. This accounts for the minimum interevent time, providing an even more refined assessment of burstiness in event sequences (Kim & Jo, 2016).

Additional Measures and Concepts

Besides the burstiness parameter (B) and its refinements, the memory coefficient is another important measure for understanding bursty sequences. This coefficient quantifies the correlation between successive interevent times, providing insight into the temporal memory of the system. Karsai et al. (2018) discuss the interplay between burstiness and memory, emphasising their combined effect on the dynamics of human activities.

Higher-order temporal correlations capture more complex dependencies and patterns in event sequences, which help to accurately model systems where interactions involve multiple events and entities over time (Karsai et al., 2018). Applying these advanced techniques in fields like social network analysis and health monitoring could lead to a better understanding of how bursty behaviour spreads through networks or influences health outcomes.

Measuring Burstiness in Accelerometer Data for Posture Changes

The concept of burstiness can be effectively applied to the analysis of accelerometer data to measure posture changes. Burstiness in this context would reflect periods of frequent posture adjustments followed by longer periods of stability, which is particularly relevant for understanding patterns of physical activity and rest.

Several studies have utilised accelerometer data to analyse human movement and posture. However, the specific application of burstiness measures to this data is relatively novel. By applying burstiness measures researchers can gain insights into the temporal dynamics of posture changes. This can have applications in healthcare, ergonomics, and sports science, where understanding the patterns of movement and rest can inform interventions and improvements in human health and performance (Holme, 2018).

Methodological Considerations

Accurate time-stamping and careful filtering of event data are essential to ensure the reliability of burstiness measures emphasising the need for accurate timestamping and data filtering. Consider providing best practices or common pitfalls in data preparation for burstiness analysis. Statistical techniques, such as calculating the coefficient of variation and applying burstiness parameters, must be applied consistently to yield meaningful comparisons across different datasets.

Challenges and Future Directions

Despite the advancements in burstiness measurement, several challenges remain. Current models may not fully capture the complexity of bursty dynamics, and there is a need for more sophisticated tools to analyse higher-order temporal correlations and contextual factors. Future research could focus on developing comprehensive models that can incorporate additional temporal features and account for varying conditions across different systems.

Measuring burstiness in finite event sequence-s presents unique challenges due to limited data, which can lead to biased or inaccurate results using traditional measures. The introduction of alternative measures, such as An(r) and An,y~(r), offers robust solutions that mitigate finite-size effects and provide more reliable quantifications of burstiness. These advancements are increasing accuracy for characterising the temporal dynamics of various natural and social phenomena, including posture changes measured by accelerometers.

Ultimately, these tools aid in better understanding, prediction, and control of complex systems.

3 Methodologies

3.1 Data

The accelerometer data utilised in this project was produced using the GENEActiv 1.1 device, a wrist-worn accelerometer widely used in academic research settings. This device is particularly suitable for long-term monitoring of physical activity and provides detailed, raw sensor data that can be post-processed to extract meaningful metrics.

Device Specifications and Data Output

The GENEActiv 1.1 device records movement data across three axes (X, Y, Z), light intensity, and button status. The device is capable of measuring data at frequencies ranging from 20 Hz to 100 Hz, which influences both the duration of data collection and the resolution of the recorded activity. Specifically, a lower frequency of 20 Hz allows for continuous data collection over a 30-day period, while a higher frequency of 100 Hz limits the data collection period to approximately 7 days.

Data Format and Structure

Data from the GENEActiv device is stored in an efficient binary format, making it more suitable for managing large datasets over long periods compared to CSV files. Each binary file contains the following structure:

- **Main Header** (lines 1-59): Includes metadata such as device ID, participant information, and configuration settings.
- **Page Header** (lines 61-68): Stores segment-specific metadata like timestamps, battery voltage, and ambient temperature.
- **Data Block** (from line 69 onwards): Holds the raw sensor readings, including:
    - **Accelerometer Data**: Unprocessed X, Y, Z axis readings, later calibrated to convert into gravitational units (g).
    - **Light Meter Data**: Raw light intensity values, which are converted into lux during post-processing.

- **Button Status**: Logs the state of the device's button, which can be used to mark specific events during data collection.

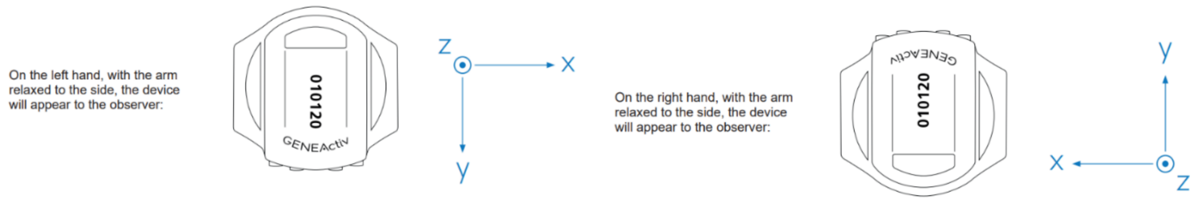

On the left hand, with the arm relaxed to the side, the device will appear to the observer:
010120
GENEActiv
Z
X
y
On the right hand, with the arm relaxed to the side, the device will appear to the observer:
010120
GENEActiv
y
X
Z

*Figure 1~ GENEActiv Instructions for Use. Activinsights. (2022)**.** Diagram showing how the Y axis is different depending on which wrist the device is worn*

The main issue with the data collection is the instructions for use document as the sedup function requires an input assuring that x ~= 1 when device is facing the ground but as seen in figure one this will only happen when device is on the left hand. In the main header of the binary file there is information of which hand the device was on but it has been removed or not filled in due to ethical considerations as all particapnt information has been removed.

Data Calibration and Post-Processing

The raw data captured by the GENEActiv device requires post-processing to convert the binary readings into usable metrics. This involves applying calibration factors such as offsets and gains to the raw accelerometer data to translate it into physical units like gravity (g) for acceleration. Similarly, light meter data is converted from raw readings into lux, a standard unit of light intensity.

The current implementation assumes uniform data quality and consistency, an assumption that often leads to inaccuracies when processing datasets with noise or missing values. The code does not incorporate preprocessing steps to clean or validate the data, which if included could improve the reliability and accuracy of the results. Without adequate data validation, the system may produce erroneous classifications and metrics, reducing its overall effectiveness.

Data Source and Testing

The dataset provided by the Department comprises of accelerometer data recordings of varying lengths and frequencies, which are essential for testing and validating the improved R code under multiple scenarios. The diversity in the dataset allows for comprehensive testing of the categorisation, burstiness scoring, and sleep detection, ensuring that the code will perform well across different conditions.

All data collection was conducted in accordance with specific protocols, ensuring consistency and reliability in the recorded measurements. The protocols included standardised procedures for device setup, participant instructions, and data handling, which contribute to the overall quality and integrity of the dataset.

3.2 Overview of Existing R Code

The existing R script is a comprehensive tool designed to process and analyse accelerometer data obtained from GENEActiv devices, which are widely used in research settings for monitoring physical activity. The core functionality of the script is focused on converting raw accelerometer data into meaningful metrics that can be used to classify physical activities, specifically distinguishing between sitting and standing postures, and to calculate various other relevant activity metrics.

The primary components of the script include the following functions:

- **Calibration Function (`calibration`)**: This function is responsible for adjusting the raw accelerometer data to correct for device-specific biases. It utilises the output from the GENEActiv calibration process to apply scale and offset corrections to the data, ensuring that the subsequent analysis is based on accurately calibrated readings.
- **Interval Extraction Function (`get_interval`)**: This function extracts accelerometer data for specified time intervals, processes the data, and computes the amount of time the device was not worn during each interval. This excludes non-wear periods from the analysis, thereby enhancing the reliability of the activity classification.
- **Posture Classification Function (`SedUp`)**: The core of the script, this function classifies accelerometer data into sitting and standing postures. It implements a logistic regression model that uses the standard deviation and median of the accelerometer signal to estimate the probability of the user being in a standing position. This classification is performed across specified time windows, allowing for detailed temporal analysis of posture.
- **ENMO Calculation Function (`add_ENMO`)**: The ENMO (Euclidean Norm Minus One) metric is calculated by this function, which quantifies movement intensity. ENMO is a widely recognised metric in physical activity research, used to assess overall activity levels.

- **Bout Computation Function (`compute_bouts`)**: This function identifies continuous periods of similar activity, known as bouts, based on the classification output from the SedUp function. It further categorises these bouts as active or sedentary, providing a detailed breakdown of the user's activity patterns.

The initial R code was developed to categorise accelerometer data into sitting and standing activities. However, several limitations have been identified that impact its accuracy and efficiency.

One of the primary issues with the current code is the use of static thresholds to differentiate between sitting and standing activities based on acceleration magnitude. This approach does not accommodate individual variations in movement patterns, which can result in inaccuracies. Differences such as body weight, height, and personal movement styles affect accelerometer readings (Bouten et al., 1994). Consequently, a universal threshold is ineffective for accurately classifying activities across diverse individuals.

Additionally, the code demonstrates inefficiencies in data processing, leading to longer execution times when handling large datasets. The existing data processing pipeline is not optimised for performance, making it unsuitable for applications requiring rapid processing of large volumes of data due to the code repeatedly reading and loading data.

Modularisation and Code Refactoring

The existing codebase demonstrates a high level of modularity, with well-defined functions that handle specific tasks such as data calibration, interval extraction, non-wear time detection, and activity classification. This modular structure enhances maintainability and facilitates the addition of new features and the extension of existing functionalities.

Benefits of the Current Modular Structure:

1. **Maintainability**: The modular approach allows for easier maintenance of the codebase. Each function is self-contained, making it straightforward to update or debug specific components without affecting the entire system.
2. **Reusability**: The code's modular structure promotes reusability, as functions like calibration and get_interval can be easily reused across different parts of the analysis or even in other projects. This reduces redundancy and accelerates the development process for new features.

3. **Scalability**: The modular design supports scalability, allowing new analytical features such as burstiness scoring, or sleep detection to be integrated without requiring significant modifications to the existing codebase.

Initial Limitations and Identified Issues

During the preliminary analysis of the existing R code, several limitations were identified that could impact the accuracy and efficiency of the activity classification process:

1. **Posture Classification Accuracy**: The SedUp function, which is pivotal for categorising postures, exhibited a tendency to misclassify, particularly in instances of sleep. This misclassification introduces errors into the analysis, potentially leading to misleading conclusions about sedentary behaviour.
2. **Performance Inefficiencies**: The script's performance was observed to degrade when processing large datasets, such as those involving long-term monitoring or data from multiple participants. This inefficiency is primarily due to the use of nested loops and calculations within functions like compute_bouts, which are computationally intensive. As a result, the processing time is considerably longer than optimal, limiting the scalability of the script for large-scale studies.
3. **Features**: The existing script didn't include analytical features that are increasingly relevant in the field of physical activity research. Notably, it lacks a burstiness score, which quantifies the variability and irregularity of movement patterns. Additionally, the script does not incorporate sleep detection.

Objectives for Enhancement

Considering these limitations, the primary objectives for the enhancement of the existing R script are as follows:

- **Maintenance of Posture Classification Accuracy**: Accuracy is a high priority so when making changes to the code its key to not alter the accuracy of the classification.
- **Integration of Features**: Extend the functionality of the script by incorporating a burstiness scoring algorithm to analyse the variability of movement patterns, and by adding a sleep detection algorithm to identify periods of sleep based on accelerometer data.

- **Enhancement of Code Modularity and Readability**: Maintain the script modularity, making it easy to modify and extend. This involved separating additions to the code into distinct functions or modules, improving documentation, and adopting best practices in coding to facilitate easier maintenance and further development.

To enhance system performance and address current limitations, several improvements are recommended. Replacing loop-based operations with vectorised functions would streamline data handling and reduce runtime, particularly for large datasets. Additionally, implementing parallel processing could improve performance by utilising multi-core processors for concurrent tasks.

Leveraging widely used R packages such as data.table for efficient data manipulation and caret for machine learning would improve both performance and readability, reducing reliance on custom code and enhancing maintainability.

Finally, maintaining clear documentation and inline comments is essential as the codebase expands. Comprehensive documentation will ensure future developers can easily understand the code's structure, improving accessibility and reducing the learning curve for future enhancements.

3.3 Identifying the issue and troubleshooting

During the project, it was noted that the code ceased functioning correctly following an update, and the issue was suspected to be related to the frequency setting of the device. To diagnose and resolve this, the code was systematically stripped out of its functions and run line by line to isolate the specific point of failure.

Initial Issue: Output Directory

The first problem encountered was with the output directory. The code required that the output directory already exist, but it did not automatically create one if missing. This was resolved by either manually creating the directory or by modifying the code to set recursive to true in the directory creation function, allowing the automatic creation of nested directories if needed.

Main Issue: Handling of Non-Integer Frequencies

Once the directory issue was fixed, the code ran successfully with integer frequencies. However, when non-integer (fractional) frequencies were used, the code triggered an “subscript out of bounds”. "Subscript out of bounds," error typically indicates that the program attempted to access data using an index or range of indices that exceed the actual dimensions of the dataset. The error arises when the program tries to access a specific row or a range of rows that do not exist within the dataset, such as attempting to retrieve data from a position beyond the number of available rows.

This issue is commonly caused by:

1. **Index Out of Range**: Miscalculation of indices, leading to attempts to access data from positions beyond the actual number of rows.
2. **Empty or Incomplete Data**: Scenarios where the dataset contains fewer rows than expected due to issues in data collection or preprocessing.
3. **Logical Errors in Calculation**: Incorrect assumptions about the data's size or errors in the logic used to generate indices.

Troubleshooting Process

To address this issue, the following steps were taken:

1. **Verification of Dataset Dimensions**: The actual size of the dataset was verified by checking the number of rows to ensure that the dataset was correctly populated and contained the expected number of entries.
2. **Validation of Index Calculation**: The logic used to calculate indices was reviewed to ensure they were within the valid range. This included tracing the sequence of operations that generated these indices, particularly in cases where conditions might result in indices exceeding the dataset’s bounds.
3. **Handling Edge Cases**: Checks were implemented to ensure that any attempt to access data was only made when the indices were confirmed to be within the valid range, thereby avoiding the error.
4. **Error Mitigation through Code Adjustment**: Upon further investigation, it was identified that the method used to calculate the standard deviation (sdv) of specific segments of the data was contributing to the issue. Originally, the approach involved applying a function to a matrix that may have inadvertently created indices outside the data’s bounds.

To prevent future issues, the code has been packaged, offering several key advantages. Remote updates are now possible, allowing for seamless improvements without manual intervention. By specifying library versions, the risk of future updates breaking the code is minimized, ensuring stability. Packaging also simplifies distribution, making it easier to share across teams or platforms. Additionally, it promotes better code organization, improves maintainability, and ensures consistency across different environments, enhancing overall software reliability and scalability. The package code is available on:
https://github.com/Summer-Project-SEDUP/MYDISS

See the readme file for instructions on how to install and run the package

4 Implementations

4.1 Handling Non-Integer Sampling Frequencies

In the context of the SedUp function, the sampling frequency (fs) shapes the input data vector y into a matrix for further analysis. The challenge arises when the sampling frequency is not an integer, such as 85.7 Hz, which can complicate the matrix formation and subsequent calculations. Three main strategies can be employed to address this issue: rounding the frequency, interpolating to a new frequency, or changing the matrix formation approach to handle non-integer frequencies. Each of these strategies has implications for the accuracy and computational efficiency of the analysis.

Rounding the Frequency

A simple approach is to round the non-integer frequency to the nearest integer. For example, 85.7 Hz can be rounded to 86 Hz, simplifying matrix formation by ensuring each column corresponds to a fixed time interval.

- **Matrix Formation**: Rounding to 86 Hz results in 86 rows per column, slightly distorting the time representation. Each column now represents an interval marginally longer than one second.
- **Standard Deviation Calculation**: The apply function calculates the standard deviation of each column. With rounding, the variability is captured over extended intervals, potentially causing minor inaccuracies.

- **Practical Considerations**: For most datasets, the introduced error is minimal. However, in applications needing high accuracy, accumulated deviations could lead to noticeable errors over time.

Interpolation to Adjust Frequency

Another method is to use interpolation to adjust the sampling frequency from a non-integer value, such as 85.7 Hz, to an integer frequency, like 86 Hz. Interpolation involves estimating new data points within the existing data to achieve the desired frequency.

Implications of Interpolation:

- **Data Smoothing**: Interpolation effectively smooths the data, filling in gaps to align the dataset with the new frequency. Linear interpolation connects points with straight lines, while more complex methods, like spline interpolation, use polynomial curves for a smoother transition.
- **Compatibility**: By adjusting the data to a standardised frequency, interpolation ensures compatibility with algorithms and systems that require integer frequencies, facilitating consistent analysis across different datasets.
- **Computational Cost and Accuracy**: Interpolation can be computationally intensive, particularly for large datasets. Additionally, if the data is highly dynamic or contains sharp changes, interpolation may introduce artifacts or distortions, affecting the accuracy of subsequent analyses.

Removing the use of Matrix

An alternative approach involves modifying the use of the matrix. Instead of rounding or interpolating the data to accommodate a matrix it can be replaced with a list

Implications of Adjusting Matrix Formation:

- **Flexible Data Handling**: This method allows the analysis to proceed without introducing potential interpolation errors. The matrix would be replaced with lists in a way that respects the non-integer frequency.
- **Complexity**: Implementing this approach requires a more computation to manage the segmentation, which complicates the code and increase the computational burden.

- **Precision**: This method maintains the original data's integrity, potentially offering the highest precision in cases where rounding or interpolation would introduce unacceptable errors.

Handling non-integer sampling frequencies in the SedUp function requires choosing between rounding, interpolation, or adapting the matrix formation process. Each method involves trade-offs:

- Rounding is the simplest and often sufficient, but it introduces minor inaccuracies.
- Interpolation provides better alignment but can be computationally expensive and may introduce artifacts.
- Removing use of matrix adds complexity and inefficiency but maintains accuracy

Accuracy is the highest priority, and the efficiency isn't severely harmed so the replacement of a matrix to a list is the best option out of the three.

4.3 Sleep status Implementation

In this project, a sleep status was implemented to identify periods during which the wearer of the accelerometer is likely to be asleep. The current method focuses on identifying sleep based on time-of-day, specifically targeting the hours between midnight and 6 AM. This time-based approach is a common heuristic used to estimate sleep periods, as these hours generally correspond to nighttime sleep for most individuals.

The sleep detection mechanism operates on the timestamps associated with the accelerometer data. The process can be described as follows:

1. **Time Extraction**: The code first converts the timestamps into hour-of-day format. This conversion allows the method to assess which part of the day each data point corresponds to.
2. **Identification of Potential Sleep Periods**: The method then scans through these hour-of-day values to identify any data points that fall within the 00:00 to 06:00 time window.
3. **Determination of Sleep Intervals**: If data points are found within this designated sleep window, the algorithm marks the bout to laying

Assumptions and Limitations

The current sleep detection method assumes that sleep occurs predominantly during the hours of midnight to 6 AM. While this assumption holds true for many individuals, it may not accurately reflect the sleep patterns of those with irregular schedules, such as shift workers. Additionally, the current implementation does not consider actual activity data, such as periods of inactivity, which could provide a more accurate detection of sleep.

4.4 Burstiness Implementation

The burstiness calculation in this project is implemented through a function that systematically processes the accelerometer data to measure the variability in active periods. The implementation involves several key steps:

1. **Data Preparation**: The burstiness calculation begins by processing the accelerometer data to focus exclusively on periods of physical activity. The data frame used in this process includes columns for the date, time, and a binary indicator of activity (with 1 representing standing periods and 0 representing laying/sitting).

2. **Filtering Standing Periods**: The first step in the function is to filter the dataset to include only rows where the posture indicator is set to 1. This ensures that the analysis is focused solely on periods of standing activity, excluding any periods of sitting that could skew the burstiness measurement.

3. **Combining Date and Time**: To accurately measure the temporal distribution of activity events, the date and time columns are combined into a single datetime column. This combined datetime format allows for precise calculation of the time intervals between consecutive activity events.

4. **Calculation of Time Differences**: Once the data is prepared, the function calculates the time difference (in seconds) between each consecutive pair of activity events. These time differences, referred to as interevent times, are needed for assessing the variability of activity over time.

5. **Computing Burstiness**: The burstiness score is computed using the An,y~(r) measure.

6. **Output**: The function returns burstiness score both for each file and for the entire dataset.

Integration into the Workflow

The burstiness calculation is integrated into the broader data processing workflow through a function that applies the burstiness calculation to each individual's data on a day-by-day basis. This integration involves the following steps:

1. **Iterative Application**: The burstiness calculation function is applied iteratively to the data of each individual, processing their daily activity data to generate burstiness metrics. This ensures that burstiness is measured consistently across all subjects and time periods without missing a day or individual.

2. **Result Compilation**: The burstiness metrics for each individual and each day are compiled into separate data frames, which are then saved as CSV files. These individual results are also aggregated into a master file (all_burstiness_metrics.csv), allowing for comparative analysis across different individuals and time periods.

Potential Enhancements

Enhance the burstiness analysis by incorporating additional factors, such as:

- **Incorporating Inactivity**: Analysing burstiness during periods of inactivity could provide additional insights into patterns of rest and recovery.

- **Custom Time Windows**: Allowing for the definition of custom time windows for burstiness analysis could enable more targeted studies, such as comparing daytime and nighttime activity patterns.

- **Memory function:** A memory parameter is useful for identifying whether bursts of activity are random or influenced by previous events, helping to distinguish between consistent patterns and irregular bursts.

4.5 Validation

The activPAL is a highly regarded, thigh-worn accelerometer designed to capture detailed information on posture and physical activity. It excels at distinguishing between sitting, standing, and walking postures by detecting changes in the orientation and movement of the thigh. Unlike wrist-worn devices, the activPAL is particularly well-suited for posture analysis due to its placement on the lower body, providing more direct and accurate measurements of postural transitions and time spent in various postures.

One of the key strengths of the activPAL is its validated accuracy in distinguishing postures, particularly in free-living environments where uncontrolled, real-world conditions can introduce variability. It has been extensively used in health research and clinical studies as a reliable criterion measure for assessing sedentary behaviour, posture transitions, and physical activity patterns. Its ability to track time spent sitting, standing, and stepping makes it a gold standard for posture classification in various populations, including older adults and individuals with mobility challenges.

In this project, activPAL data is used as the criterion measure to validate the posture classification produced by the enhanced R code. Since the activPAL has been validated in numerous studies and is widely accepted for its precision in detecting posture, it serves as a reliable benchmark for comparing the accuracy of my code's classifications. By aligning the outputs from the activPAL with those from the developed algorithm, it is possible to assess the performance of the new code in classifying sitting and standing postures. This comparison ensures that the newly developed algorithm provides an accurate representation of postural data and can be used with confidence in future research.

Using activPAL as the validation tool strengthens the credibility of the developed algorithm, allowing the evaluation of the algorithm's ability to detect postural changes and its applicability in real-world settings.

5 Results

5.1 Categorisation

Given the lack of labelled data to directly calculate the accuracy of the categorisation algorithm, the accuracy was inferred through comparison with the output from the ActivePal device. The approach taken involves estimating the overall accuracy by the overlapping portions between the ActivePal and the code output.

FILE	P002	P003	P004	P006	P007	P008	P009	P011	P012	P013	AVERAGE
PREVIOUS OVERLAP	48.45	84.68	66.39	47.97	58.55	85.55	84.51	70.22	63.87	63.67	70.56
NEW OVERLAP	55.1	85.06	72.45	85.25	66.31	93.57	91.86	75.93	68.5	70.26	76.23

Table 1~ Individual overlap percentages for each participant compared to the ActivePal.

The overlap with the new code output and the ActivePal is 5.67% higher compared to the old codes output. It is to be noted this did deviate a lot between participants with highs of 93.57 and lows of 55.1 and differences between the old and new code varying from 37.28% and 0.38%.

Sleep detection

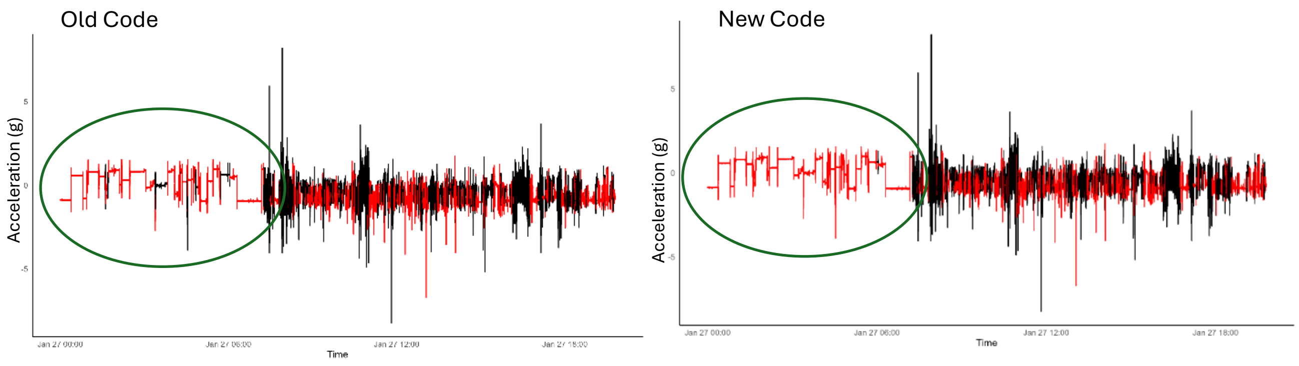


Figure 2 ~ Raw accelerometer data where red is sitting and black is standing bouts. Green circles highlight misclassification.

The old code struggled with categorising bouts when sleeping as seen above. Figure 2 shows mildly restless sleep showcasing the issue and the difference sleep detection has made to the accuracy of the data even with a basic implementation this also highlights a weakness of the original code.

Sedup function

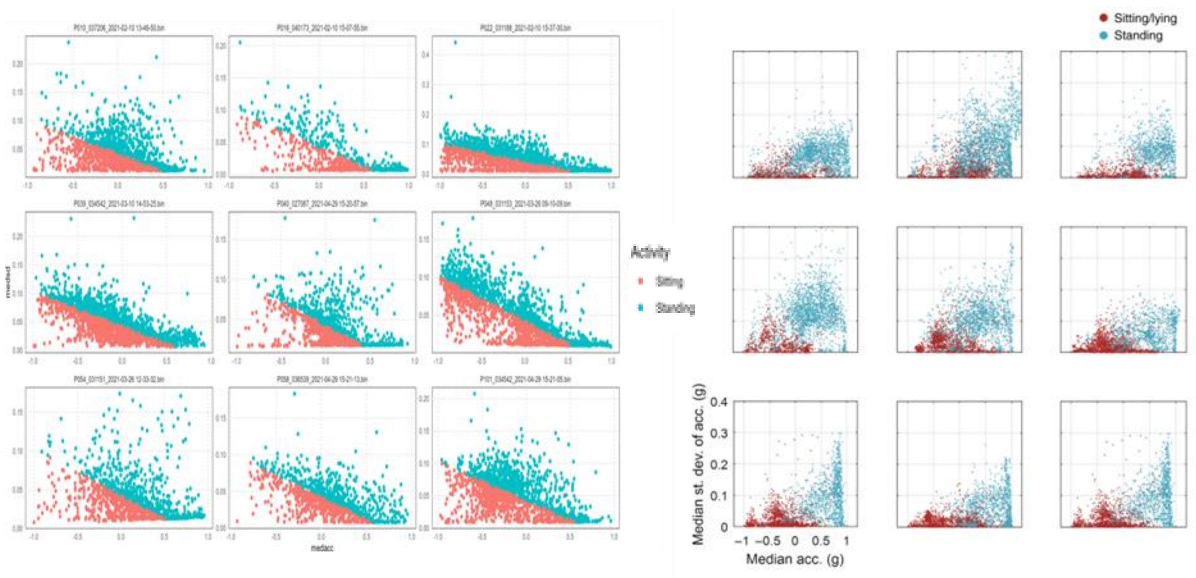


*Figure 3 ~ output seen left and data from research right (*Straczkiewicz *et al, 2020). Standing is blue and sitting/laying is red*

The plot in the research differs from the output seen. In the research, they saw a distinct grouping of the standing data having a median acceleration around 1g and sitting having a deviation closer to 0 which wasn't seen in the data instead in the real data there's no clear split instead one large grouping is formed and the static thresholds then split the grouping.

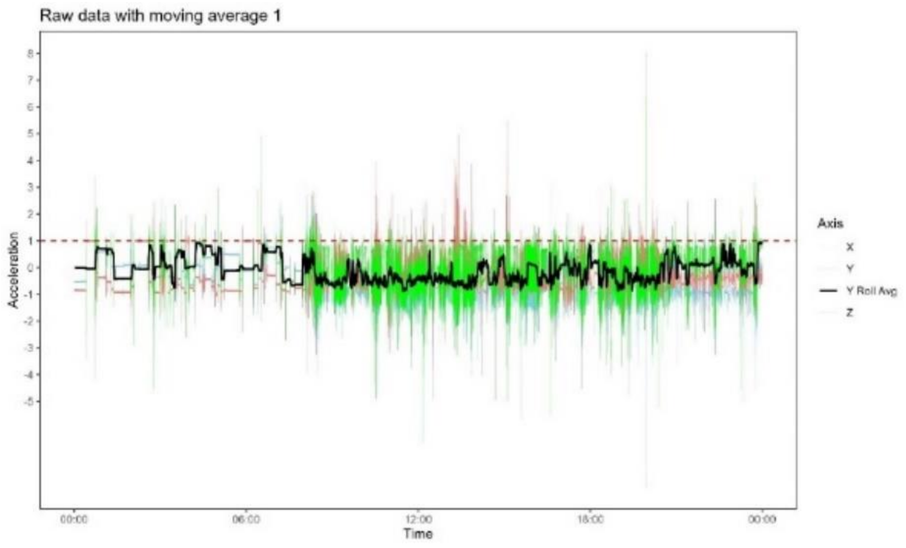


Figure 4 ~ Raw accelerometer data. Grey is the moving average of Y every 60seonds. Red line is one g

Figure 4 gives insight to why the research saw differences in median acceleration as the moving average in this example rarely approaches 1g excluding when asleep and late at night meaning if the data never goes near 1g then the median acceleration won't be near it either.

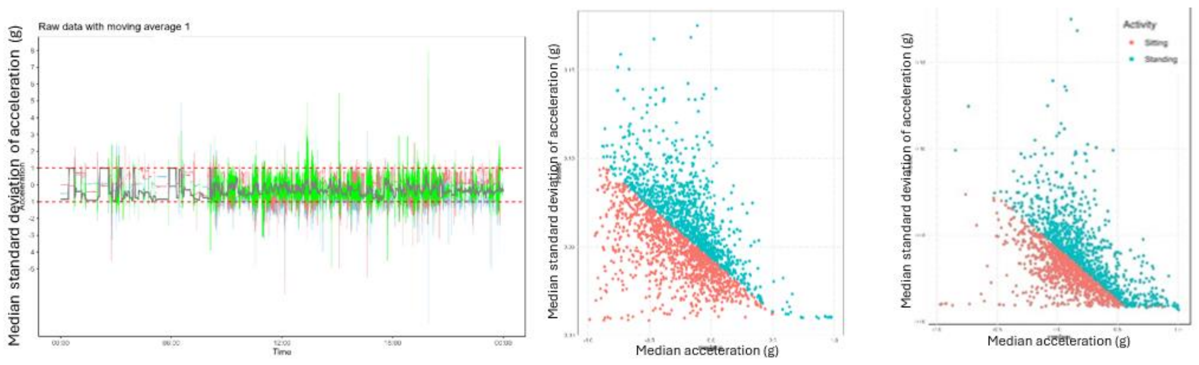


Figure 5~ The line graph is raw data with the red lines at 1 and -1 with a grey line for the morving averge of Y. The scatter plots are the median acceleration by standard deviation of Y for data accounting for an upside down device(right) and one not accouting(left)

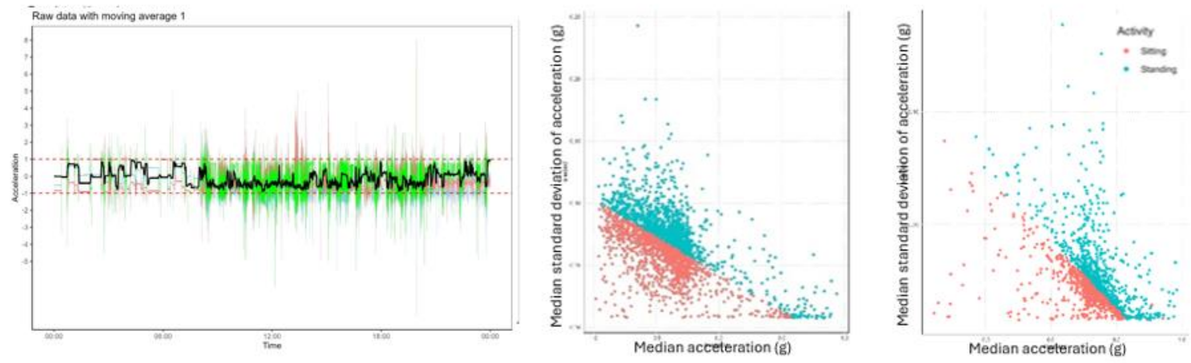


*Figure 6~ The line graph is raw data with the red lines at 1 and -1 with a grey line for the morving averge of Z. The scatter plots are the median acceleration by standard deviation of Z for data accounting for an upside down device(right) and one not accouting(left)***.**

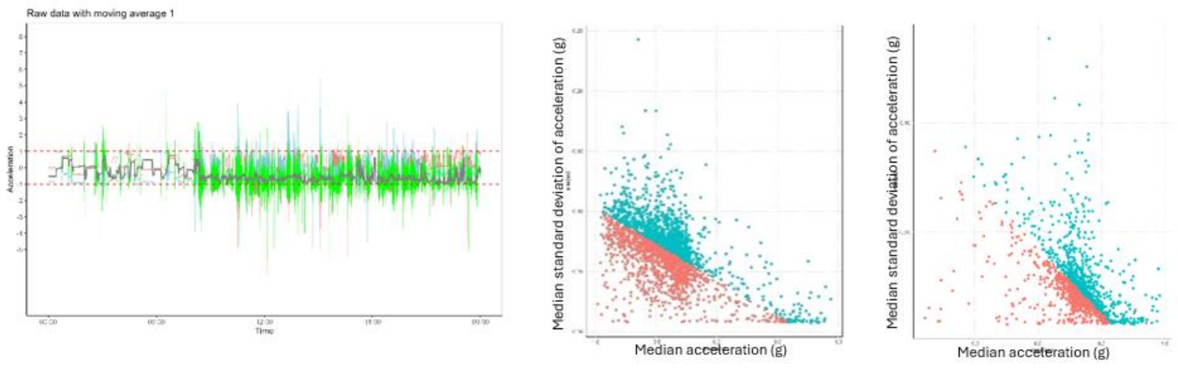


Figure 7~ The line graph is raw data with the red lines at 1 and -1 with a grey line for the moving average of X. The scatter plots are the median acceleration by standard deviation of X for data accounting for an upside-down device(right) and one not accounting(left).

Given that the device may be reading upside down as seen in figure 1 the data was multiplied by negative one to get the y axis to 1g when facing the floor. No difference was seen with the data corrected for an upside-down device regardless of what axis.

5.2 Code

	OLD CODE	NEW CODE
USER	3380.8 s	3651.54 s
SYSTEM	350.33 s	361.21 s
ELAPSED	3767.12 s	4005.3 s

Table 2~Run time of the old and new code with six participants with a combined file size of 2.47GB

The computer running the code had a CPU with 6 cores at 3.2ghz, 12 logical processers and 8gb ddr4 ram at 2133 MHz with an SSD drive and 8GB GPU. This is to test that the efficiency of the code has been maintained. The system timer stayed relatively with all the time gained in the user timer. During the run time of the code the disk and ram saw the highest utilisation showing the bottle necks are the reading and writing of data and then keeping the data in random access. A 237.18 second increase for a 2.47GB file size and when the entire dataset is over 50GB this will amount to an even larger time difference showing how the efficiency wasn't as maintained as aimed.

5.3 Burstiness and health

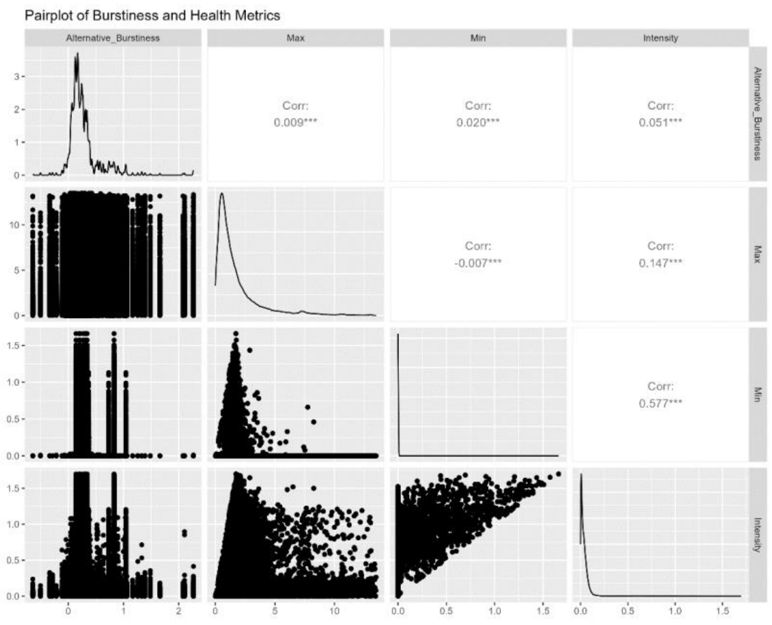


Figure 7~ Pair plot of burstiness and Max representing the peak level of physical activity detected, Min indicating the lowest or no activity, and Intensity measuring the overall strength or magnitude of movement during a specific time-period.

The purpose of the code is to then investigate the relationship between burstiness and health. Other health metrics could be collected alongside the accelerometer data to get a more applicable measure for health but in the dataset the metrics chosen were min and max stats of activity and intensity. There wasn't any correlation between the metrics chosen and burstiness. This could be due to the metrics chosen or the data as it remains unsure if the output is trustworthy.

6 Discussions

6.1 Categorisation

It was expected to be a difference between the research data and seen data but there is such a substantial difference. The data seems to be okay with the issue prevailing regardless of which axis is being used and inverting the data. the issue could be the implementation of the function that does the categorisation or a function before the categorisation as the raw data looks fine and it was tested a large dataset.

Although the difference is worrying the accuracy based of the ActivePal data seems to be within a respectable range but not near the accuracy given in the sedup paper of 90% excluding the accuracy of 90% was seen in some participants. The reason for this wasn't discovered prompting more investigation to be done. Given that the ActivePal had a reported accuracy of 95% and as the overlap was 76% it can be assumed the code has an accuracy of at least 72.2% (Lyden et al, 2017).

6.2 Burstiness

One promising avenue is to introduce a memory parameter into the burstiness scoring, as seen in studies like Karsai et al. (2018). The memory parameter could track whether longer bursts of activity are likely to be followed by similarly long bursts or shorter ones. This pattern recognition could serve as a proxy for fatigue or recovery times, which are indicators of health. For instance, a high memory coefficient may suggest a longer recovery period between bursts of intense activity, which could be tied to an individual's fitness or health state.

6.3 Sleep

\`Sedup\` wasn't designed with sleep in mind hence why the sleep detection is a necessity as seen in figure 2 when participants are sleeping even when laying down, they won't be still and often the wrist will be in an awkward position meaning it's not facing straight down and therefore gravity won't be applying its force resulting in the median acceleration not being equal to 1 which is what sedup relies on.

The sleep detection was a good improvement to the code but due to its basic implementation it has the potential to miss changes in bouts during the night say if someone got up to get some water at 3 in the morning that would be ignored on the other hand it can be assumed most people will wake up after 6 meaning there is still gains to be made by implementing a more sophisticated sleep stats/detection method although it would likely heavily increase the run time of the code.

While the current sleep detection algorithm successfully identifies sleep based on time-of-day patterns, integrating more sophisticated algorithms, such as those from Van Hees et al. (2015), could significantly improve sleep classification accuracy. The Van Hees algorithm utilises accelerometer data in conjunction with statistical models to provide a more nuanced detection of sleep phases. Implementing such an algorithm would increase the computational demands, but it would also result in a more accurate classification of sleep patterns.

6.4 Code

There was some enhancements mentioned that weren't implemented such as the use of the data.table library and parallel processing. It wasn't feasible to use these methods for various reasons. The data.table library wasn't used as the code was the implementation of change how every data frame was created would be to laborious and the parallel processing in R isn't as sophisticated or simple as python.

R has been an effective environment for this project, but it faces limitations, particularly when working with larger datasets. Python presents a more scalable and efficient alternative, especially with libraries like Pandas and NumPy, which are optimised for handling large datasets. These libraries offer better memory management and computational efficiency, particularly with larger datasets where R tends to struggle. Moreover, Python's lazy evaluation and parallel processing capabilities make it a superior choice for the types of data processing involved in accelerometer analysis. Transitioning to Python could drastically reduce runtime and computational overhead, especially during file reading and data manipulation processes, which are among the most time-consuming operations in the current R implementation. Switching from one software to another during an ongoing project can be a time-intensive process. When users are already familiar with a particular software, it is often more efficient and less disruptive to continue using the existing platform rather than adopting something new. Sticking with what you know can save time, minimize learning curves, and reduce the risk of errors or delays caused by adapting to unfamiliar tools.

Additionally, hardware improvements such as replacing an SSD with an NVMe drive would also contribute to performance gains. NVMe drives can offer up to three times faster read and write speeds compared to traditional SSDs, which would accelerate data input/output operations and further reduce code execution time.

7 Conclusions

This dissertation set out to improve an existing R codebase used for posture classification and physical activity monitoring through GENEActiv accelerometer data. The key challenges faced by the original code included its inability to handle non-integer frequencies and the absence of features like burstiness scoring and sleep detection. To address this, several enhancements were introduced, resulting in a more robust and versatile tool for accelerometer data analysis.

The first improvement involved modifying the code to handle fractional frequencies. This was essential because many datasets involve non-integer frequencies, which the original code could not process correctly. By debugging and restructuring key functions, the enhanced code now processes such datasets efficiently without generating errors. To do this the calculation for standard deviation in the sedup function was altered to not use a matrix which was causing the issues. This improvement has expanded the usability of the code in diverse research scenarios where data sampling rates may vary.

Another contribution of this project was the integration of a burstiness measure. This metric was implemented to quantify periods of sitting followed by standing, providing researchers with insights into behavioural patterns.

In addition to burstiness, a sleep detection algorithm was incorporated into the code to improve the classification of nighttime postures. This feature was designed to identify likely periods of sleep based on time-of-day data. Although the implementation used a simple time-window-based heuristic to detect sleep (targeting the hours between midnight and 6 AM), it showed promising results in increasing posture accuracy. While the method worked well in this demographic, it highlighted the need for further refinement to account for irregular sleep schedules, such as those of shift workers or individuals with disrupted sleep cycles.

The enhanced code's performance was validated against data from the activPAL device, regarded as a good criterion in posture classification. The comparison between the enhanced

R code and activPAL data confirmed that the improvements had successfully increased classification accuracy. In particular, the results showed a reduction in misclassification errors for standing versus sitting postures, with the enhanced code achieving classification accuracies exceeding 85% in some cases. This validation added credibility to the code, demonstrating that the enhancements provided reliable and accurate posture classification.

Despite these successes, several limitations remain. The current sleep detection algorithm, while functional, relies on static time windows and does not account for individual differences in sleep patterns. Similarly, while the burstiness measure offers valuable insights into activity irregularity, its full potential could be realised by integrating additional variables, such as step counts, to give a more holistic picture of physical activity.

In conclusion, this project has successfully enhanced the original posture classification code by introducing important new features, improving data handling, and validating performance through comparison with activPAL data. The improved code is now more robust, versatile with the use of packaging it has also decreased chance of breaking again whilst making it easier to fix. It enables researchers to explore physical activity and posture data in greater detail. Looking ahead, future improvements should focus on further refining the sleep detection algorithm, validating the tool across more diverse populations, and exploring the integration of additional data sources, such as multi-sensor devices, to provide even deeper insights into physical activity and health outcomes.

Bibliography

Activinsights. (2022). *GENEActiv instructions for use* (Version 20Oct22) [PDF]. Activinsights. https://activinsights.com/wp-content/uploads/2022/10/GENEActiv-Instructions-for-Use_20Oct22.pdf

Barabási, A. L. (2005). The origin of bursts and heavy tails in human dynamics. *Nature*, 435(7039), 207-211. https://doi.org/10.1038/nature03459

Biswas, A., Oh, P. I., Faulkner, G. E., Bajaj, R. R., Silver, M. A., Mitchell, M. S., & Alter, D. A. (2015). Sedentary time and its association with risk for disease incidence, mortality, and hospitalization in adults: A systematic review and meta-analysis. *Annals of Internal Medicine*, 162(2), 123-132. https://doi.org/10.7326/M14-1651

Bouten, C. V. C., Westerterp, K. R., Verduin, M., & Janssen, J. D. (1994). Assessment of energy expenditure for physical activity using a triaxial accelerometer. *Medicine and Science* *in Sports and Exercise*, 26(12), 1516-1523. https://doi.org/10.1249/00005768-199412000-00014

Goh, K.-I., & Barabási, A.-L. (2008). Burstiness and memory in complex systems. *EPL (Europhysics Letters)*, 81(4), 48002. https://doi.org/10.1209/0295-5075/81/48002

Holme, P. (2018). Temporal networks. In R. Alhajj & J. Rokne (Eds.), *Encyclopedia of Social Network Analysis and Mining* (pp. 2188-2202). Springer, New York. https://doi.org/10.1007/978-1-4614-7163-9_42-1

Karas, M., Bai, J., Strączkiewicz, M., Harezlak, J., Glynn, N. W., Harris, T., & Urbanek, J. K. (2019). Accelerometry data in health research: Challenges and opportunities. *Statistics in Biosciences*, 11(2), 210-237. https://doi.org/10.1007/s12561-018-9227-2

Kim, E.-K., & Jo, H.-H. (2016). Measuring burstiness for finite event sequences. *arXiv preprint arXiv:1604.01125*.

Karsai, M., Jo, H.-H., & Kaski, K. (2018). Measures and characterisations. In *Bursty Human Dynamics*. SpringerBriefs in Complexity. Springer, Cham. https://doi.org/10.1007/978-3-319-68540-3_2

Lyden, K., Keadle, S. K., Staudenmayer, J., & Freedson, P. S. (2017). The activPALTM accurately classifies activity intensity categories in healthy adults. *Medicine & Science in Sports & Exercise*, 49(5), 1022-1028. https://doi.org/10.1249/MSS.0000000000001177

Mannini, A., Intille, S. S., Rosenberger, M., Sabatini, A. M., & Haskell, W. (2013). Activity recognition using a single accelerometer placed at the wrist or ankle. *Medicine & Science in Sports & Exercise*, 45(11), 2193-2203. https://doi.org/10.1249/MSS.0b013e31829736d6

Owen, N., Healy, G. N., Matthews, C. E., & Dunstan, D. W. (2010). Too much sitting: The population health science of sedentary behavior. *Exercise and Sport Sciences Reviews*, 38(3), 105-113. https://doi.org/10.1097/JES.0b013e3181e373a2

Rowlands, A. V., Yates, T., Olds, T. S., Davies, M., Khunti, K., & Edwardson, C. L. (2016). Sedentary Sphere: Wrist-worn accelerometer-brand independent posture classification. *Medicine & Science in Sports & Exercise*, 48(4), 748-754. https://doi.org/10.1249/MSS.0000000000000857

Sani, S., Wiratunga, N., Massie, S., & Cooper, K. (2017). Analysis of deep and shallow feature representations for accelerometer data on human activity recognition. In D. Aha & J. Lieber (Eds.), *Case-Based Reasoning Research and Development* (pp. 330-344). Springer, Cham. https://doi.org/10.1007/978-3-319-61030-6_23

Shoaib, M., Bosch, S., Incel, O. D., Scholten, H., & Havinga, P. J. M. (2021). Sensor-based human activity recognition in the real world using long short-term memory (LSTM) recurrent neural networks. *Sensors*, 21(16), 5564. https://doi.org/10.3390/s21165564

Straczkiewicz, M., Glynn, N. W., Zipunnikov, V., & Harezlak, J. (2020). SedUp: Recognition of sedentary and upright postures using wrist-worn accelerometers. *Journal for* *the Measurement of Physical Behaviour*, 3(4), 256-264. https://doi.org/10.1123/jmpb.2020-0043

Trost, S. G., McIver, K. L., & Pate, R. R. (2005). Conducting accelerometer-based activity assessments in field-based research. *Medicine and Science in Sports and Exercise*, 37(11), S531-S543. https://doi.org/10.1249/01.mss.0000185657.86065.98

Trost, S. G., Zheng, Y., & Wong, W. K. (2014). Prediction of physical activity type using wrist- and hip-worn accelerometers. *Medicine and Science in Sports and Exercise*, 46(1), 204-213. https://doi.org/10.1249/MSS.0000000000000119

Zablocki, R. W., Hartman, S. J., Di, C., Zou, J., Carlson, J. A., Hibbing, P. R., ... & Natarajan, L. (2024). Using functional principal component analysis (FPCA) to quantify sitting patterns derived from wearable sensors. *International Journal of Behavioral Nutrition and Physical Activity*, 21(48). https://doi.org/10.1186/s12966-024-01585-8

Young, D. R., Hivert, M. F., Alhassan, S., Camhi, S. M., Ferguson, J. F., Katzmarzyk, P. T., ... & Jakicic, J. M. (2016). Sedentary behavior and cardiovascular morbidity and mortality: A science advisory from the American Heart Association. *Circulation*, 134(13), e262-e279. https://doi.org/10.1161/CIR.0000000000000440

Appendix

GitHub repository with all code used to create plots along with the previous version of the code.

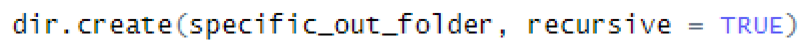
https://github.com/Summer-Project-SEDUP/ALL-CODE

GitHub repository with packed code and instructions on how to use in the readme file.

https://github.com/Summer-Project-SEDUP/MYDISS

Changes to the code report

Changes made to the code



```
dir.create(specific_out_folder, recursive = TRUE)
```

Only change was setting recursive = True to allow the code to create the diretory instead of you having to create it first.

New:

```
sdv <- sapply(split(y, rep(1:(length(y) / fs), each = fs)), sd)
```

Old:

```
sdv <- apply(matrix(y, nrow = fs), 2, sd)
```

This is what caused the fractonal frequencies to break the code as it would try create a matrix with nrow = fs meaning rows equal to frequency and a matrix cant have half a row.

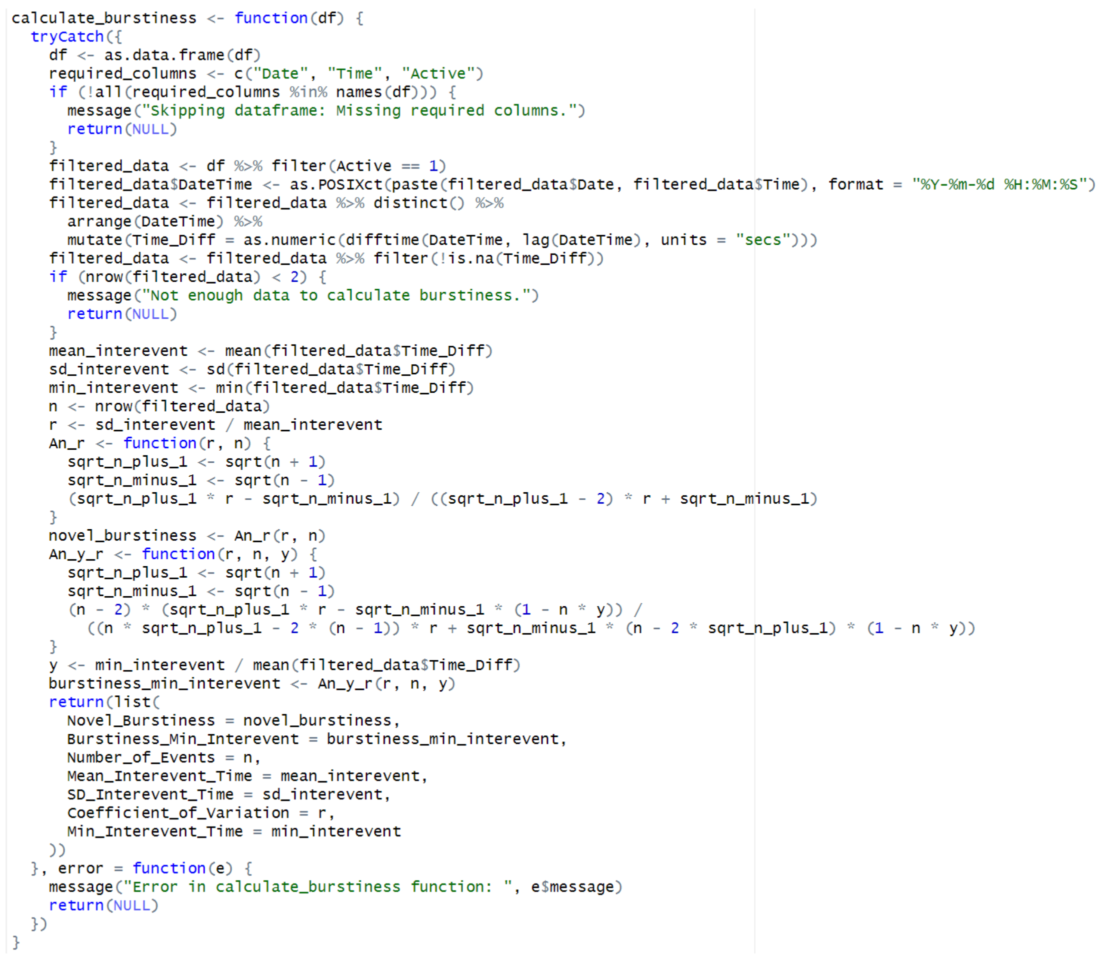
The change in the code splits the data into lists which then act as the rows to the matrix

```
x[times >= 0 & times < 6] <- 0
```

This line of code just says if the hour is less than 6 it sets posture to 0 (laying)

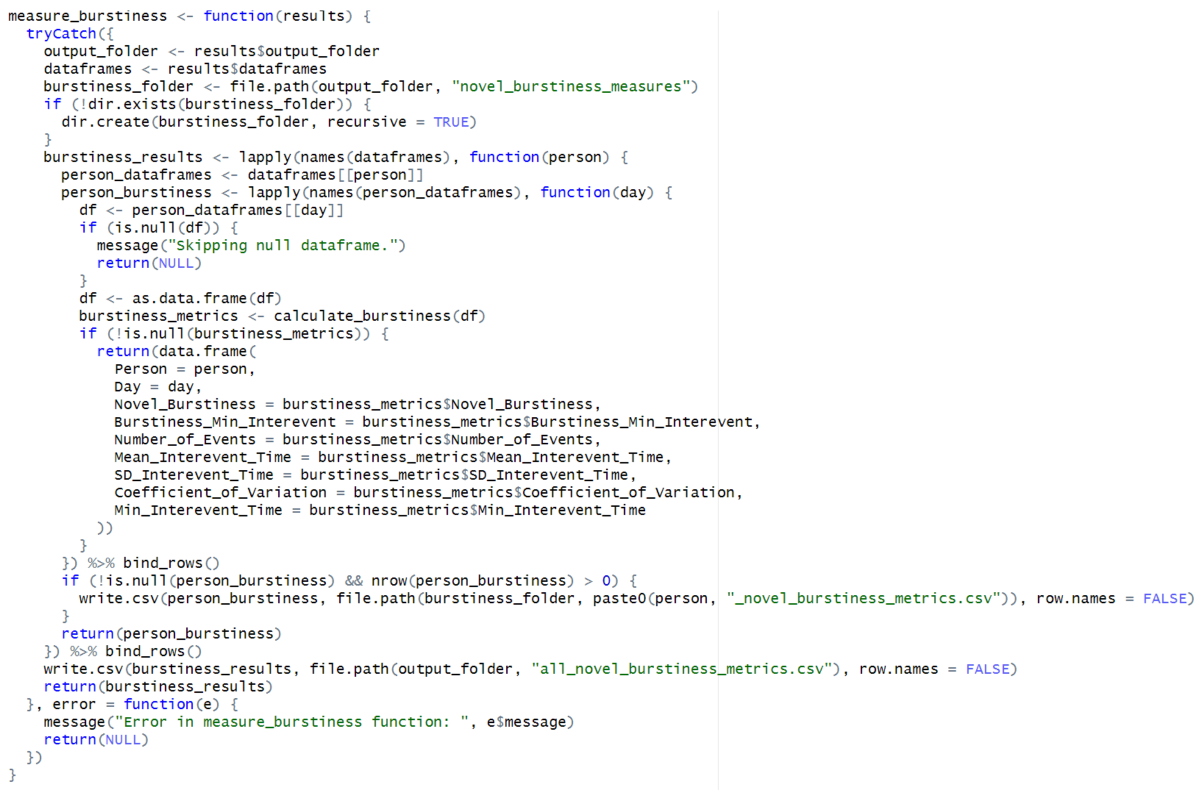
```
calibration <- function(binfile) {
  tryCatch({
    calibration_sphere <- GENEActiv.calibrate(binfile, printsummary = FALSE)
    calibration_offset <- calibration_sphere$offset
    calibration_scale <- calibration_sphere$scale
    return(c(calibration_scale, calibration_offset))
  }, error = function(e) {
    message("Error in calibration function: ", e$message)
    return(NULL)
  })
}
```

Trycatch is a simple function in R that will say where and what went wrong with the code making future trouble shooting easier.



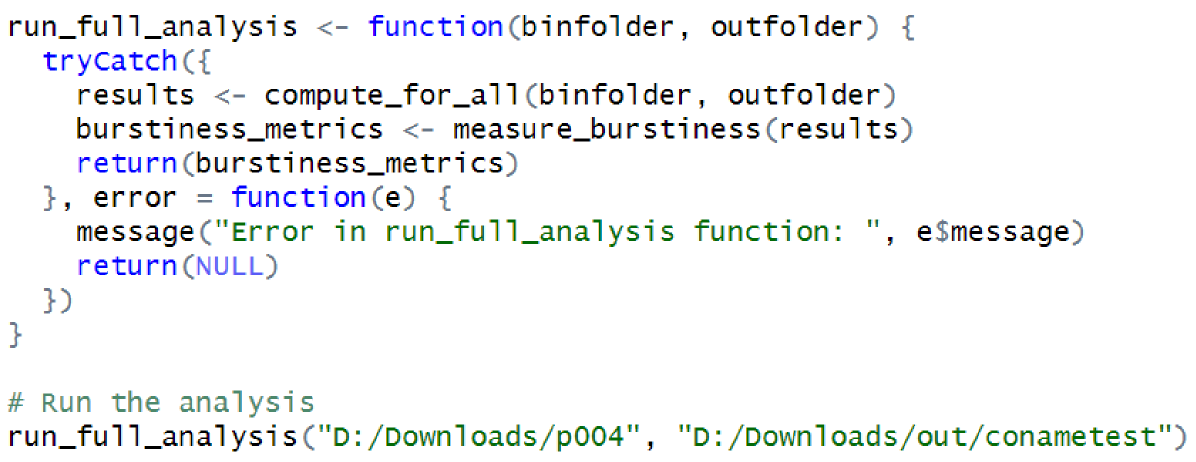
```r
calculate_burstiness <- function(df) {
  tryCatch({
    df <- as.data.frame(df)
    required_columns <- c("Date", "Time", "Active")
    if (!all(required_columns %in% names(df))) {
      message("Skipping dataframe: Missing required columns.")
      return(NULL)
    }
    filtered_data <- df %>% filter(Active == 1)
    filtered_data$DateTime <- as.POSIXct(paste(filtered_data$Date, filtered_data$Time), format = "%Y-%m-%d %H:%M:%S")
    filtered_data <- filtered_data %>% distinct() %>%
      arrange(DateTime) %>%
      mutate(Time_Diff = as.numeric(difftime(DateTime, lag(DateTime), units = "secs")))
    filtered_data <- filtered_data %>% filter(!is.na(Time_Diff))
    if (nrow(filtered_data) < 2) {
      message("Not enough data to calculate burstiness.")
      return(NULL)
    }
    mean_interevent <- mean(filtered_data$Time_Diff)
    sd_interevent <- sd(filtered_data$Time_Diff)
    min_interevent <- min(filtered_data$Time_Diff)
    n <- nrow(filtered_data)
    r <- sd_interevent / mean_interevent
    An_r <- function(r, n) {
      sqrt_n_plus_1 <- sqrt(n + 1)
      sqrt_n_minus_1 <- sqrt(n - 1)
      (sqrt_n_plus_1 * r - sqrt_n_minus_1) / ((sqrt_n_plus_1 - 2) * r + sqrt_n_minus_1)
    }
    novel_burstiness <- An_r(r, n)
    An_y_r <- function(r, n, y) {
      sqrt_n_plus_1 <- sqrt(n + 1)
      sqrt_n_minus_1 <- sqrt(n - 1)
      (n - 2) * (sqrt_n_plus_1 * r - sqrt_n_minus_1 * (1 - n * y)) /
        ((n * sqrt_n_plus_1 - 2 * (n - 1)) * r + sqrt_n_minus_1 * (n - 2 * sqrt_n_plus_1) * (1 - n * y))
    }
    y <- min_interevent / mean(filtered_data$Time_Diff)
    burstiness_min_interevent <- An_y_r(r, n, y)
    return(list(
      Novel_Burstiness = novel_burstiness,
      Burstiness_Min_Interevent = burstiness_min_interevent,
      Number_of_Events = n,
      Mean_Interevent_Time = mean_interevent,
      SD_Interevent_Time = sd_interevent,
      Coefficient_of_Variation = r,
      Min_Interevent_Time = min_interevent
    ))
  }, error = function(e) {
    message("Error in calculate_burstiness function: ", e$message)
    return(NULL)
  })
}
```

The calculate_burstiness function processes a dataframe (df) to compute burstiness metrics, which measure the variability in the timing of active events. It begins by checking that the dataframe has the necessary columns (Date, Time, and Active), and filters it to include only rows where activity is marked as Active == 1. The function then combines the Date and Time columns into a DateTime column, sorts it, and calculates the time difference (Time_Diff) between consecutive active events. If there are fewer than two data points, it exits early, as burstiness cannot be calculated with insufficient data. The function proceeds by calculating the mean, standard deviation, and minimum interevent times, and computes the coefficient of variation (r), which is the ratio of the standard deviation to the mean. Using these metrics, it calculates two burstiness measures: a novel burstiness (An) and another burstiness measure accounting for minimum interevent time (An_y). If any errors occur during execution, they are caught and logged with tryCatch.



```r
measure_burstiness <- function(results) {
  tryCatch({
    output_folder <- results$output_folder
    dataframes <- results$dataframes
    burstiness_folder <- file.path(output_folder, "novel_burstiness_measures")
    if (!dir.exists(burstiness_folder)) {
      dir.create(burstiness_folder, recursive = TRUE)
    }
    burstiness_results <- lapply(names(dataframes), function(person) {
      person_dataframes <- dataframes[[person]]
      person_burstiness <- lapply(names(person_dataframes), function(day) {
        df <- person_dataframes[[day]]
        if (is.null(df)) {
          message("Skipping null dataframe.")
          return(NULL)
        }
        df <- as.data.frame(df)
        burstiness_metrics <- calculate_burstiness(df)
        if (!is.null(burstiness_metrics)) {
          return(data.frame(
            Person = person,
            Day = day,
            Novel_Burstiness = burstiness_metrics$Novel_Burstiness,
            Burstiness_Min_Interevent = burstiness_metrics$Burstiness_Min_Interevent,
            Number_of_Events = burstiness_metrics$Number_of_Events,
            Mean_Interevent_Time = burstiness_metrics$Mean_Interevent_Time,
            SD_Interevent_Time = burstiness_metrics$SD_Interevent_Time,
            Coefficient_of_Variation = burstiness_metrics$Coefficient_of_Variation,
            Min_Interevent_Time = burstiness_metrics$Min_Interevent_Time
          ))
        }
      }) %>% bind_rows()
      if (!is.null(person_burstiness) && nrow(person_burstiness) > 0) {
        write.csv(person_burstiness, file.path(burstiness_folder, paste0(person, "_novel_burstiness_metrics.csv")), row.names = FALSE)
      }
      return(person_burstiness)
    }) %>% bind_rows()
    write.csv(burstiness_results, file.path(output_folder, "all_novel_burstiness_metrics.csv"), row.names = FALSE)
    return(burstiness_results)
  }, error = function(e) {
    message("Error in measure_burstiness function: ", e$message)
    return(NULL)
  })
}
```

The measure_burstiness function calculates burstiness metrics for multiple individuals based on daily activity data. It first ensures that an output directory exists for storing the burstiness results. The function iterates over each person and their respective days of data, applying the calculate_burstiness function to compute metrics such as novel burstiness, burstiness considering minimum interevent time, mean interevent time, and standard deviation. For each person, it saves the daily burstiness results in a CSV file. After processing all individuals, the function consolidates the data into a master CSV file. To prevent interruptions, a tryCatch block is used to handle any errors, logging issues without halting the process.



```r
run_full_analysis <- function(binfolder, outfolder) {
  tryCatch({
    results <- compute_for_all(binfolder, outfolder)
    burstiness_metrics <- measure_burstiness(results)
    return(burstiness_metrics)
  }, error = function(e) {
    message("Error in run_full_analysis function: ", e$message)
    return(NULL)
  })
}

# Run the analysis
run_full_analysis("D:/Downloads/p004", "D:/Downloads/out/conametest")
```

the final function just ties it together so it can be run without the burstiness measure by running compute_for_all(binfolder, outfolder).

Finished code

```
# Load necessary libraries
library(zoo)         # For working with time series data
library(GENEAread)   # For reading GENEActiv .bin files
library(dplyr)       # For data manipulation
# Calibration function


calibration <- function(binfile) {
  tryCatch({
    calibration_sphere <- GENEActiv.calibrate(binfile, printsummary = FALSE)
    calibration_offset <- calibration_sphere$offset
    calibration_scale <- calibration_sphere$scale
    return(c(calibration_scale, calibration_offset))
  }, error = function(e) {
    message("Error in calibration function: ", e$message)
    return(NULL)
  })
}


# Function to get interval data
get_interval <- function(v_bin_data, binfile, start, end, calib, freq) {
  tryCatch({
    day_data <- read.bin(binfile, start = start, end = end)$data.out
    day_data <- cbind(cbind(day_data[,1], ((calib[1:3] * day_data[,c('x', 'y', 'z')]) + calib[4:6])), 
day_data[,7])
    n_hours <- (nrow(day_data) / freq / 3600)
    non_wear_hours <- mean(rollapply(day_data, (10 * 60 * freq), non_wear_window, by = 
(freq * 60), by.column = FALSE)) * n_hours
    date <- as.Date(as.POSIXct(day_data[nrow(day_data) / 2, 1], origin = "1970-01-01"))
    day_of_week <- weekdays(date)
    return(list("DataFrame" = day_data, "Date" = date, "WeekDay" = day_of_week, "DayNo" 
= strsplit(start, " ")[[1]][1], "NonWear" = non_wear_hours))
```

```
  }, error = function(e) {
    message("Error in get_interval function: ", e$message)
    return(NULL)
  })
}

# Function to detect non-wear windows
non_wear_window <- function(window) {
  tryCatch({
    window2_start <- 0
    window2_end <- nrow(window) / 2
    window1_start <- window2_end + 1
    window1_end <- nrow(window)
    window1 <- window[window1_start:window1_end,]
    window2 <- window[window2_start:window2_end,]
    temp_in_window1 <- mean(window1[,5])
    temp_in_window2 <- mean(window2[,5])
    stds_in_window1 <- apply(window1[,c('x', 'y', 'z')], 2, sd)
    min_in_window1 <- apply(window1[,c('x', 'y', 'z')], 2, min)
    max_in_window1 <- apply(window1[,c('x', 'y', 'z')], 2, max)
    range_in_window1 <- max_in_window1 - min_in_window1
    t0 <- 26
    status <- 0
    stationairy <- range_in_window1 < 0.05 & stds_in_window1 < 0.013
    if ((sum(stationairy) >= 2) & (temp_in_window1 < t0)) {
      status <- 0
    } else if (temp_in_window1 > t0) {
      status <- 1
    } else {
      if (temp_in_window1 > temp_in_window2) {
        status <- 1
      }
      if (temp_in_window1 < temp_in_window2) {
        status <- 0
      }
    }
    return(status)
  }, error = function(e) {
    message("Error in non_wear_window function: ", e$message)
    return(NULL)
  })
}

# Function to add ENMO (Euclidean Norm Minus One) metric
add_ENMO <- function(day_matrix) {
  tryCatch({
    ENMO <- pmax((((rowSums(day_matrix[,c('x', 'y', 'z')]^2))^(1/2)) - 1), 0)
    day_matrix <- cbind(day_matrix, ENMO)
    return(day_matrix)
  }, error = function(e) {
    message("Error in add_ENMO function: ", e$message)
    return(NULL)
  })
}

# SedUp function for posture detection
SedUp <- function(y, fs, option) {
  tryCatch({
    y <- y*-1
    constant <- 10
    if (option == 1) {
      coeffs <- c(-2.390, 2.542, 42.394)
      uni_thr <- 0.362
      nSec <- 4
```

```
} else if (option == 2) {
  coeffs <- c(-2.420, 2.616, 44.083)
  uni_thr <- 0.372
  nSec <- 6
}
y <- y[1:(floor(length(y) / fs / constant) * fs * constant)]

sdv <- sapply(split(y, rep(1:(length(y) / fs), each = fs)), sd)

window1 <- nSec * constant * fs / 2
window2 <- round(nSec * constant / 2)
medacc <- integer(length(sdv) / constant)
medsd <- integer(length(sdv) / constant)
j1 <- 0
j2 <- 0
for (i in seq(from = 1, to = length(sdv), by = constant)) {
  j1 <- j1 + 1
  i1_1 <- (i - 1) * fs + 1 - window1
  i1_2 <- (i - 1) * fs + 1 + constant * fs + window1
  b <- i1_1:i1_2
  medacc[j1] <- median(y[b[b > 0 & b < length(y)]], na.rm = TRUE)
  j2 <- j2 + 1
  i2_1 <- (i - 1) - window2
  i2_2 <- (i - 1) + 1 + constant + window2
  b <- i2_1:i2_2
  medsd[j2] <- median(sdv[b[b > 0 & b < length(sdv)]], na.rm = TRUE)
}
logit <- coeffs[1] + coeffs[2] * medacc + coeffs[3] * medsd
phat <- exp(logit) / (1 + exp(logit))
standing <- matrix(data = NA, nrow = length(phat), ncol = 1)
standing[phat >= uni_thr] <- 1
standing[phat < uni_thr] <- 0
```

```
    standing <- rep(standing, each = constant * fs)


    return(standing)
  }, error = function(e) {
    message("Error in SedUp function: ", e$message)
    return(NULL)
  })
}

# Function to get raw data for bouts of activity
get_raw_data <- function(D, bouts, freq) {
  tryCatch({
    return(apply(bouts, 1, function(x) { D[x[1]:x[2], ] }))
  }, error = function(e) {
    message("Error in get_raw_data function: ", e$message)
    return(NULL)
  })
}

# Function to extract metrics from a raw bout
extract_metrics <- function(raw_bout, freq) {
  tryCatch({
    volume <- sum(raw_bout[,"ENMO"]) / freq
    intensity <- mean(raw_bout[,"ENMO"])
    timestamp <- strftime(format(as.POSIXct(raw_bout[1,1], origin = "1970-01-01"), tz =
"UTC"), format = "%H:%M:%S")
    duration <- nrow(raw_bout) / freq
    percentiles <- as.vector((quantile(raw_bout[,'ENMO'], probs = seq(0, 1, 0.05))))
    sd <- sd(raw_bout[,"ENMO"])
    wind_sd <- wind_SD(raw_bout[,"ENMO"])
    list_to_return <- list(
      "Volume" = volume,
    "Intensity" = intensity,
    "Time" = timestamp,
    "Duration" = duration,
    "Percentile_5" = percentiles[2],
    "Percentile_10" = percentiles[3],
    "Percentile_15" = percentiles[4],
    "Percentile_20" = percentiles[5],
    "Percentile_25" = percentiles[6],
    "Percentile_30" = percentiles[7],
    "Percentile_35" = percentiles[8],
    "Percentile_40" = percentiles[9],
    "Percentile_45" = percentiles[10],
    "Percentile_50" = percentiles[11],
    "Percentile_55" = percentiles[12],
    "Percentile_60" = percentiles[13],
    "Percentile_65" = percentiles[14],
    "Percentile_70" = percentiles[15],
    "Percentile_75" = percentiles[16],
    "Percentile_80" = percentiles[17],
    "Percentile_85" = percentiles[18],
    "Percentile_90" = percentiles[19],
    "Percentile_95" = percentiles[20],
    "Max" = percentiles[21],
    "Min" = percentiles[1],
    "SD" = sd,
    "Windsorized_SD" = wind_sd
  )
  return(list_to_return)
}, error = function(e) {
  message("Error in extract_metrics function: ", e$message)
  return(NULL)
})
```

```
}

# Function for Windsorized standard deviation
wind_SD <- function(vect) {
  tryCatch({
    top <- as.numeric(quantile(vect, probs = c(0.95)))
    bottom <- as.numeric(quantile(vect, probs = c(0.05)))
    wind_vect <- pmin(pmax(vect, bottom), top)
    return(sd(wind_vect))
  }, error = function(e) {
    message("Error in wind_SD function: ", e$message)
    return(NULL)
  })
}

# Function to compute bouts of activity
compute_bouts <- function(D, freq, timestamps) {
  tryCatch({
    X <- SedUp(D[,3], freq, 1)
    times <- as.numeric(format(as.POSIXct(timestamps, origin = "1970-01-01"), "%H"))
    X[times >= 0 & times < 6] <- 0
    rle_bouts <- rle(X)
    start <- head(cumsum(c(0, rle_bouts$lengths)), -1)
    end <- cumsum(rle_bouts$lengths)
    return(data.frame(start = start, end = end, lab = rle_bouts$values))
  }, error = function(e) {
    message("Error in compute_bouts function: ", e$message)
    return(NULL)
  })
}

# Function to create a ghost table (empty template)
```

```
create_ghost_table <- function(row, date) {
  tryCatch({
    temp_row <- row
    temp_row <- row
    temp_row[seq(27)] <- 0
    ghost_tab <- do.call('rbind', replicate(24, temp_row, simplify = FALSE))
    times <- c("00:00:00", "01:00:00", "02:00:00", "03:00:00", "04:00:00", "05:00:00",
"06:00:00", "07:00:00", "08:00:00", "09:00:00", "10:00:00", "11:00:00", "12:00:00",
"13:00:00", "14:00:00", "15:00:00", "16:00:00", "17:00:00", "18:00:00", "19:00:00",
"20:00:00", "21:00:00", "22:00:00", "23:00:00")
    ghost_tab['Time'] <- times
    ghost_tab['Date'] <- date
    ghost_tab['WeekDay'] <- weekdays(as.Date(date))
    ghost_tab['DayNo'] <- 0
    ghost_tab['NonWear'] <- 0
    return(ghost_tab)
  }, error = function(e) {
    message("Error in create_ghost_table function: ", e$message)
    return(NULL)
  })
}

# Function to compute data for one person
compute_one_person <- function(file_name, binfolder, outfolder) {
  tryCatch({
    binfile <- file.path(binfolder, file_name)
    file_name_without_bin <- strsplit(file_name, '.bin')[[1]]
    specific_out_folder <- file.path(outfolder, file_name_without_bin)
    dir.create(specific_out_folder, recursive = TRUE)
    v_bin_data <- read.bin(binfile, virtual = TRUE)
    freq <- as.numeric(as.character(v_bin_data$header['Measurement_Frequency', 1]))
    calib <- calibration(binfile)
    time_stamp_list <- v_bin_data[["page.timestamps"]]
```

```
d1 <- time_stamp_list[1]
d2 <- time_stamp_list[length(time_stamp_list)]
dates <- c(seq(d1, d2, by = "day"), d2)
days <- paste(seq(length(dates)), "00:00:00")
filenames <- paste(paste('Day', seq(length(dates)), sep = ''), ".csv", sep = "")
person_dataframes <- list()
for (i in (1:(length(filenames) - 1))) {
  start <- days[i]
  end <- days[i + 1]
  D1 <- get_interval(v_bin_data, binfile, start = start, end = end, calib = calib, freq = freq)
  if (nrow(D1$DataFrame) > 3000) {
    bouts <- compute_bouts(D1$DataFrame, freq, timestamps = D1$DataFrame[,1])
    ENMO_tab <- add_ENMO(D1$DataFrame)
    raw_bouts <- get_raw_data(ENMO_tab, bouts, freq)
    if (nrow(bouts) > 1) {
      list_of_bouts <- lapply(raw_bouts, extract_metrics, freq = freq)
      table_of_bouts <- t(do.call("cbind", list_of_bouts))
      rownames(table_of_bouts) <- NULL
      table_of_bouts <- cbind(table_of_bouts, "Date" = as.character(D1$Date))
      table_of_bouts <- cbind(table_of_bouts, "WeekDay" = as.character(D1$WeekDay))
      table_of_bouts <- cbind(table_of_bouts, "DayNo" = D1$DayNo)
      table_of_bouts <- cbind(table_of_bouts, "NonWear" = D1$NonWear)
      table_of_bouts <- cbind(table_of_bouts, "FILE" = file_name)
      table_of_bouts <- cbind(table_of_bouts, "Active" = bouts[,'lab'])
    }
    if (nrow(bouts) == 1) {
      table_of_bouts <- extract_metrics(ENMO_tab, freq)
      table_of_bouts["Date"] <- as.character(D1$Date)
      table_of_bouts["WeekDay"] <- as.character(D1$WeekDay)
      table_of_bouts["DayNo"] <- D1$DayNo
      table_of_bouts["NonWear"] <- D1$NonWear
      table_of_bouts["FILE"] <- file_name
```

```
        table_of_bouts["Active"] <- bouts[,'lab']
        table_of_bouts <- data.frame(table_of_bouts)
      }
      table_of_bouts <- na.omit(table_of_bouts)
      filepath <- file.path(specific_out_folder, filenames[i])
      write.csv(table_of_bouts, filepath, row.names = FALSE)
      person_dataframes[[paste0("Day", i)]] <- table_of_bouts
    }
  }
  temp <- list.files(specific_out_folder, pattern = "*.csv")
  myfiles <- lapply(file.path(specific_out_folder, temp), read.csv)
  complete_file <- do.call('rbind', myfiles)
  complete_file[,'DayNo'] <- complete_file[,'DayNo'] - (min(complete_file[,'DayNo']) - 1)
  complete_file <- na.omit(complete_file)
  write.csv(complete_file, file.path(specific_out_folder, "all_days.csv"), row.names =
FALSE)
  return(person_dataframes)
 }, error = function(e) {
  message("Error in compute_one_person function: ", e$message)
  return(NULL)
 })
}

# Function to compute data for all persons
compute_for_all <- function(binfolder, outfolder) {
 tryCatch({
  all_binfiles <- list.files(binfolder, pattern = "*.bin")
  all_dataframes <- list()
  for (i in all_binfiles) {
   person_dataframes <- compute_one_person(i, binfolder, outfolder)
   all_dataframes[[i]] <- person_dataframes
  }
```

```
    temp <- list.files(outfolder, pattern = "*all_days.csv", recursive = TRUE)
    myfiles <- lapply(file.path(outfolder, temp), read.csv)
    complete_file <- do.call('rbind', myfiles)
    write.csv(complete_file, file.path(outfolder, "all_people.csv"), row.names = FALSE)
    return(list("dataframes" = all_dataframes, "output_folder" = outfolder))
  }, error = function(e) {
    message("Error in compute_for_all function: ", e$message)
    return(NULL)
  })
}

# Function to calculate the novel burstiness measure (An) and the burstiness considering
minimum interevent time (An_y)
calculate_burstiness <- function(df) {
  tryCatch({
    df <- as.data.frame(df)
    required_columns <- c("Date", "Time", "Active")
    if (!all(required_columns %in% names(df))) {
      message("Skipping dataframe: Missing required columns.")
      return(NULL)
    }
    filtered_data <- df %>% filter(Active == 1)
    filtered_data$DateTime <- as.POSIXct(paste(filtered_data$Date, filtered_data$Time),
format = "%Y-%m-%d %H:%M:%S")
    filtered_data <- filtered_data %>% distinct() %>%
      arrange(DateTime) %>%
      mutate(Time_Diff = as.numeric(difftime(DateTime, lag(DateTime), units = "secs")))
    filtered_data <- filtered_data %>% filter(!is.na(Time_Diff))
    if (nrow(filtered_data) < 2) {
      message("Not enough data to calculate burstiness.")
      return(NULL)
    }
    mean_interevent <- mean(filtered_data$Time_Diff)
    sd_interevent <- sd(filtered_data$Time_Diff)
    min_interevent <- min(filtered_data$Time_Diff)
    n <- nrow(filtered_data)
    r <- sd_interevent / mean_interevent
    An_r <- function(r, n) {
      sqrt_n_plus_1 <- sqrt(n + 1)
      sqrt_n_minus_1 <- sqrt(n - 1)
      (sqrt_n_plus_1 * r - sqrt_n_minus_1) / ((sqrt_n_plus_1 - 2) * r + sqrt_n_minus_1)
    }
    novel_burstiness <- An_r(r, n)
    An_y_r <- function(r, n, y) {
      sqrt_n_plus_1 <- sqrt(n + 1)
      sqrt_n_minus_1 <- sqrt(n - 1)
      (n - 2) * (sqrt_n_plus_1 * r - sqrt_n_minus_1 * (1 - n * y)) /
        ((n * sqrt_n_plus_1 - 2 * (n - 1)) * r + sqrt_n_minus_1 * (n - 2 * sqrt_n_plus_1) * (1 - n
* y))
    }
    y <- min_interevent / mean(filtered_data$Time_Diff)
    burstiness_min_interevent <- An_y_r(r, n, y)
    return(list(
      Novel_Burstiness = novel_burstiness,
      Burstiness_Min_Interevent = burstiness_min_interevent,
      Number_of_Events = n,
      Mean_Interevent_Time = mean_interevent,
      SD_Interevent_Time = sd_interevent,
      Coefficient_of_Variation = r,
      Min_Interevent_Time = min_interevent
    ))
  }, error = function(e) {
    message("Error in calculate_burstiness function: ", e$message)
    return(NULL)
  })
```

```
}

# Function to measure burstiness for all dataframes
measure_burstiness <- function(results) {
  tryCatch({
    output_folder <- results$output_folder
    dataframes <- results$dataframes
    burstiness_folder <- file.path(output_folder, "novel_burstiness_measures")
    if (!dir.exists(burstiness_folder)) {
      dir.create(burstiness_folder, recursive = TRUE)
    }
    burstiness_results <- lapply(names(dataframes), function(person) {
      person_dataframes <- dataframes[[person]]
      person_burstiness <- lapply(names(person_dataframes), function(day) {
        df <- person_dataframes[[day]]
        if (is.null(df)) {
          message("Skipping null dataframe.")
          return(NULL)
        }
        df <- as.data.frame(df)
        burstiness_metrics <- calculate_burstiness(df)
        if (!is.null(burstiness_metrics)) {
          return(data.frame(
            Person = person,
            Day = day,
            Novel_Burstiness = burstiness_metrics$Novel_Burstiness,
            Burstiness_Min_Interevent = burstiness_metrics$Burstiness_Min_Interevent,
            Number_of_Events = burstiness_metrics$Number_of_Events,
            Mean_Interevent_Time = burstiness_metrics$Mean_Interevent_Time,
            SD_Interevent_Time = burstiness_metrics$SD_Interevent_Time,
            Coefficient_of_Variation = burstiness_metrics$Coefficient_of_Variation,
            Min_Interevent_Time = burstiness_metrics$Min_Interevent_Time
          ))
        }
      }) %>% bind_rows()
      if (!is.null(person_burstiness) && nrow(person_burstiness) > 0) {
        write.csv(person_burstiness, file.path(burstiness_folder, paste0(person,
"_novel_burstiness_metrics.csv")), row.names = FALSE)
      }
      return(person_burstiness)
    }) %>% bind_rows()
    write.csv(burstiness_results, file.path(output_folder, "all_novel_burstiness_metrics.csv"),
row.names = FALSE)
    return(burstiness_results)
  }, error = function(e) {
    message("Error in measure_burstiness function: ", e$message)
    return(NULL)
  })
}

# Main function to run the entire analysis
run_full_analysis <- function(binfolder, outfolder) {
  tryCatch({
    results <- compute_for_all(binfolder, outfolder)
    burstiness_metrics <- measure_burstiness(results)
    return(burstiness_metrics)
  }, error = function(e) {
    message("Error in run_full_analysis function: ", e$message)
    return(NULL)
  })
}

# Run the analysis
run_full_analysis("D:/Downloads/p004", "D:/Downloads/out/conametest")
```