NeuroKit2: A Python Toolbox for Neurophysiological Signal Processing

Dominique Makowski[1,*], Tam Pham[1], Zen J. Lau[1], Jan C. Brammer[2], François Lespinasse[3, 4], Hung Pham[5], Christopher Schölzel[6], & S.H. Annabel Chen[1, 7, 8]

[1] School of Social Sciences, Nanyang Technological University, Singapore

[2] Behavioural Science Institute, Radboud University, Netherlands

[3] Département de psychologie, Université de Montréal, Canada

[4] Centre de Recherche de l'Institut Universitaire Geriatrique de Montréal

[5] Eureka Robotics, Singapore

[6] Life Science Informatics, THM University of Applied Sciences, Germany

[7] Centre for Research and Development in Learning, Nanyang Technological University, Singapore

[8] Lee Kong Chian School of Medicine, Nanyang Technological University, Singapore

Author Note

* Correspondence concerning this article should be addressed to Dominique Makowski (HSS 04-18, 48 Nanyang Avenue, Singapore; dmakowski@ntu.edu.sg).

Correspondence concerning this article should be addressed to Dominique Makowski, HSS 04-18, 48 Nanyang Avenue, Singapore. E-mail: dmakowski@ntu.edu.sg

Abstract

NeuroKit2 is an open-source, community-driven, and user-friendly Python package dedicated to neurophysiological signal processing with an initial focus on bodily signals (e.g., ECG, EDA, EMG, EOG, PPG etc.). Its design philosophy is centred on user-experience and accessibility to both novice and advanced users. The package provides a consistent set of high-level functions that enable data processing in a few lines of code using validated pipelines, which we illustrate in two examples covering the most typical scenarios, such as an event-related paradigm and an interval-related analysis. The package also includes tools dedicated to specific processing steps such as rate extraction and filtering methods, offering a trade-off between efficiency and fine-tuned control to the user. Rather than focusing on specific signals, NeuroKit2 was developed to provide a comprehensive means for a simultaneous processing of a wide range of signals. Its goal is to improve transparency and reproducibility in neurophysiological research, as well as foster exploration and innovation.

Keywords: Neurophysiology, Biosignals, Python, ECG, EDA, EMG

Word count: 2513

NeuroKit2: A Python Toolbox for Neurophysiological Signal Processing

Neurophysiological measurements increasingly gain popularity in the study of cognition and behavior. These measurements include electroencephalography (EEG), electrocardiography (ECG), electromyography (EMG) and electrodermal activity (EDA). Their popularity is driven by theoretical motivations (e.g., the growth of embodied or affective neuroscience; Kiverstein & Miller, 2015) as well as practical reasons. The latter include low costs (especially compared with other imaging techniques, such as MRI or MEG), ease of use (e.g., portability, setup speed), and the increasing availability of recording devices (e.g., wearables; Yuehong, Zeng, Chen, & Fan, 2016). Moreover, the extraction of meaningful information from neurophysiological signals is facilitated by current advances in signal processing algorithms (Clifton, Gibbons, Davies, & Tarassenko, 2012; Roy et al., 2019). Unfortunately, these algorithms are mostly inaccessible to researchers without experience in programming and signal processing. Moreover, many software tools for neurophysiological analyses are limited to one type of signal (for instance, focused on ECG). This makes it inconvenient for researchers who might have to learn and concurrently rely on different software to process multimodal data.

Another important issue existing in psychology and neuroscience has been coined as the "reproducibility crisis" (Maizey & Tzavella, 2019; Miłkowski, Hensel, & Hohol, 2018; Nosek, Cohoon, Kidwell, & Spies, 2015; Topalidou, Leblois, Boraud, & Rougier, 2015), and has lead to a profound questioning and reassessment from different actors involved (researchers, publishers, fund agencies, …). One of the main identified contributing factor is the actual opacity of data processing, where analysis pipelines are not described in enough details to ensure a full and exact reproduction. One of the suggested response to that issue has been to provide, alongside the study, the analysis script, which in turns opens new challenges. Indeed, these scripts must be shareable (not always feasible with closed-source and proprietary software or programming languages), accessible (enticing documented and well-organized scripts) and reproducible (which is inherently difficult for many software relying on a graphical user interface - GUI - in which the manual point-and-click sequence is hard to automate).

NeuroKit2 addresses these challenges by offering a free, user-friendly, and comprehensive solution for neurophysiological data processing, with an initial focus on bodily signals (including ECG, PPG, RSP, EDA, EMG, EOG) and generic functions that could also support other signal processing such as EEG (for which more specific support is in development). It is an open-source Python package, developed by a multi-disciplinary team that actively invites new collaborators. It aims at being accessible, well-documented, well-tested, cutting-edge, flexible and efficient, allowing users to select from a wide range of validated analysis pipelines as well as creating their own. Historically, *NeuroKit2* is the re-forged successor *NeuroKit1* (Makowski, 2020), taking on its most successful features and design choices, and re-implementing them in a professional and well-thought way.

The package is implemented in Python 3 (Van Rossum & Drake, 2009), which means that *NeuroKit2's* users benefit from an large amount of learning resources and a vibrant community. The package depends on relatively few, well established and robust packages from the Python data analysis ecosystem (Virtanen et al., 2020) such as *NumPy*, *pandas*, *SciPy*, *scikit-learn* and *MatplotLib* (with an additional system of optional dependencies), making *NeuroKit2* itself a viable dependency in other software.

NeuroKit2's source code is available under the permissive MIT license on GitHub (*https://github.com/neuropsychology/NeuroKit*). Its documentation (*https://neurokit2.readthedocs.io/*) is automatically built and rendered from the code and includes guides for installation and contribution, a description of the package's functions, as well as several "hands-on" examples and tutorials (e.g., how to extract and visualize individual heartbeats, how to analyze event-related data etc.). Importantly, users can add new examples by simply uploading a Python notebook (Kluyver et al., 2016) to the GitHub repository. The notebook will automatically be displayed on the website, ensuring easily accessible and evolving documentation. Moreover, users can try out the example notebooks directly in their browser via a cloud-based *Binder* environment (Jupyter et al., 2018). Finally, the issue tracker on GitHub offers a convenient and public forum that allows newcomers and potential collaborators to report bugs, get help and gain insight into the development of the package.

NeuroKit2 aims at being reliable and trustworthy, including peer-reviewed processing pipelines and functions tested against established software such as *BioSPPy* (Carreiras et al., 2015), *hrv under review*, *PySiology* (Gabrieli, Azhari, & Esposito, 2019), *HeartPy* (Gent, Farah, Nes, & Arem, 2019), *systole* (Legrand & Allen, 2020) or *nolds* (Schölzel, 2019). The repository leverages a comprehensive test suite and continuous integration to ensure stability and prevent errors. Thanks to its collaborative and open development, *NeuroKit2* can remain cutting-edge and continuously evolve, adapt, and integrate new methods as they are emerging.

Finally, we believe that the design philosophy of *NeuroKit2* contributes to an efficient (i.e., allowing to achieve a lot with few functions) yet flexible (i.e., enabling fine control and precision over what is done) user interface (API). We will illustrate these claims with two examples of common use-cases (the analysis of event-related and resting state data), and will conclude by discussing how *NeuroKit2* contributes to neurophysiological research by raising the standards for validity, reproducibility and accessibility.

Design Philosophy

As stated above, *NeuroKit2* aims at being accessible to beginners and, at the same time, offering a maximal level of control to experienced users. This is achieved by allowing beginning users to implement complex processing and analyses pipelines with very few functions, while still enabling fine-tuned control and precision over arguments and parameters to more experienced users. In concrete terms, this trade-off is allowed by a API structure organized in three three layers of abstraction.

Low-level: Base Utilities for Signal Processing

The basic building blocks are functions for general signal processing, i.e., filtering, resampling, interpolation, peak detection, etc. These functions are signal-agnostic, and include a lot of parameters (e.g., one can change the filtering method, frequencies, and order, by overwriting the default arguments). Most of these functions are based on established algorithms implemented in *scipy* (Virtanen et al., 2020). Examples of such functions include `signal_filter()`, `signal_interpolate()`, `signal_resample()`, `signal_detrend()`, and `signal_findpeaks()`.

Mid-level: Neurophysiological Processing Steps

The base utilities are used by mid-level functions specific to the different physiological modalities (i.e., ECG, RSP, EDA, EMG, PPG). These functions carry out modality-specific signal processing steps, such as cleaning, peak detection, phase classification or rate computation. Critically, for each type of signal, the same function names are called (in the form `signaltype_functiongoal()`) to achieve equivalent goals, e.g., `*_clean()`, `*_findpeaks()`, `*_process()`, `*_plot()`, making the implementation intuitive and consistent across different modalities.

For example, the `rsp_clean()` function uses `signal_filter()` and `signal_detrend()`, with different sets of default parameters that can be switched with a “method” argument (corresponding to different published or established pipelines). For instance, setting `method="khodadad2018"` will use the cleaning workflow described in Khodadad et al. (2018). However, if a user wants to build their own custom cleaning pipeline, they can use the cleaning function as a template, and tweak the parameters to their desires in the low-level signal processing operations.

High-level Wrappers for Processing and Analysis

The mid-level functions are assembled in high-level "master" functions, that are convenient entry points for new users. For instance, the `ecg_process()` function internally chains the mid-level functions `ecg_clean()`, `ecg_findpeaks()`, `ecg_rate()`. A specific processing pipeline can be selected with the `method` argument, that is then propagated throughout the internal functions. Easily switching between processing pipelines allows for the comparison of different methods, and streamlines critical but time-consuming steps in reproducible research, such as the validation of data preparation and quality control (Quintana, Alvares, & Heathers, 2016). Finally, the package includes convenience meta-functions (e.g., `bio_process`) that enable the combined processing of multiple types of signals at once (e.g., `bio_process(ecg=ecg_signal, eda=eda_signal)`).

Performing an entire set of operations with sensible default parameters in one function can be rewarding, especially for beginners, allowing them to perform cutting-edge processing or replication of research steps without requiring much programming expertise. Moreover, it contributes to the demystification of the usage of "pure" programming tools (as opposed to GUI-based software such as *SPSS*, *Kubios*, or *Acqknowledge*), providing a welcoming framework to further explore the complexities of physiological data processing. Importantly, more advanced users can easily build custom analysis pipelines by using the mid-level functions, allowing for a finer control over the processing parameters. We believe that this implementation is a well-calibrated trade-off between flexibility and user-friendliness.

Examples

In this section, we present two examples that illustrate the most common use-cases. The first example is an event-related paradigm, in which the interest lies in short-term physiological changes related to specific events (see **Figure 1** and **Table 1**). The second example shows how to extract the characteristics of physiological activity during a longer period of time (not necessarily tied to a specific and sudden event). The example datasets are made available with the package and can be downloaded using the `data()` function.

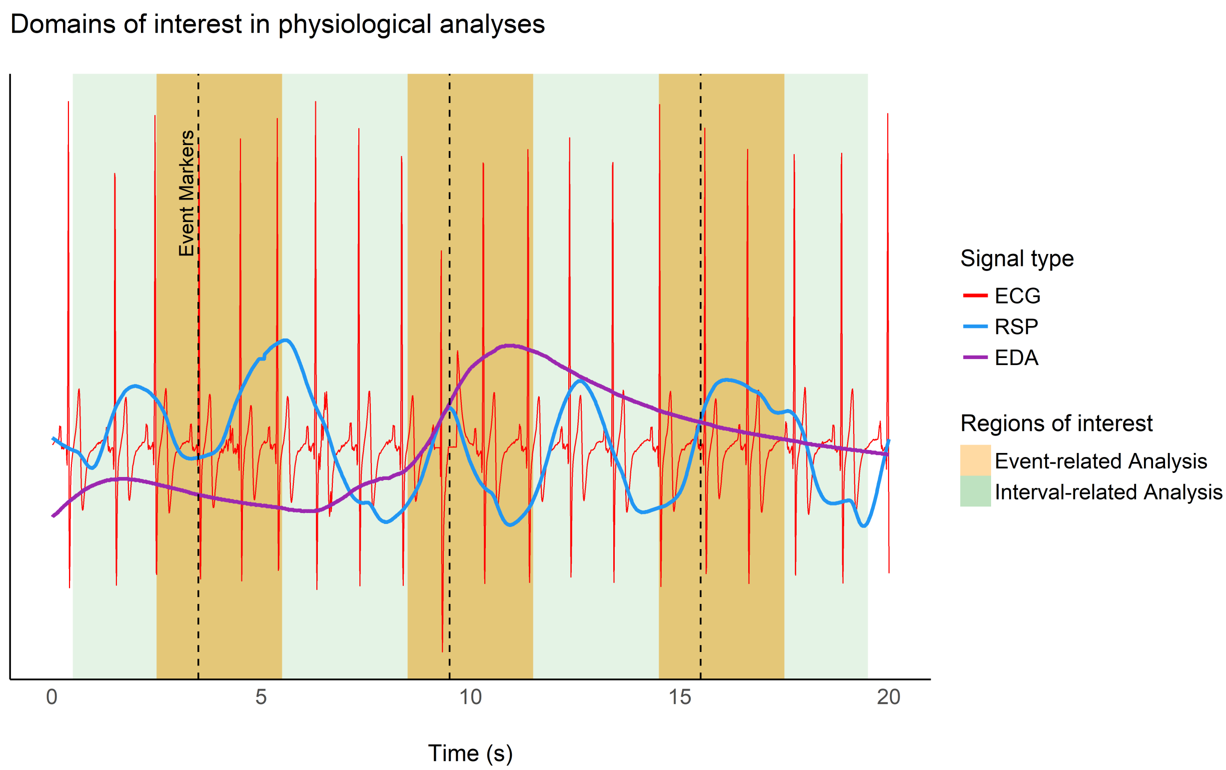

Figure 1. Illustration of the difference between event-related analysis, focusing on activity changes in short windows (the orange rectangles), and interval-related analysis, pertaining to features of large areas, or the whole signal (e.g., the green rectangle).

Event-related Paradigm

This example dataset contains ECG, RSP and EDA signals of one participant who was presented with four emotional images (from the NAPS database; Marchewka, Żurawski, Jednoróg, & Grabowska, 2014), in a typical (albeit highly shortened) experimental psychology paradigm.

Signals are 2.5 minutes long and are recorded at a frequency of 100Hz (note that the sampling rate is low for storage purposes and should be higher in actual recordings, see Quintana et al., 2016). It has 4 channels including three physiological signals, and one corresponding to

Table 1

Examples of features computed in different domains.

Event-related Features	Interval-related Features
ECG Rate Changes (Min, Mean, Max, Time of Min, Max, Trend)	ECG Rate Characteristics (Mean, Amplitude)
RSP Rate Changes (Min, Mean, Max, Time of Min, Max)	Heart Rate Variability (HRV) indices
RSP Amplitude Measures (Min, Mean, Max)	Respiratory Rate Variability (RRV) indices
ECG and RSP Phase (Inspiration/Expiration, Systole/Diastole, Completion)	Respiratory Sinus Arrhythmia (RSA) indices
SCR peak and its characteristics (amplitude, rise time, recovery time)	Number of SCR Peaks and mean amplitude

the marking of events with a photosensor (which signal decreases when a stimulus appeared on the screen).

```python
# Load the package
import neurokit2 as nk

# Download the example dataset
data = nk.data("bio_eventrelated_100hz")

# Process the data
df, info = nk.bio_process(ecg=data["ECG"],
                          rsp=data["RSP"],
                          eda=data["EDA"],
                          sampling_rate=100)

# Find events
conditions = ["Negative", "Neutral", "Neutral", "Negative"]
```

```
events = nk.events_find(event_channel=data["Photosensor"],
                        threshold_keep='below',
                        event_conditions=conditions)

# Epoch the data
epochs = nk.epochs_create(data=df,
                          events=events,
                          sampling_rate=100,
                          epochs_start=-0.1,
                          epochs_end=4)

# Extract event related features
results = nk.bio_analyze(epochs)

# Show subset of results
results[["Condition", "ECG_Rate_Mean", "RSP_Rate_Mean", "EDA_Peak_Amplitude"]]
```

Table 2

Subset of the ouput related to event-related analysis characterizing the pattern of physiological changes related to specific stimuli.

Condition	ECG_Rate_Mean	RSP_Rate_Mean	EDA_Peak_Amplitude
Negative	-0.92	1.41	0.93
Neutral	-3.03	1.25	0.41
Neutral	0.28	0.00	0.02
Negative	-3.34	-1.12	1.06

In this example, after loading the package and the example dataset, each physiological signal is processed using `bio_process()`. Stimulus onsets in the photosensor are detected separately with `events_find()`. Once we have the preprocessed signals and the location of events, we can slice the data into segments corresponding to a time window (ranging from -0.1 to 4 seconds) around each stimulus with `epochs_create()`. Finally, relevant features are computed for each epoch (i.e., each stimulus) by passing them to `bio_analyze()`.

The features include for example the changes in rate of ECG and RSP signals (e.g. maximum, minimum and mean rate after stimulus onset, and the time at which they occur), and the peak characteristics of the EDA signal (e.g., occurrence of skin conductance response (SCR), and if SCR is present, its corresponding peak amplitude, time of peak, rise and recovery time). In addition, respiration and cardiac cycle phases are extracted (i.e., the respiration phase - inspiration/expiration - and cardiac phase - systole/diastole - occurring at the onset of event).

This example shows the straightforward process of extracting features of physiological responses. This pipeline can easily scale up to group-level analysis by aggregating the average of features across participants. In addition to streamlining data analyses, *NeuroKit2* aims to provide researchers an extensive suite of signal features, allowing for precise interpretations in terms of relationship between physiological activity and neurocognitive processes. In this example (see **Table 2**), exposure to negative stimuli, as compared to neutral stimuli, is related to stronger cardiac deceleration, higher skin conductance response, and accelerated breathing rate (note that this descriptive interpretation is given solely for illustrative purposes).

Resting-state Features

The second dataset corresponds to 5 minutes of physiological activity of a human participant at rest (eyes-closed in a seated position), under no specific set of instructions. It contains three channels (ECG, PPG and RSP) sampled at a frequency of 100Hz.

```python
# Load the package
import neurokit2 as nk

# Download the example dataset
data = nk.data("bio_resting_5min_100hz")

# Process the data
df, info = nk.bio_process(ecg=data["ECG"],
                          rsp=data["RSP"],
                          sampling_rate=100)

# Extract features
results = nk.bio_analyze(df)

# Show subset of results
results[["ECG_Rate_Mean", "HRV_RMSSD", "RSP_Rate_Mean", "RSA_P2T_Mean"]]
```

Table 3

Subset of properties characterizing the physiological activity over a period of 5 minutes of resting-state.

ECG_Rate_Mean	HRV_RMSSD	RSP_Rate_Mean	RSA_P2T_Mean
86.39	3.88	15.74	0.01

In this example, the steps of the analysis are identical to the previous example, including loading the package, the dataset and processing the data. The difference is that there is no epoching, as we want to compute features related to the whole dataset (see **Table 3**). Thus, we can directly pass the dataframe to `bio_analyze()`, which will detect that these are not epochs, and compute the appropriate features accordingly. These include for instance the average heart and breathing rate, as well as indices of heart rate variability (HRV) and respiratory sinus arrhythmia (RSA).

This example illustrates a second type of physiological analysis, that we refer to as interval-related (as opposed to event-related). Interval-related analyses compute features of signal variability and activation patterns over a longer-term period of time (typically minutes). *NeuroKit2* allows for the fast creation of a standardized and reproducible pipeline to describe this kind of physiological activity, which can be beneficial for a wide variety of applications.

Discussion

NeuroKit2 is a neurophysiological signal processing software accessible to people with all levels of programming experience and background. Its development is focused on creating an intuitive user-experience, as well as building a collaborative community. It is also a pragmatic answer to the broader need for transparent and reproducible methods in neurophysiology. Its modular structure and organization not only facilitates the use of existing and validated processing pipelines, but also creates a fertile ground for experimentation and innovation.

We expect the package's future evolution to be driven by the communities' needs and the advances in related fields. For instance, although *NeuroKit2* already implements a lot of useful functions for EEG processing (such as entropy and fractal dimensions quantification), its support could be further improved (for example with high-level functions built on top of utilities provided by the leading EEG Python software, namely *MNE*; Gramfort et al., 2013). Possible other future directions include extending the support for other types of bodily signals (e.g., electrogastrography - EGG, electrooculography - EOG) and achieving performance gains for large datasets by using efficient algorithms. Further validation of the available processing pipelines could be made through the (re)analysis of public databases. In line with this objective, the support of standardized data structure formats (e.g. WFDB, BIDS, …) could be extended.

In conclusion, we believe that *NeuroKit2* provides useful tools for anyone who is interested in analyzing physiological data from research-grade hardware as well as wearable "smart health devices". By increasing the autonomy of researchers and practitioners, and by shortening the delay between data collection and results acquisition, *NeuroKit2* could be useful beyond academic research in neuroscience and psychology, including applications such as biofeedback, personal physiological monitoring and exercise science. Finally, we hope that *NeuroKit2* encourages users to become part of a supportive open-science community with diverse areas of expertise rather than relying on closed-source and proprietary software, thus shaping the future of neurophysiology and its related fields.

Conflict of Interest

The authors declare that the research was conducted in the absence of commercial or financial relationships that could constitute a conflict of interest.

Acknowledgements

We would like to thank Prof. C. F. Xavier for inspiration, all the contributors (https://neurokit2.readthedocs.io/en/latest/authors.html), and the users for their support. Additionally, François Lespinasse would like to thank the Courtois Foundation for its support through the Courtois-NeuroMod project (https://cneuromod.ca)

References

Carreiras, C., Alves, A. P., Lourenço, A., Canento, F., Silva, H., Fred, A., & others. (2015). BioSPPy: Biosignal processing in Python. Retrieved from https://github.com/PIA-Group/BioSPPy/

Clifton, D. A., Gibbons, J., Davies, J., & Tarassenko, L. (2012). Machine learning and software engineering in health informatics. In *2012 first international workshop on realizing ai synergies in software engineering (raise)* (pp. 37–41). IEEE.

Gabrieli, G., Azhari, A., & Esposito, G. (2019). PySiology: A python package for physiological feature extraction. In *Neural approaches to dynamics of signal exchanges* (pp. 395–402). Springer Singapore. https://doi.org/10.1007/978-981-13-8950-4_35

Gent, P. van, Farah, H., Nes, N. van, & Arem, B. van. (2019). HeartPy: A novel heart rate algorithm for the analysis of noisy signals. *Transportation Research Part F: Traffic Psychology and Behaviour*, *66*, 368–378. https://doi.org/10.1016/j.trf.2019.09.015

Gramfort, A., Luessi, M., Larson, E., Engemann, D. A., Strohmeier, D., Brodbeck, C., … others. (2013). MEG and eeg data analysis with mne-python. *Frontiers in Neuroscience*, *7*, 267.

Jupyter, Bussonnier, Forde, Freeman, Granger, Head, … Willing. (2018). Binder 2.0 - Reproducible, interactive, sharable environments for science at scale. In Fatih Akici, David Lippa, Dillon Niederhut, & M. Pacer (Eds.), *Proceedings of the 17th Python in Science Conference* (pp. 113–120). https://doi.org/%2010.25080/Majora-4af1f417-011%20

Khodadad, D., Nordebo, S., Mueller, B., Waldmann, A., Yerworth, R., Becher, T., … others. (2018). Optimized breath detection algorithm in electrical impedance tomography. *Physiological Measurement*, *39*(9), 094001.

Kiverstein, J., & Miller, M. (2015). The embodied brain: Towards a radical embodied cognitive neuroscience. *Frontiers in Human Neuroscience*, *9*, 237.

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B. E., Bussonnier, M., Frederic, J., … others. (2016). Jupyter notebooks-a publishing format for reproducible computational workflows. In *ELPUB* (pp. 87–90).

Legrand, N., & Allen, M. (2020). Systole: A python toolbox for preprocessing, analyzing, and synchronizing cardiac data. Retrieved from https://github.com/embodied-computation-group/systole

Maizey, L., & Tzavella, L. (2019). Barriers and solutions for early career researchers in tackling the reproducibility crisis in cognitive neuroscience. *Cortex*, *113*, 357–359.

Makowski, D. (2020). Neurokit: A python toolbox for statistics and neurophysiological signal processing (eeg, eda, ecg, emg...). *GitHub*. Retrieved from https://github.com/neuropsychology/NeuroKit.py

Marchewka, A., Żurawski, Ł., Jednoróg, K., & Grabowska, A. (2014). The nencki affective picture system (naps): Introduction to a novel, standardized, wide-range, high-quality, realistic picture database. *Behavior Research Methods*, *46*(2), 596–610.

Miłkowski, M., Hensel, W. M., & Hohol, M. (2018). Replicability or reproducibility? On the replication crisis in computational neuroscience and sharing only relevant detail. *Journal of Computational Neuroscience*, *45*(3), 163–172.

Nosek, B. A., Cohoon, J., Kidwell, M., & Spies, J. R. (2015). Estimating the reproducibility of psychological science. *Science*, *349*(6251), aac4716.

Quintana, D., Alvares, G. A., & Heathers, J. (2016). Guidelines for reporting articles on psychiatry and heart rate variability (graph): Recommendations to advance research communication. *Translational Psychiatry*, *6*(5), e803–e803.

Roy, Y., Banville, H., Albuquerque, I., Gramfort, A., Falk, T. H., & Faubert, J. (2019). Deep learning-based electroencephalography analysis: A systematic review. *Journal* *of Neural Engineering*, *16*(5), 051001.

Schölzel, C. (2019). Nonlinear measures for dynamical systems (Version 0.5.2). Zenodo. https://doi.org/10.5281/zenodo.3814723

Topalidou, M., Leblois, A., Boraud, T., & Rougier, N. P. (2015). A long journey into reproducible computational neuroscience. *Frontiers in Computational Neuroscience*, *9*, 30.

Van Rossum, G., & Drake, F. L. (2009). *Python 3 reference manual*. Scotts Valley, CA: CreateSpace.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., … Contributors, S. 1. 0. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, *17*, 261–272. https://doi.org/https://doi.org/10.1038/s41592-019-0686-2

Yuehong, Y., Zeng, Y., Chen, X., & Fan, Y. (2016). The internet of things in healthcare: An overview. *Journal of Industrial Information Integration*, *1*, 3–13.