MDPI

Technical Note

System for automated Quality Control (SaQC) to enable traceable and reproducible data streams

Lennart Schmidt [1,2]*, David Schäfer [1,2], Juliane Geller [1,2], Peter Lünenschloss [1,2], Bert Palm [1,2], Corinna Rebmann [3], Karsten Rinke [4] and Jan Bumberger [1,2,5]

[1] Research Data Management - RDM, Helmholtz Centre for Environmental Research GmbH - UFZ, Permoserstraße 15, 04318 Leipzig, Germany
[2] Department of Monitoring and Exploration Technologies, Helmholtz Centre for Environmental Research GmbH - UFZ, Permoserstraße 15, 04318 Leipzig, Germany
[3] Department of Computational Hydrosystems, Helmholtz Centre for Environmental Research GmbH - UFZ, Permoserstraße 15, 04318 Leipzig, Germany
[4] Department of Lake Research, Helmholtz Centre for Environmental Research GmbH - UFZ, Brückstraße 3a, 39114 Magdeburg, Germany
[5] German Centre for Integrative Biodiversity Research (iDiv) Halle-Jena- Leipzig, Puschstraße 4, 04103 Leipzig, Germany
* Correspondence: lennart.schmidt@ufz.de

Abstract: With growing amounts of data from environmental sensor networks, automated quality control (QC) workflows are essential to provide high quality data in near real time for further use. While multiple domain-specific and -agnostic QC-tests are available, current software implementations are heterogeneous, inflexible and cumbersome as to their user interaction. We present the System for automated Quality Control (SaQC), a software framework for automated quality control that aims to alleviate these issues by being standardized yet extensible and by providing the user with built-in functions inside a a low-code configuration environment. Two use cases present basic and advanced quality control applications by using the features of SaQC. Also, we elaborate how SaQC can be used to make QC-workflows traceable and reproducible, thus promoting FAIR and AI-ready data streams of high quality.

Keywords: quality control, quality assurance, anomaly detection, sensor data, environmental data, FAIR, AI-ready, open source software

Citation: Schmidt, L.; Schäfer, D.; Geller, J.; Lünenschloss, P.; Palm, B.; Rebmann, C.; Rinke, K.; Bumberger, J. System for automated Quality Control (SaQC) to enable traceable and reproducible data streams. *Sensors* **2022**, *1*, 0. https://doi.org/

Received:
Accepted:
Published:

1. Introduction

Climate change, land use change and human interaction are putting the worlds' ecosystems under significant pressure. For a holistic quantification of these impacts in order to promote a sustainable resource use, interdisciplinary and large-scale observation networks are required as a means to leverage environmental sensor data across domain and terrain boundaries [1]. To that respect, large scale observatories (e.g. ILTER, eLTER, NEON, ICOS or TERENO) have been set-up and are continuously expanding, thus gaining significance in the environmental science community [see 2–6]. Adding to this, an increasing number of newly-developed, sophisticated environmental sensors are available to monitor additional environmental variables in science, business and other areas of human life. Consequently, the amount of environmental data that are being collected is steadily increasing, posing infrastructural challenges with respect to the data processing routines handling these data streams. Moreover, the requirements by academia regarding data quality and annotation are rising. Data providers are expected to adhere to the FAIR principles, i.e. to provide Findable, Accessible, Interoperable and Reusable data to the scientific community [7]. For a successful reuse and, thus, the exploitation of environmental observatory data's full potential for knowledge discovery, it is essential to establish data pipelines that address one major challenge related to sensor data: The process of quality control (QC), i.e. the

separation of erroneous from usable data based on expert knowledge [8]. Errors in environmental data can arise from multiple sources, among these are the suboptimal calibration or malfunctioning of sensors, technical failures during data recording and transmission or due to the influence of environmental conditions (e.g. sensor fouling, obstruction, freezing etc.) [9,10]. Ideally, the majority of these error sources are prevented by data workflow design (*quality assurance*). In application, however, post-processing steps to separate erroneous from usable data are commonly required (*quality control*) [11]. Manual QC, i.e. visual inspection of the data by domain experts is laborious, introduces subjective bias, requires expert-knowledge that is neither reproducible nor transferable and is simply unattainable for provisioning of large data volumes in real-time [12,13]. Fully automated QC-workflows, on the other hand, offer objectivity in the QC-process, reproducibility and efficient handling of large data volumes while allowing for practically unlimited test specifications that can be adapted over time [14]. Thus, automation of QC-workflows is the only future-proof option for data providers [15].

Figure 1 is a schematic representation of such an automated QC-workflow (left to right): *Raw data* flows from the sensor network to the database, runs through QC back into the database as *QC-ed data*, which can then be published as a data set and be used for further analysis. The QC-part (dashed blue box) generally consists of QC-tests that are run on raw data in order to identify erroneous datapoints (e.g. outlier detection algorithms) and assign a so-called flag, e.g. "Bad", "Suspicious" or "Good" to each datapoint as a measure of its quality. These are then sent back into the database, along with the QC-ed data. In the process, raw data is always kept to allow for a potential reprocessing [11].

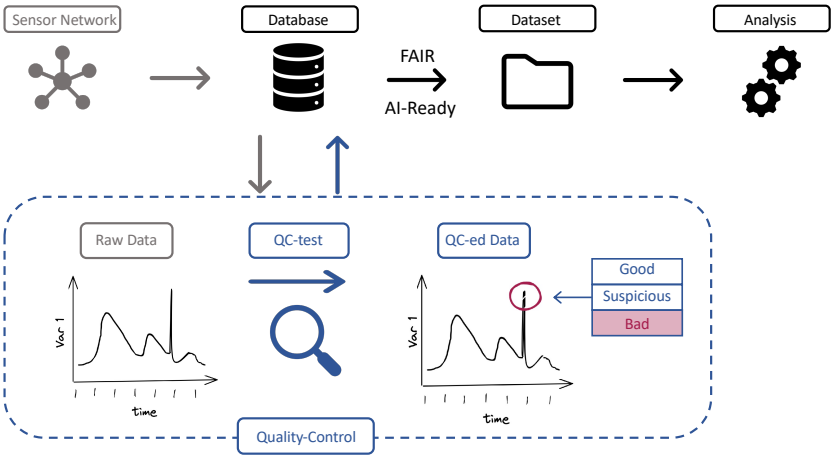

Figure 1. Illustration of a generalized data flow from sensor to analysis including automated quality control: *Raw data* flows from the sensors into the database and on to the QC-test. Here, a spike in the data (red circle) is identified as erroneous and flagged as "Bad". The resulting *QC-ed data*, along with a quality flag for each data point, is sent back into the database, from where it can be published as a dataset and later be used for analysis.

There are documented implementations of automated QC-pipelines in the environmental community, mostly from large and renown research infrastructures like NEON [16], GHCN [17], ICOS [18] or IAGOS [19]. Commonly, these infrastructures develop QA/QC protocols that are to be fulfilled by the members of the respective network. However, in many cases, the implementation of these protocols lies with the members who might be domain-experts with limited technical expertise. This poses a challenge and can result in disadvantages w.r.t to data quality and standardization, requiring additional harmo-

nization steps at the network level [15]. Similarly, small observation networks and single research institutes are challenged by the technical implementation of their QC-workflows, leading to the following shortcomings: In current practice, automated QC-workflows are use case specific and mostly hard-coded, making them inflexible to new functionalities and changing requirements such as altered data pre-processing steps or input data structures (CIT). Also, aside from large networks, there are no standardized flagging schemata, these are usually defined as best-suited for the respective use case, inhibiting interoperability. Additionally, current implementations do not ensure the entire workflow to be versioned and reproducible. This might lead to different versions of QC-ed datasets that are published over time without specific notice in the metadata or without the possibility to reproduce previous versions, if needed. Additionally, current implementations pose a challenge to the users, commonly domain experts: finding a robust choice and parametrization of quality tests that identifies as many suspicious values as possible without removing valid data is usually a time-consuming endeavor, thus requiring an intuitive and efficient user interface. Current implementations, however, typically require programming knowledge in order to parametrize or make changes to the QC-tests, which can not be presumed to be available.

In the environmental community, there are a few software tools available to assist the domain experts in setting up their QC-workflows. However, most of them are either domain specific or still exhibit one or more of the above shortcomings - see section 1.2 for a detailed screening of available tools. To our knowledge, there is no actively maintained, entirely open source software framework that facilitates the implementation of standardized QC-workflows for timeseries datasets from any kind of (environmental) sensor network, being flexible and user-friendly enough to account for the above shortcomings while accounting for traceability and reproducibility, two major aspects to fulfill FAIR criteria.

In this publication, we present the current state of (automated) quality control as well as the newly developed **System for Automated Quality Control (SaQC)** that aims at alleviating the issues mentioned above by providing a Python framework for setting up standardized QC-workflows for environmental timeseries data that is still flexible in all relevant interfaces to enable automated FAIR data pipelines in environmental science. In detail, it is:
Müssen ausgeschrieben und nachgeschärft werden

- **Universal:** applicable to any timeseries dataset from any domain
- **Flexible:** variable input and output data structures and flagging schemata
- **Extensible:** integration of custom pre- and post-processing methods and QC-tests
- **Traceable:** enables metadata enrichment along the pipeline
- **Reproducible:** enables versioning of QC-workflows
- **Accessible:** configuration by a low-code user interface
- **Open Source:** Available to anyone free-of-charge

Following a detailed software description in section 2, two use case examples in section 3 provide an in-depth explanation of how SaQC is employed in practical application. These use cases present some of its basic and advanced functionalities and the results of the QC are discussed. Following this, current limitations of the software as well as future development and research directions are presented.

1.1. Current state of automated quality control

Generally, the topic of automated quality control has been promoted primarlily by the meteorological community with large networks like GHCN, IAGOS etc. In 1971, the World Meteorological Organisation (WMO) published the first version of its "Guide on the Global Data-Processing System" which was updated continuously until 2003 [20], providing the community with extensive and detailed guidelines on a suite of QC- and consistency tests to use along with parametrization and flagging guidelines and has since been the baseline reference in the meteorological community and ist still referred to by recent WMO

applications, e.g. WMO2017 and WMO2018. These guidelines commonly address National Meteorological Networks that are requested to provide free and unrestricted access to climatological information [21]. National services have published similar guidelines for the operators of their observation networks, e.g. [22].

While a multitude of deterministic and statistical QC-tests are available, there is no standard set-up of QC-tests to be executed for an arbitrary environmental dataset. However, several (domain-specific) publications are available that present the entire QC-workflow of known observatories and thus serve as orientation for researchers aiming at setting up their own QC-workflow. For instance, [21] present the set-up of the Agroclimatic infromation Network of Andalusia, Spain (RIAA). In doing so, an intuitive introduction to the topic of QC in general as well as the most-commonly used methods and QC-tests is provided. Similarly, [17] present a documentation of the QA/QC-routine of the Global Historical Climatology network (GHCN) that incorporates 19 QC-tests and consistency-checks. [12] present an overview of the QA and QC-routines employed for common meteorological variables in the Oklahoma Mesonet metorological network, along with instructions.

Within the meteorological community, the Eddy-Covariance community has traditionally been faced with particularly challenging QC tasks [23]. Thus, it has undertaken considerable efforts to develop physics-based QC routines to assure the quality and potentially correct its data on the exchange of gas and water fluxes between the ecosystem and the atmosphere, see e.g. [24], [23] or [25]. Similarly, the task of QC-ing of solar radiance measurements has received much attention, see [26] for an overview. Next to meteorology, the oceanographic community has invested considerable efforts in setting up operational QC-systems - With strong representivity errors due to high spatial variance in the measurements along with mobile measurement location, making internal and external consistency checks indispensable (see e.g. [27,28]). Another task that is known to be challenging is the QC of in situ soil moisture data, [29] and [30] studied several spectrum-based approaches for spike detection, break detection and detection of constant values.

As the parametrization of QC-tests for a specific use-case is a challenging task (CITE), generic statistical methods to facilitate this process have been developed. Showcasing the set-up of the NEON-network, [31] present a statistical framework that allows for automated and standardized parameter calibration of multiple QC-tests. Based on the statistical distributions of the statistical moments/metrics of the data (mean, variance, min ,max), the appropriate test parameters can be derived. Building on this, [32] present the potential negative effects when assuming parametric distributions on the QC-process and derive an approach that is based on statistical validity intervals that are derived only from Min+Max of historical datasets. For the case of precipitation data, [33] illustrate that the the use of gamma distributions for outlier detection, especially when fitted conditionally on the conditions at neighboring stations. In addition to QC-tests at sensor level, cross-sensor/-variable and internal/external consistency checks are commonly implemented (CITE). [17] present and extensive framework on internal, temporal and spatial consistency checks. [34] provide a sensitivity analysis of two commonly-used approaches to check for spatial consistency (spatial regression and inverse distance weighting), even proving the applicability during extreme events.

Another important aspect of QC-workflows is how to summarize and communicate the QC-results to allow the user to assess the performance of the QC workflow and, thus, of overall quality of the dataset. [35] propose a modular and flexible framework for the NEON network to aggregate flagging information at the level of both sensor and whole data product as summary reports and figures. In order for a QC-workflow to assure good "quality of quality control", frequent auditing of QC-routines and its results is necessary [31], especially focussing on informing the user about potential errors that may remain.

QC-workflows should be monitored regarding spatial and temporal patterns to avoid introducing systematic bias or overflagging as a result of malparametrized tests [15]. [36] give a thorough overview of existing strategies and present a framework to identify the best parameter thresholds of a suite of QC-tests to reduce both false positve and false negative errors.

Generally, there is little standardization regarding implementation details of QC-workflows among networks. There are no network- or domain-agnostic guidelines as to which QC- and consistency checks have to be executed in which order and whether previously flagged data should be included. This can, however, lead to significant differences in the results and impede interoperability [31]. Similarly, there are no interoperable standards regarding flagging schemata or the monitoring of a QC-workflows' performance.

BRÜCKE BAUEN: Interoperability, Standards require standardized+flexible software

1.2. Related software

Generally, most existing frameworks are domain-specific and thus limited to domain-specific data sources, QC-tests or visualization capabilities.

The R-package AirSensor [37], built to visualize and analyze local air quality conditions, does include basic QC functions (range, outliers tests) as well as basic internal/external consistency routines. Also it provides timeseries processing and visualization functions and the possibility to include custom QC-tests. It is, however taylored to the specific application of low-cost air quality sensors in the community-based "PurpleAir" network, thus not built flexibly enough to be implemented for any kind of sensor network. Quite similar, [38] present (the development model of) the R-package eddy4R for processing and analyzing eddy-covariance data. Among other functionalities, it provides general QA-routines, qc-tests according to [31], tracking of quality information following [35] and multiple approaches for de-spiking. [39] provide the solar irradiance community with a free web-service of their "bias based" quality control method (BQC) to find low magnitude errors in data that are usually not captured by classical QC-methods. Users can simply load their own data files, perform the BQC-method and investigate the results. Within the Baseline Surface Radiation Network (BSRN), community guidelines exist on a suite of recommended QC-test along with parameters for several variables of surface radiation [40]. These are made available inside the BSRN-Toolbox that allows the user to visualize and quality control their surface radiation data [41] using the GUI of a software package that the user has to install on his/her own PC. FEHLT NOCH: pyhydroqc - ML-based QC of hydrological data as Python framework

(Rather) domain-agnostic:
[42] present ODM Tools Python, a Python-based tool that enables users to query and export, visualize, and edit hydrologic time series observations stored in an Observations Data Model (ODM) database. The main focus of the tool is to provide a GUI for manual QC, where provenance of all manual edits is kept as each step is recorded and saved in a Python-script to keep the process traceable and reproducible. To that end, there are similarities to SaQC. However, while ODM Tools aims at making the manual QC-process user-friendly and reproducible, SaQC tackles similar aims but mostly for the automated QC-part.
A tool that is domain-agnostic and freely-available is the Great Expectations Framework [**?**]. It is also availale as a Python library and provides the user with a multitude of so-called "expectations", i.e. tests to check the validity of a given batch of data or dataset. Due to an active user community, the range of available tests is large, ranging from simple checks to statistical methods. However, the aim of the framework is different from SaQC: Instead of identifying single erroneous values and assigning a flag, the aim is "dataset validation", i.e.

to validate the integrity of a dataset itself by checking its columns or entire data blocks for certain conditions. Consequently, it does not provide mechanism to assign and handle flag per single datapoint. Thus, its use for QC of arbitrary sensor networks as envisioned in this paper is limited, merely serving as a potential additional tool to apply prior to executiong SaQC. Its equivalent in the statistical programming language R is the assertR-package [43]. In a machine learning setting, where (erroneous) differences between training and test data can have severe influences, tensorflow data validation provides the user to check incoming data against a preset data schema, but also against data drift and skew [44]. [45] present a prototypical implementation of the so-called NRAQC-framework. While being limited in functionality it is innovative in such that it is implemented as a web-based microservice architecutre instead of as a local software package as most other tools. This potentially enables greater scalability across networked workstations while granting access to users from anywhere without requiring any local installation.

Probably, the most similar tool to SaQC is the GCE Data Toolbox [46] which provides a vast array of processing functionalitites to set up QC-workflows. It is written in Matlab and, similar to SaQC, it provides a GUI, a CLI as well as all functions available via a Matlab API. As SaQC, it provides a wide array of pre- and postprocessing functionalities, only some QC-tests but is flexible w.r.t to the possibiltiy of the integration of custom tests. It also includes mechanisms for flag-handling and, from within Matlab, QC-workflows can be designed, scheduled and triggered, keeping track of the the provenance, thus ensuring reproducibility. (NEEDS WORKOVER, HARD TO SAY WHY SAQC IS BETTER - David?).

Prorietary solutions do typically provide the means to perform manual and automatic QC, but only as part of a whole data management and alert system. Thus, the implementation inside custom infrastructures is hindered. Additionally, the data management systems are typically bound to a specific domain, e.g. Kisters WISKI or Aquarius for the water sector, and are thus not flexible enough to be adapted to data from other domains (e.g. ecology). Other proprietary software frameworks like Loggernet from Campbell Scientific are bount to specific hardware, raising the effort of integration of hadware from other manufacturers.

Table 1. Comparison of available software frameworks for automated QC

	Software Framework			
Requirement	SaQC	ODM-Tools	GCE Toolbox	Great Expectations
Flags at datapoint level	x	x	x	-
Suite of QC-tests	x	-		-
Universal	x	-	x	x
Flexible	x		x	x
Extensible	x	x	x	x
Traceable	x	x	x	
Reproducible	x	x	x	
Accessible	x	x	x	
Open Source	x	x	-	x
Reference				

2. Materials and Methods

2.1. Software description

SaQC is a free and Open Source Software framework available on the Python Package Index (PyPI)[1] and its Git repository[2]. It is both, a command line application and a Python framework that addresses the exploratory nature of quality control by offering

[1] https://pypi.org/project/saqc/
[2] https://git.ufz.de/rdm-software/SaQC

a continuously growing number of quality check routines through a flexible and simple configuration system. Below its user interface, SaQC is highly customizable and extensible thanks to its modular structure with well-defined interfaces.
Based on the generalized data flow from sensor to analysis as shown in figure 1, figure 2 is a detailed presentation of the QC-component of such a data flow when using SaQC. Below, this exemplary SaQC-based QC-workflow is explained in more detail and in chronological order, with its elements corresponding to the numbers 1) - 9) as indicated in the figure. This way, the components of SaQC and their contribution to achieve the *aims of the software* (ch. **??**) are explained in greater detail.

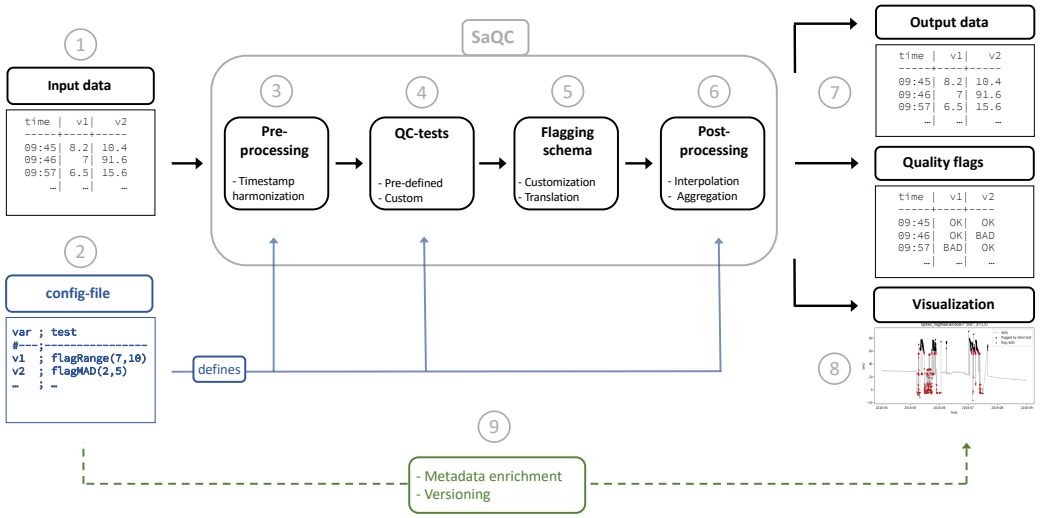

Figure 2. General QC-workflow using SaQC: Input data and a config-file are passed to SaQC which returns quality-controlled output data, along with quality flags and visualizations of the QC-results. The colors in the figure discriminate four categories: data flows (black), configuration (blue), metadata flows (green) and annotations (grey). The numbers correspond to the workflow elements as described in more detail in the text below.

(1) Input data can be supplied as .csv or .parquet-files when using SaQC as a command line application, in any tabular structure when using it in Python, directly. **(2)** Using a text-based configuration scheme that requires no knowledge of programming, users can then parametrize existing processing routines as well as QC-tests inside a *config-file*. Also, users can write custom routines and tests using a simplified, Python-like syntax. A detailed presentation of how to set up a config-file can be found in section 2.3 **(3)** The config-file also defines the pre-processing routines that might have to be executed prior to the QC-test. Both pre-processing- and QC-routines are tightly coupled as most QC-tests require data pre-processing such as timestamp harmonization. **(4)** The selection of pre-defined QC-tests aims at being applicable across domains and ranges from standard tests such as constant value or spike detection to more advanced methods like pattern recognition and multivariate tests. More information on both processing and QC-test functionalities can be found in section 2.2. Users can also implement entirely new QC-test functionalities in the config-file or in the core Python code. **(5)** Depending on the result of the tests, each datapoint gets assigned a flag following a user-defined flagging scheme that defines the number and hierarchy of the flags, e.g. *Good*, *Suspicious* and *Bad*. SaQC allows entirely user-defined flagging schemes and provides functionality to translate these between each other. **(6)** If desired, post-processing routines like interpolation or aggregation can be applied on the newly quality-controlled data. **(7)** By default, the processed data, along with the assigned flags, is then returned as .csv or .parquet-files (command line application) or any other user-defined format (Python). **(8)** Adding to the file output, SaQC provides various visualization functionalities that allow the user to investigate the effect of different test

parametrizations on the QC-results. Examples of the available visualizations can be found in the use cases in section 3. **(9)** Across the whole workflow, SaQC ensures traceability and reproducibility by metadata enrichment and versioning of pipelines (see section 2.5 for details).

For documentation and tutorials, readers can refer to the online documentation of SaQC[3] which also contains the Getting Started Guide[4].

2.2. Function overview

SaQC ships with extensive processing capabilities as well as various domain-independent QC-tests, giving a total of 29 functions that are continuously extended. Figure 3 aims at giving an intuitive representation of how these functions are further categorized into **1) processing** which serves as either the preliminary (pre-processing) or finalizing (post-processing) step to both **2) basic QC-tests** and **3) advanced QC-tests**. In SaQC, each of these three categories is populated by multiple functional groups, which in turn encompass a number (#) of functions that serve the same objective. Table 2 provides a list and description of these functional groups, objectives and the respective number of functions. For an in-depth description of each of the functions, readers are referred to the software documentation.

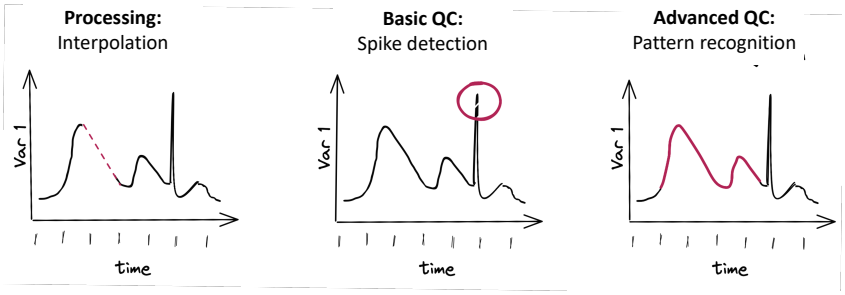

Figure 3. Schematic visualization the three usage categories of exemplary 1) processing, 2) basic QC-test, and 3) advanced QC-test (from left to right).

Table 2. The three usage categories as included in SaQC. Each category consists of functional groups which encompass a certain number (#) of functions that serve the same objective.

Group	Objective
Processing	*Processing steps prior to (pre-processing) or after QC-testing (post-processing)*
Resampling	Aligning data to equi-distant timestamps by shifting, aggregation or interpolation
Interpolation	Gap-filling using both parametric and non-parametric interpolation methods
Basic QC	*Low algorithmic complexity and parametrization effort*
Constants	Deterministic and variance-based detection of undesired stationary behaviour
Breaks	Detection of missing/isolated values or jumps in the data
Outliers	Detection of outliers and spikes, both deterministic and statistical methods
Manual flagging	Integration of precedent manual QC by experts from additional files
Advanced QC	*Higher algorithmic complexity and parametrization effort*
Changepoints	Detection of points of undesired system state transitions
Drift correction	Detection and correction of sensor drift based on deviation from reference system state
Pattern recognition	Detection of undesired patterns in the data based on Dynamic time Warping and Wavelets
Modelling	Identification of anomalies using curve-fits, filters or multivariate scoring approaches
Custom functions	Combination of existing/integration of custom QC-tests inside config-file or source code

Furthermore, the following facts should be kept in mind with respect to the use of the functions of SaQC:

[3] https://rdm-software.pages.ufz.de/saqc/index.html
[4] https://rdm-software.pages.ufz.de/saqc/getting_started/TutorialCLI.html

- custom functions can be implemented either right in the config-file (simplified syntax, logical operators) or in the source code (Python)
- if in Python: The modular architecture of SaQC is designed to accommodate custom processing and QC functions of arbitrary complexity via predefined interfaces
- processing functions can be run independently, i.e. without antecedent or subsequent QC, if desired
- while the functions are run on the data and its timestamps, inherent rules for handling of flags during these operations are adhered to, e.g. when resampling
- similarly, when first executing pre-processing, e.g. resampling, and then assigning flags, these flags can be re-projected back to the original source data

2.3. Configuring SaQC for use

As **command line application**, SaQC is controlled by a config-file (.csv) listing the variables of the dataset as well as both the processing routines and QC-tests along with respective parameters that are to be executed. The content of such a configuration could look like this:

```
varname     ; test
------------;-----------------------------------
SM2         ; shift(freq="15Min")
SM2         ; flagRange(min=10, max=60)
SM2         ; flagMAD(window="30d", z=3.5)
SM2         ; interpolateInvalid(method='linear')
```

Here, a timestamp shift to a regular interval of 15min is performed during pre-processing. This is followed by a range test that flags all values outside the range of [10,60] and a spike test. Lastly, data gaps are filled using linear interpolation. In fact, this is the config-file as used for use case 1 in section 3.1.
As soon as the basic inputs, a dataset and the config-file are prepared, running SaQC is as simple as:

```
saqc --config path_to_configuration.csv --data path_to_data.csv
```

where `path_to_configuration.csv` is the spaceholder for the configuration file described above, and `path_to_data.csv` acts as the spaceholder for the input dataset.
The same set-up can be achieved directly in a Python script by using SaQC's **Python API**, which is explained in more detail in the software documentation.

2.4. Flag Handling inside SaQC

- User-defined flag handling inside functions/during Aggregation
- Flagging History
- Coding of flags as numeric values
- Flexibility

2.5. Enabling traceable and reproducible workflows

The FAIR-principles itself do not explicitly include quality control as a prerequisite [47]: "The FAIR guiding principles request that optimal care is taken to enable users to determine the 'usefulness' (for their purpose) of the data and other research objects they find, which includes rich, machine readable provenance" [7]. In other words, simply noting the absence of quality control in the metadata of a dataset would suffice to make a dataset FAIR. However, data does only actually become re-usable by going beyond that, i.e. by establishing traceable QC-workflows. This, in turn, requires more attention if FAIR guidelines are still to be met: R1.2., i.e. part of the definition of Reusability, states that "datasets are to be associated with detailed provenance [47]". In the case of QC-workflows using SaQC, this is achieved by extensive metadata enrichment along the QA/QC-pipeline: Processing steps, QC-tests and their respective parametrization are accessible for every single quality flag produced by SaQC (*traceability*).
The ultimate goal of such successful metadata provenance is that "[datasets] can be replicated and/or combined in different settings" [7]. SaQC supports versioning of the entire

pipeline including all parametrizations through version control systems (e.g. Git). As such, every single quality flag is not only traceable, but can be replicated if necessary (*reproducibility*).

The following figure 4 is a variation of the exemplary SaQC-workflow as previously depicted in figure 2 but with a focus on the metadata component (green colour). Figure 4 illustrates in detail how metadata enrichment and versioning can be achieved in SaQC-based workflows: The input data comes with source metadata, e.g. sensor location, ID or data author. As it runs through the SaQC-workflow, and as such through the QC-tests as defined in the config-file, metadata is being collected and merged with the source metadata. For instance, in the case of a datapoint identified as a spike (red circle in fig. 4), this includes the quality flag as well as information on the flagging schema it refers to. Also, the QC-test that assigned the flag (here: spike-test) is written, which becomes relevant when multiple QC-tests are executed on the same variable. To account for varying implementations across software versions, the version of SaQC is noted as well. In order to keep track of the version of the entire workflow, the config-file is versioned via Git and the respective Git Commit Hash can be written into the metadata. With this, a user receiving any dataset that was QC-ed using SaQC is able to **1) trace** and **2) reproduce** every single step of the pipeline - Thus meeting the *aims of the software* as stated in section **??**.

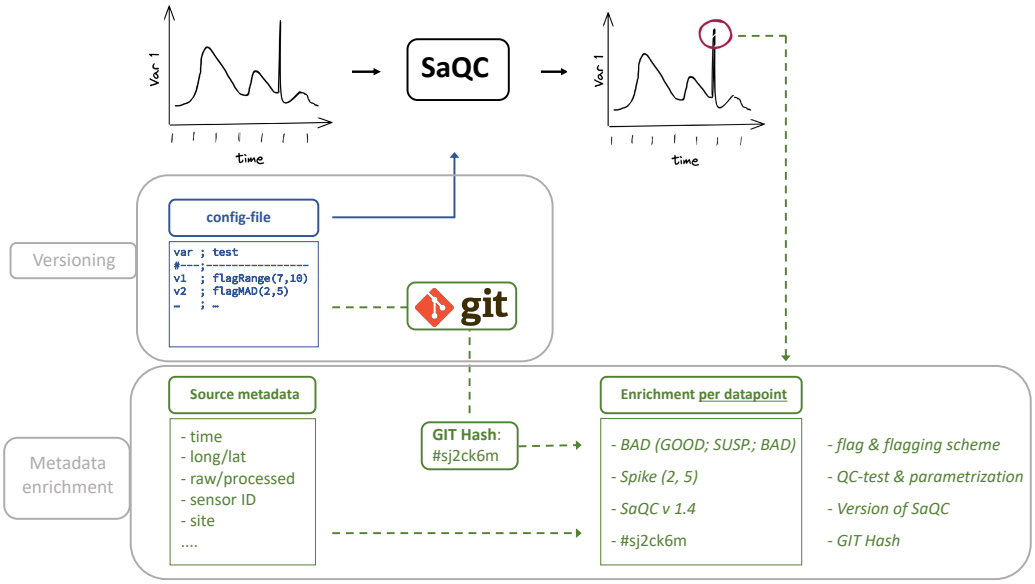

Figure 4. Schematic illustration of metadata enrichment (traceability) and versioning (reproducibility) when using SaQC. The red circle marks a datapoint identified as a spike by a QC-test. The other colors discriminate four categories: Data flows (black), configuration (blue), metadata flows (green) and annotations (grey)

3. Results and Discussion

This section presents two use cases of SaQC in actual applications at the Helmholtz-Centre for Environmental Research (UFZ), more specifically at two observatories that are part of the TERENO Bode Hydrological Observatory [48] which is located in the Harz region in central Germany. The first use case at the forest ecosystem observatory Hohes Holz [49] employs some of the functionalities listed as *Processing* and *Basic QC-test*. The second one at the hydrological Rappbode-observatory [50] presents the use of *Processing* functionalities and both *Basic* and *Advanced QC-tests*. Both use cases are presented in more detail in our collection of Cookbooks in the online documentation[5], along with the necessary data and config-files to reproduce all results as presented here. For both use cases, the timeseries

[5] https://rdm-software.pages.ufz.de/saqc/index.html

snippets that are shown were selected to be intuitive and illustrative with the aim of showcasing the functions of SaQC. The authors do not intend to evaluate the performance of the developed QC-workflows in detail, therefore the results are only discussed briefly. Instead, the focus is kept on the software itself by concluding the section with a discussion of current limitations of the software and future development and research directions.

3.1. Use case 1: Basic quality control of soil moisture data

At the Hohes Holz observatory (Central Germany), a patch of mixed beech forest, a huge set of environmental variables is measured continuously to improve the scientific understanding of carbon and water fluxes between the ecosystem and the atmosphere under the influence of environmental changes (see fig. 5). Various meteorological, hydrological and ecological variables are observed at high spatial and temporal resolution, among these distributed soil moisture content measurements using about 180 single sensors (sensor model: SMT100, TRUEBNER GmbH, Neustadt, Germany).

Figure 5. Left: Birds-eye view of the forest ecosystem observatory Hohes Holz. Right: Measurement set-up of multiple soil moisture sensors at different soil depths as employed in use case 1. Image source: UFZ.

In this use case, quality control of soil moisture data is performed in four steps. The dataset contains the variables *time*, *battery* (battery voltage of the data acquisition platine) and soil moisture *SM2* in vol.% at one sensor. The following description handles line by line of the actual config-file from section 2.3, also included in appendix A.1, along with illustrative figures. An in-depth presentation of the workflow, including all figures, algorithmic details and references, can be found in the respective Cookbook [6]. Figure 6 depicts the raw data of sensor *SM2* before quality control, exhibiting unrealistic values beyond a plausible value range.

[6] https://rdm-software.pages.ufz.de/saqc/getting_started/TutorialCLI.html

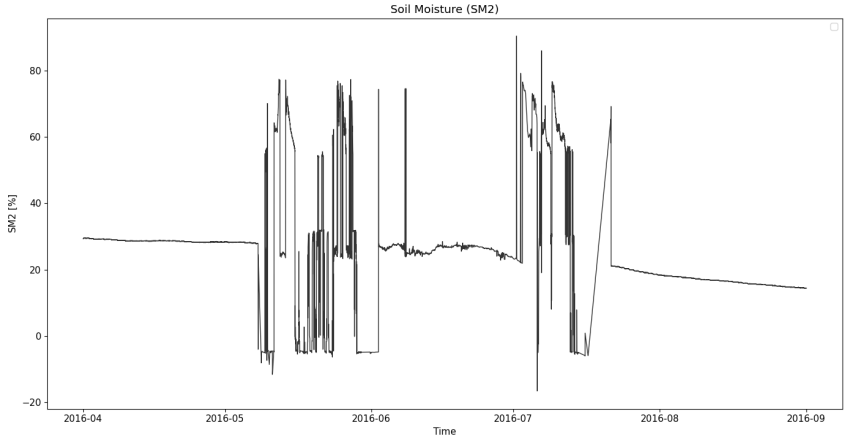

Figure 6. Raw soil moisture data of sensor *SM2* from a distributed soil moisture sensor network at the research site 'Hohes Holz'.

1. Timestamp harmonization via resampling: As a result of irregular data acquisition frequency of the wireless sensor network, the timestamps of the measurements are not equidistant. See table 3 for an exemplary screenshot of the data and the irregular timestamps that are supposed to be regular at 15min-interval.

Table 3. Exemplary screenshot of the raw soil moisture data with irregular timestamps.

time	battery	SM2
2016-04-01 00:04:48	3573	32.6
2016-04-01 00:20:42	3572	32.7
2016-04-01 00:36:05	3572	32.6
...		

To ensure regular time-steps for further processing, the datapoints are shifted to the closest timestamp of a 15min-grid, thereby also eliminating possible duplicates. The corresponding line in the config-file is this one:

```
varname ; test
#-------;----------------------------------
SM2     ; shift(freq="15Min")
```

And the resulting output data is displayed in table 4:

Table 4. Exemplary screenshot of the pre-processed soil moisture data after step 1, now with regular timestamps.

time	battery	SM2
2016-04-01 00:05:00	3573	32.6
2016-04-01 00:20:00	3572	32.7
2016-04-01 00:35:00	3572	32.6
...		

2. Range test: Next, a range test to exclude values below/above a physically meaningful threshold is performed (here: [10, 60], vol.% soil moisture).

```
SM2     ; flagRange(min=10, max=60)
```

3. MAD outlier test: Following this, a Modified Z-Score (MAD) outlier detection test, as proposed by [51], is performed to detect and flag spikes in the data. The resulting flags of steps 2 and 3 are depicted in figure 7.

```
SM2     ; flagMAD(window="30d", z=3.5)
```

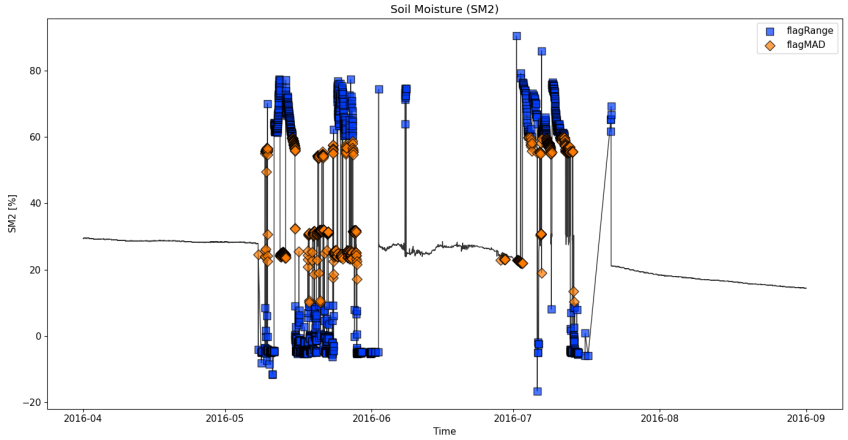

Figure 7. Quality flags of the variable *SM2* as a result of steps 2 and 3, i.e. a range-test (flagRange, blue markers) and a MAD outlier test (flagMAD, orange markers)

4. Linear interpolation of missing values As a final post-processing step, the missing values that result from identifying erroneous values during the preceding flagging are filled using linear interpolation:

```
SM2     ; interpolateInvalid(method='linear')
```

The final, clean results, i.e. a comparison of the timeseries of the variable *SM2* before and after the QC-process is displayed in figure 8. The major error pattern, fluctuations beyond a plausible value range, has been successfully corrected for. Given the amount of erroneous data points that were identified (fig. 7) and excluded, the resulting data gaps make up a considerable portion of the whole timeseries snippet. Therefore, the application of linear interpolation in step 4 is questionable and would have to be communicated to the end user in each datapoints' metadata as explained in section 2.5. Here, it is only applied with the aim of showcasing the functions of SaQC.

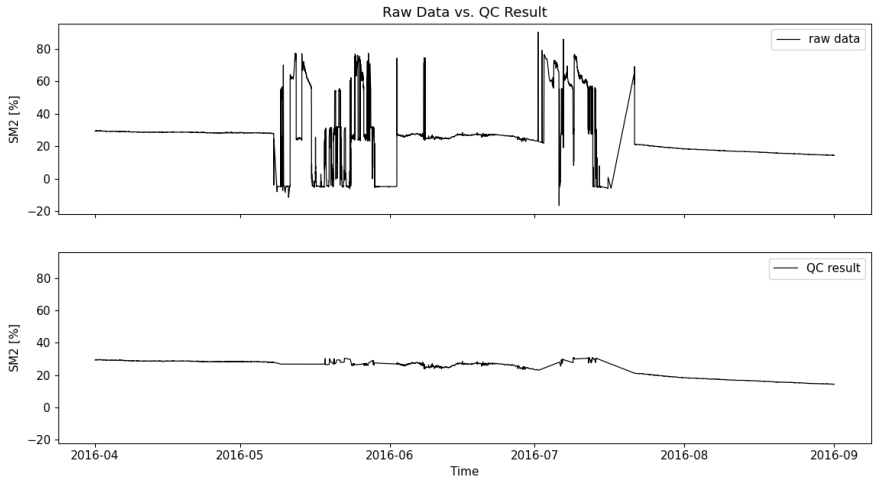

Figure 8. Timeseries of the variable *SM2* before (top) and after (bottom) the QC-process.

3.2. Use case 2: Advanced quality control of surface water body data

At the Rappbode observatory, Germany's largest drinking water reservoir with a maximum storage of about 100 million m^3, a multitude of hydrological variables are recorded to investigate the inflow dynamics of nutrients and dissolved organic carbon (DOC) from the surrounding catchment (see fig. 9). For more details on the observatory, please see [50].

Figure 9. Left: Birds-eye view of the Rappbode reservoir, the site of the corresponding hydrological observatory. Right: Measurement buoy carrying sensors as employed in use case 2. Image source: UFZ.

More specifically, the variables *water level*, *water temperature*, and *sac254* (spectral absorption coefficient at 254 nm, used as a proxy for DOC) are measured in all four major inflows using the sensors as presented in table 5. These measurements are highly sensitive and require regular maintenance as well as sophisticated quality control which is performed in five steps. In this section, the first four steps are elaborated for one of the variables, *sac254*, only. In the last step, all three variables are used as covariates for multivariate flagging. This description handles line by line of the actual config-file needed for the final results (see appendix A.2), along with illustrative figures. An in-depth presentation of the entire workflow, including all figures, algorithmic details and references, can be found in the respective Cookbook [7].

Table 5. Model and manufacturer of the sensors used to measure the variables of use case 2.

Variable	Unit	Sensor model
water level	m	Analog Submersible Level Sensor, STS AG
Water temperature	°C	EXO2 Multiparameter Probe, Xylem Inc
sac254	m^{-1}	OPUS UV Spectral Sensor, TriOS, Germany GmbH

Figure 10 depicts the raw timeseries of *sac254*, i.e. before quality control. It exhibits undesired outliers and cyclical drops in its data values that are physically implausible.

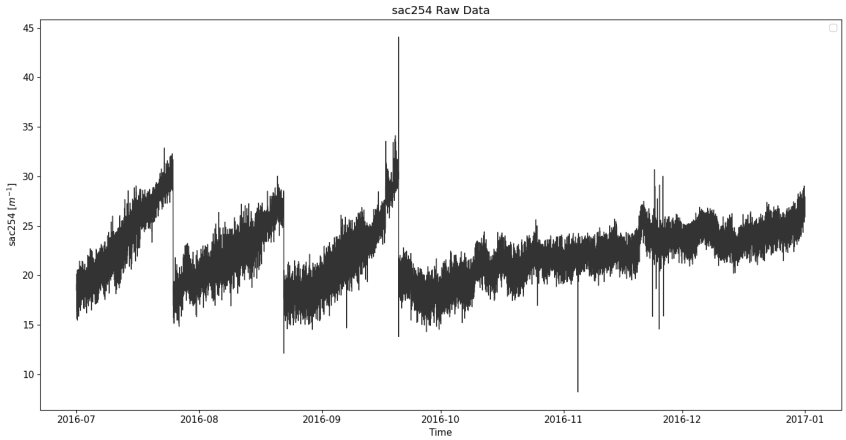

Figure 10. Raw data of the variable *sac254* before quality control.

[7] https://rdm-software.pages.ufz.de/saqc/cook_books/MultivariateFlagging.html

This advanced use case touches on the potential complexity of QC-workflows with multiple routines building on top of each other. Also, it shows some of the functionalities of SaQC that are indispensable in the practical application - e.g. the projection of flags in step 5.

1. Flagging of maintenance intervals: As a first step, data recorded during known maintenance operations is flagged as these operations affect the measurements. The respective information is encoded in a separate variable *maint* and used inside the manual flagging function.

```
varname ; test
#-------;----------------------------------------------
sac254  ; flagManual(mdata='maint')
```

2. Range test: Next, basic QC in form of a range test to exclude values below/above a physically meaningful threshold is performed (here: [0,70], m^{-1}). This is necessary to avoid incorrect interpolation of data values in the next step. The resulting flags of step 1 and 2 are visualized in figure 11

```
sac254  ; flagRange(min=0, max=70)
```

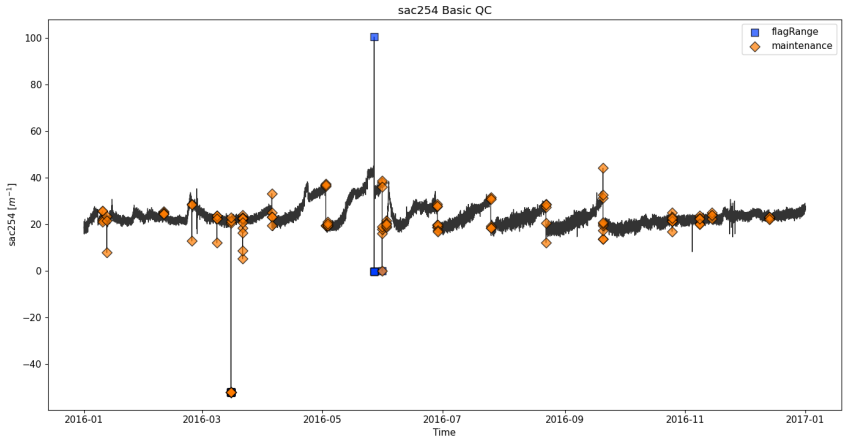

Figure 11. Flags for of the variable *sac254* as a result of steps 1 and 2, i.e. based on known maintenance intervals (flagManual, orange markers) and a range test (flagRange, blue markers)

3. Timestamp harmonization via resampling: As the sensors do not necessarily measure at the same interval or exact same time, the timestamps of all variables are resampled to an equi-distant, synchronous grid of 15min-intervals. This is needed for subsequent multivariate QC-tests. As the data values themselves have to be adapted accordingly, linear interpolation is performed on the data values where necessary, not touching on actual missing values. The interpolation time step has to be adjusted to a value that is fitting to the original temporal resolution and that does not lead to larger changes in the dynamics.

```
sac254  ; linear(freq='15min')
```

4. Drift correction: The variable *sac254* exhibits a short-term drift: Due to biofilm growth and dirt accumulation on the sensor lenses, the reference, or base value of the measurements increases over time as illustrated in figure 12. Based on expert knowledge, these increases are assumed to follow an exponential growth and are thus corrected for by subtraction of an exponential term that was parameterized manually in advance. Whenever maintenance, i.e. sensor cleaning or replacement is performed, the drift correction is reset to its new starting point.

```
sac254  ; correctDrift(target='sac254_corr', maintenance_field='maint',
                model=expDriftModel)
```

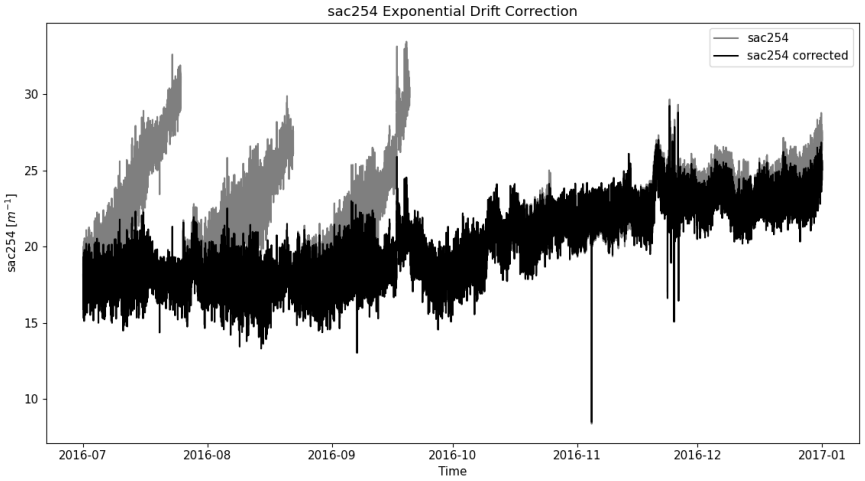

Figure 12. Illustration of the drift correction of the variable *sac254*. The grey line depicts the original data, the black line represents corrected data, both at 15min-resolution.

5. Multivariate spike detection using kNN and STRAY:

Next, not only the information of *sac254*, but of all three variables (*water level, water temperature* and *sac254*) is used to identify multivariate outliers. The methodology is based on the *oddwater*-algorithm by [52]. The aim is to employ the well-established unsupervised clustering algorithm k-Nearest Neighbors (kNN). This requires the variables to be normalized in a first step using z-score transformation. Next, the kNN-algorithm is used to assign a multivariate outlier score (new variable: *kNN_scores*) to each timestep. Figure 13 (top) provides a visual intuition of how, during this dimensionality reduction, multivariate outliers become detectable when transformed into one variable. These *kNN_scores* are then processed further using the STRAY-algorithm [53] to automatically define a threshold above which datapoints of the *kNN-scores* are considered to be actual outliers and can be flagged as such. As a last step, these flags are projected from the *kNN_scores*-variable back onto the actual data variables. Figure 13 (bottom) depicts the results after this projection is performed for *sac254*.

```
# 5: Multivariate spike detection using kNN and STRAY
# Normalization of variables for kNN
level       ; transform(target='level_norm', func='zScore', freq='20D')
water_temp  ; transform(target='water_temp_norm', func='zScore', freq='20D')
sac254_corr ; transform(target='sac254_norm', func='zScore', freq='20D')

# Flagging by kNN and STRAY
level_norm,water_temp_norm,
sac254_norm ; assignKNNScore(target='kNN_scores', freq='20D')
kNN_scores  ; flagByStray(freq='20D')

# Project results of STRAY-algorithm onto variables
level       ; transferFlags(field=['kNN_scores'])
water_temp  ; transferFlags(field=['kNN_scores'])
sac254_corr ; transferFlags(field=['kNN_scores'])
```

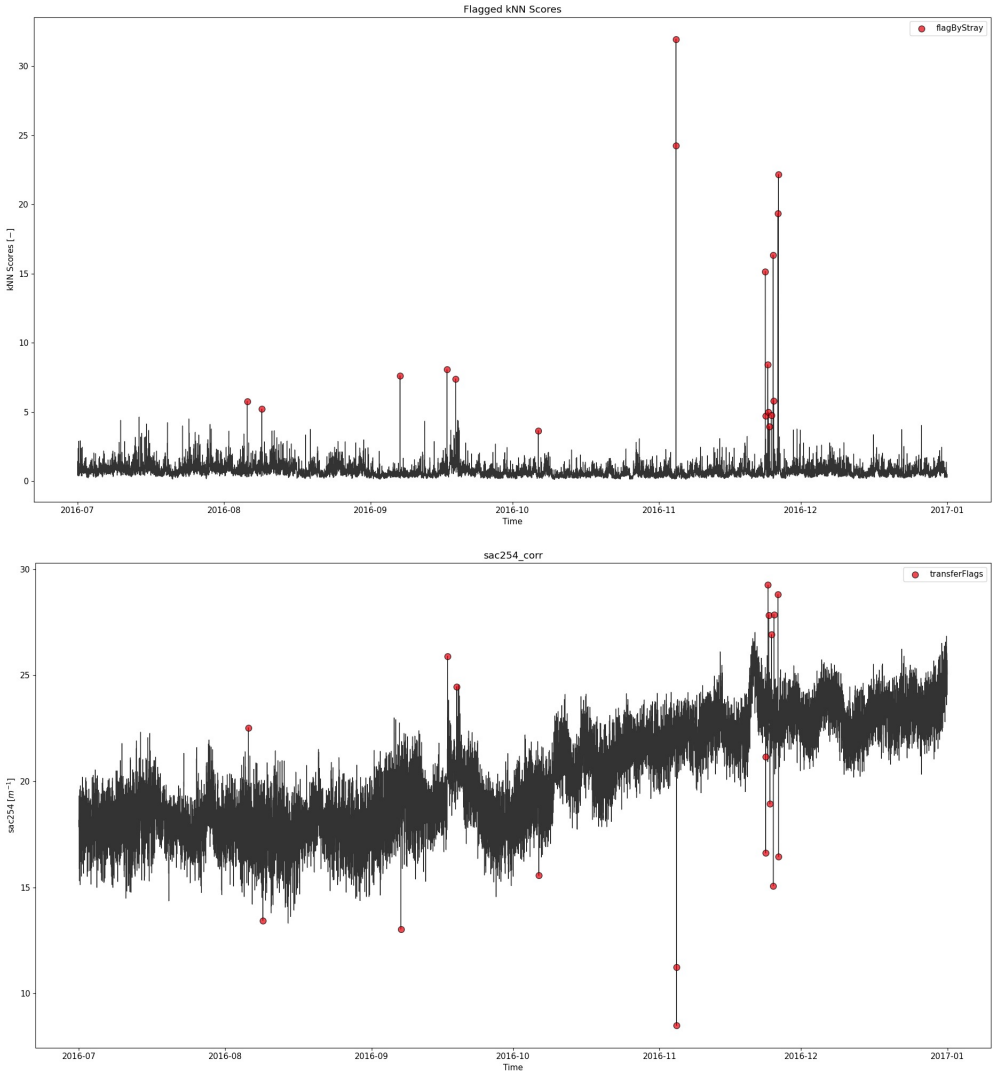

Figure 13. Top: Multivariate outliers (green markers) in the *kNN-scores* as identified by the STRAY-algorithm, i.e. based on all three variables (*water level*, *water temperature* and *sac254*). Bottom: The corresponding outliers projected onto and visualized for the variable *sac254*, only. Note that, for illustration, this plots depicts a shorter time snippet than the previous figures, i.e. is zoomed in.

The final, clean results, i.e. a comparison of the timeseries of the variable *sac254* before and after the QC-process, are presented in figure 14. In general, all major error patterns (major outliers, cyclical drops, drift) have been corrected for. A few spikes remain that should be analyzed in more detail by the domain expert and that could possibly be covered by the parametrization of one of the spike tests included in SaQC. However, it is to be noted that the figures shown here only depict snippets of the whole timeseries at one sensor while the observatory consists of numerous sensors that exhibit slightly varying error characteristics. As finding the right parametrization for a time snippet of one single sensor already requires substantial efforts by the domain experts, finding a robust parametrization for the whole network is always a trade-off between local performance (timeseries snippets of one sensor) and global performance (whole network).

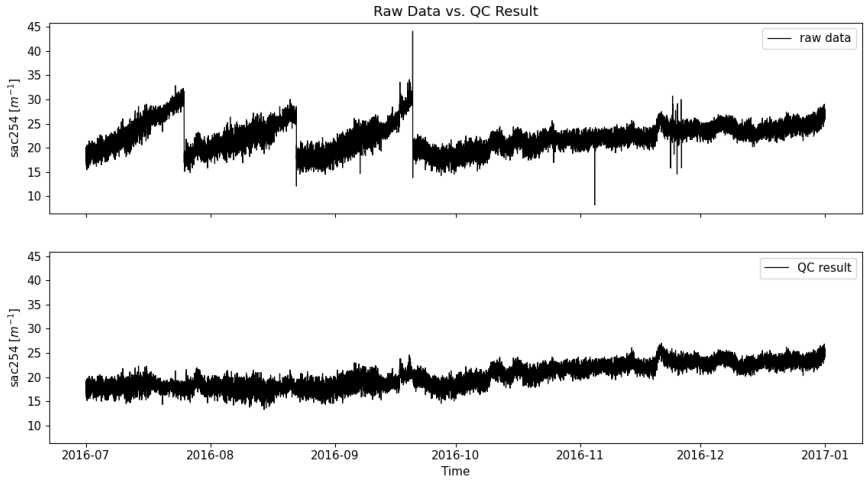

Figure 14. Timeseries of the variable *sac254* before (top) and after (bottom) the QC-process.

3.3. Limitations of the software

The previous sections illustrate how the main aims of SaQC are achieved, i.e., how domain experts are enabled to set up flexible and reproducible QC workflows for any timeseries dataset. Also, it is shown which functions are available in SaQC and how these can be used to successfully perform the actual QC on sensor data in practical application. In doing so, we show how many of the aforementioned challenges in establishing QC workflows are alleviated. Nonetheless, it is important to know about certain limitations of the software as listed in the following paragraph.

One challenge that still persists for users with limited IT skills is that, currently, it is required to install the programming language Python (without being able to use it). Similarly, configuring the QC-tests via a .csv config-file might be unfamiliar to users that are used to employ software solely using a Graphical User Interface (GUI). While the current implementation does not lessen the functionalities nor increase the complexity, the development of a web-based GUI is currently underway to make SaQC more accessible to this user group. Regarding input file types, these are currently limited to .csv and .parquet if SaQC is used via the command line interface (CLI). If used via the Python API, any file format or database API can be implemented by the user. Concerning traceability and reproducibility,, 1) versioning and 2) metadata enrichment are optional functionalities. If desired, the user is required to 1) host the QC-pipeline on the code versioning platform Git and 2) be able to store additional metadata (SaQC version, flagging schema, QC-test that was executed etc.) for each datapoint in the respective data repository. Depending on the system in use, the latter can potentially result in computational overload if the configuration of the pipeline is changed and all historical datapoints are to be re-processed. SaQC can not provide domain-specific QC-tests for all (environmental) domains and variables. However, the majority of QC-tests are domain-agnostic and, as mentioned, custom tests can be integrated in various ways at any level of complexity, ranging from simple logical rules written directly in the config-file to any custom Python function integrated into the core code via well-defined interfaces. Lastly and naturally, setting up a QC-workflow still requires considerable parametrization effort, but SaQC supports the domain expert by making it as easy as possible while rendering labor-intensive manual quality control superfluous.

3.4. Future development and research directions

Future development of SaQC is envisioned in multiple ways: Firstly, it will be implemented in further real-world environmental monitoring workflows, both at the authors' institutions and beyond. While doing so, the existing set of QC-tests will be continuously extended by domain-specific and agnostic tests. Also, the integration of Deep Learning

algorithms for anomaly detection will be evaluated as these have been shown to achieve high classification accuracy by exploiting spatio-temporal patterns in the data of the entire sensor network [54]. This could greatly reduce the existing effort to parametrize QC-tests. On the other hand, the training of anomaly detection algorithms involves challenges like handling of missing data, class imbalances and low interpretability [55]. On the user side, a GUI will be developed to expand the potential user group. Furthermore, SaQC will be disseminated in the scientific community to enable more domain scientists to use it in their specific application. With a growing user community, the authors envision an open source community that contributes specific QC-tests along with suitable parametrization schemes for future re-use.

Any automated in situ monitoring requires a sound quality control and, given the high data delivery rates of modern sensors, this quality control needs to be automatized [56,57]. On the other hand, more is needed to bridge the gap between sensors and decision makers as the emerging data streams need not be cleaned up from unrealistic or biased values but also need to be validated and ground-truthed. Therefore, the required workflows must include the comparison of quality-controlled sensor data with external or additional data. For example, the soil moisture data of use case 1 could be validated using field soil samples or remote-sensing products like SMAP. The measurements of dissolved organic carbon in surface waters (*sac254* in use case 2) could be compared against wet-chemical analysis of water samples in the lab in order to quantify accuracy and uncertainties of the in-situ sensor data. Thus, we aim at linking online sensor data with external data sources to obtain such a data validation. This is a crucial step in mediating the inclusion of online sensor data into real-world decision making.

4. Conclusions

In order to derive a system understanding of the drivers of ongoing environmental changes, observation networks that combine a multitude of interdisciplinary observatories across a large geographical extent are needed. To ensure FAIR and AI-ready data flows, automated, traceable and reproducible quality control (QC) workflows are required for these infrastructures. While both domain-specific and agnostic QC-tests are available, present implementations are use case specific, thus inflexible, and cumbersome for the user, mostly domain experts with limited IT-expertise. This publication presents SaQC, an open source framework for quality control of timeseries data that aims at alleviating the aforementioned challenges when setting up automated QC-workflows by providing the following features: It is universal as to its application domain and flexible with regard to both its input/output data structures as well as flagging schemata. While already providing a comprehensive set of methods, it is extensible in its pre- and post-processing as well as QC-testing functions. Adding to this, it provides functionality to make QC-workflows traceable and reproducible, thus promoting FAIR and AI-ready workflows. Also, SaQC is designed to be accessible, particularly to users without programming knowledge as is often the case in practical application. By means of a detailed software description and the presentation of two real-world use cases, we showcase how SaQC can be used to process and perform quality control on one's own data both at basic and at advanced level. In addition, we state current limitations of the software. Thus, the scientific community is supplied with (instructions on) a software framework to automatically deliver environmental data that is clean, up-to-date, potentially large in volume and reusable - A principal prerequisite for successful, data-driven and sustainable science to address urgent environmental challenges.

Author Contributions: conceptualization, L.S., J.G. and J.B.; methodology and software, D.S., B.P., P.L., L.S. and J.G., validation, J.B. and D.S.; formal analysis, P.L. and B.P., investigation, C.R. and K.R.;; resources, B.P.; data curation, C.R., K.R. and P.L.; writing—original draft preparation, L.S. and J.G.; writing—review and editing, all; visualization, P.L. and L.S.; supervision, J.B.; project administration, D.S.; funding acquisition, J.B., C.R., K.R. All authors have read and agreed to the published version of the manuscript.

Funding: K.R. received funding from the H2020-MSCA-ITN-2020 training programe IventWater (Grant No 956623).

Data Availability Statement: The System for Automated Quality Control (SaQC) is available both on the Python Package Index (PyPI)[8] and via its Git repository[9]. It is open source under the GNU General Public License, version 3. The computations in this publication have been executed with SaQC, version 2.0. Multiple tutorials in form of so-called Cookbooks that support the user to get started can be found in the SaQC online documentation [10]. Among these, there are two Cookbooks that present the two use cases from section 3 in detail, along with the necessary data and config-files to reproduce the results.

Acknowledgments: The Rappbode observatory was financially supported by TERENO (TERrestrial ENvironmental Observatories, funded by the Helmholtz Association and the Federal Ministry of Education and Research, BMBF) and the Talsperrenbetrieb Sachsen-Anhalt.

Conflicts of Interest: The authors declare no conflict of interest.

[8] https://pypi.org/project/saqc/
[9] https://git.ufz.de/rdm-software/SaQC
[10] https://rdm-software.pages.ufz.de/saqc/index.html

Appendix A.

Appendix A.1. Config-file use case 1

This config-file includes all steps to perform QC as presented in use case 1 in section 3.1. For a more detailed description, the reader is referred to the respective Cookbook in the SaQC online documentation.

```
varname     ; test
#-----------;-----------------------------------
SM2         ; shift(freq="15Min")
SM2         ; flagRange(min=10, max=60)
SM2         ; flagMAD(window="30d", z=3.5)
SM2         ; interpolateInvalid(method='linear')
```

Appendix A.2. Config-file use case 2

Please note that, while the description of use case 2 in section 3.2 focuses on the variable *sac254* for readability, this config-file lists the required steps for all three variables (*water level*, *water temperature* and *sac254*). For a more detailed description, the reader is referred to the respective Cookbook in the SaQC online documentation.

```
varname     ; test
#-----------;-----------------------------------
# 1: Flagging of maintenance intervals
sac254      ; flagManual(mdata='maint')

# 2: Basic QC: Range-tests
level       ; flagRange(min=0)
water_temp  ; flagRange(min=-.5)
sac254      ; flagRange(min=0, max=70)

# 3: Pre-processing: Resampling via linear interpolation
level       ; linear(freq='15min')
water_temp  ; linear(freq='15min')
sac254      ; linear(freq='15min')

# 4: Drift correction
sac254      ; correctDrift(target='sac254_corr', maintenance_field='maint',
                  model='exponential')

# 5: Multivariate spike detection using kNN and STRAY
# Normalization of variables for kNN
level       ; transform(target='level_norm', func='zScore', freq='20D')
water_temp  ; transform(target='water_temp_norm', func='zScore', freq='20D')
sac254_corr ; transform(target='sac254_norm', func='zScore', freq='20D')

# Flagging by kNN and STRAY
level_norm,water_temp_norm,
sac254_norm ; assignKNNScore(target='kNN_scores', freq='20D')
kNN_scores  ; flagByStray(freq='20D')

# Project results of STRAY-algorithm onto variables
level       ; transferFlags(field=['kNN_scores'])
water_temp  ; transferFlags(field=['kNN_scores'])
sac254_corr ; transferFlags(field=['kNN_scores'])
```

References

1. Reid, W.V.; Chen, D.; Goldfarb, L.; Hackmann, H.; Lee, Y.T.; Mokhele, K.; Ostrom, E.; Raivio, K.; Rockström, J.; Schellnhuber, H.J.; et al. Earth system science for global sustainability: grand challenges. *Science* **2010**, *330*, 916–917. doi:10.1126/science.1196263.
2. Zacharias, S.; Bogena, H.; Samaniego, L.; Mauder, M.; Fuß, R.; Pütz, T.; Frenzel, M.; Schwank, M.; Baessler, C.; Butterbach-Bahl, K.; et al. A network of terrestrial environmental observatories in Germany. *Vadose zone journal* **2011**, *10*, 955–973. doi:10.2136/vzj2010.0139.
3. Loescher, H.W.; Kelly, E.F.; Lea, R. National ecological observatory network: Beginnings, programmatic and scientific challenges, and ecological forecasting. In *Terrestrial Ecosystem Research Infrastructures*; CRC Press, 2017; pp. 27–52.
4. Mollenhauer, H.; Kasner, M.; Haase, P.; Peterseil, J.; Wohner, C.; Frenzel, M.; Mirtl, M.; Schima, R.; Bumberger, J.; Zacharias, S. Long-term environmental monitoring infrastructures in Europe: observations, measurements, scales, and socio-ecological representativeness. *Science of The Total Environment* **2018**, *624*, 968–978. doi:10.1016/j.scitotenv.2017.12.095.
5. Heiskanen, J.; Brümmer, C.; Buchmann, N.; Calfapietra, C.; Chen, H.; Gielen, B.; Gkritzalis, T.; Hammer, S.; Hartman, S.; Herbst, M.; et al. The Integrated Carbon Observation System in Europe. *Bulletin of the American Meteorological Society* **2021**, pp. 1 – 54. doi:10.1175/BAMS-D-19-0364.1.
6. Wohner, C.; Ohnemus, T.; Zacharias, S.; Mollenhauer, H.; Ellis, E.C.; Klug, H.; Shibata, H.; Mirtl, M. Assessing the bio-geographical and socio-ecological representativeness of the ILTER site network. *Ecological Indicators* **2021**, *127*, 107785. doi:10.1016/j.ecolind.2021.107785.
7. GoFair. FAIR Principles. https://www.go-fair.org/fair-principles/, accessed on 2021-08-26.
8. Crystal-Ornelas, R.; Varadharajan, C.; Christianson, D.; Damerow, J.; Weierbach, H.; Robles, E.; Ramakrishnan, L.; Faybishenko, B.; Pastorello, G. A library of AI-assisted FAIR water cycle and related disturbance datasets to enable model training, parameterization and validation **2021**. doi:10.2172/1769646.
9. Gandin, L.S. Complex quality control of meteorological observations. *Monthly Weather Review* **1988**, *116*, 1137–1156.
10. Wagner, R.J.; Boulger Jr, R.W.; Oblinger, C.J.; Smith, B.A. Guidelines and standard procedures for continuous water-quality monitors: station operation, record computation, and data reporting. Technical report, 2006.
11. Campbell, J.L.; Rustad, L.E.; Porter, J.H.; Taylor, J.R.; Dereszynski, E.W.; Shanley, J.B.; Gries, C.; Henshaw, D.L.; Martin, M.E.; Sheldon, W.M.; et al. Quantity is nothing without quality: Automated QA/QC for streaming environmental sensor data. *BioScience* **2013**, *63*, 574–585.
12. Fiebrich, C.A.; Morgan, C.R.; McCombs, A.G.; Hall, P.K.; McPherson, R.A. Quality Assurance Procedures for Mesoscale Meteorological Data. *Journal of Atmospheric and Oceanic Technology* **2010**, *27*, 1565 – 1582. doi:10.1175/2010JTECHA1433.1.
13. Jones, A.S.; Horsburgh, J.S.; Eiriksson, D.P. Assessing subjectivity in environmental sensor data post processing via a controlled experiment. *Ecological Informatics* **2018**, *46*, 86–96.
14. Organization, W.M. *Guide to the Global Observing System (WMO-No.488)*; Secretariat of the World Meteorological Organization Geneva, Switzerland, 2018.
15. Sturtevant, C.; Metzger, S.; Nehr, S.; Foken, T. Quality Assurance and Control. In *Springer Handbook of Atmospheric Measurements*; Springer, 2021; pp. 47–90.
16. Taylor, J.; Loescher, H. NEON's Fundamental Instrument Unit Dataflow and Quality Assurance Plan, 2012.
17. Durre, I.; Menne, M.; Gleason, B.; Houston, T.; Vose, R. Comprehensive automated quality assurance of daily surface observations. *Journal of Applied Meteorology and Climatology* **2010**, *49*. doi:10.1175/2010JAMC2375.1.
18. Yver-Kwok, C.; Philippon, C.; Bergamaschi, P.; Biermann, T.; Calzolari, F.; Chen, H.; Conil, S.; Cristofanelli, P.; Delmotte, M.; Hatakka, J.; et al. Evaluation and optimization of ICOS atmosphere station data as part of the labeling process. *Atmospheric Measurement Techniques* **2021**, *14*, 89–116. doi:10.5194/amt-14-89-2021.
19. Petzold, A.; Thouret, V.; Gerbig, C.; Zahn, A.; Brenninkmeijer, C.A.; Gallagher, M.; Hermann, M.; Pontaud, M.; Ziereis, H.; Boulanger, D.; et al. Global-scale atmosphere monitoring by in-service aircraft–current achievements and future prospects of the European research infrastructure IAGOS. *Tellus B: Chemical and Physical Meteorology* **2015**, *67*, 28452. doi:10.3402/tellusb.v67.28452.
20. Organization, W.M. *Guidelines on the global data-processing system (WMO-No.305)*; Secretariat of the World Meteorological Organization Geneva, Switzerland, 2003.
21. Estévez, J.; Gavilán, P.; Giráldez, J.V. Guidelines on validation procedures for meteorological data from automatic weather stations. *Journal of Hydrology* **2011**, *402*, 144–154.
22. Wagner, R.J.; Boulger Jr, R.W.; Oblinger, C.J.; Smith, B.A. Guidelines and standard procedures for continuous water-quality monitors: station operation, record computation, and data reporting. Technical report, 2006.
23. Vitale, D.; Fratini, G.; Bilancia, M.; Nicolini, G.; Sabbatini, S.; Papale, D. A robust data cleaning procedure for eddy covariance flux measurements. *Biogeosciences* **2020**, *17*, 1367–1391. doi:10.5194/bg-17-1367-2020.
24. Metzger, S.R.; Taylor, J.R.; Luo, H.; Loescher, H.W. Developing a quality assurance and quality control framework for NEON's eddy-covariance flux measurements.
25. Mauder, M.; Cuntz, M.; Drüe, C.; Graf, A.; Rebmann, C.; Schmid, H.P.; Schmidt, M.; Steinbrecher, R. A strategy for quality and uncertainty assessment of long-term eddy-covariance measurements. *Agricultural and Forest Meteorology* **2013**, *169*, 122–135.
26. Younes, S.; Claywell, R.; Muneer, T. Quality control of solar radiation data: Present status and proposed new approaches. *Energy* **2005**, *30*, 1533–1549.

27. Ingleby, B.; Huddleston, M. Quality control of ocean temperature and salinity profiles — Historical and real-time data. *Journal of Marine Systems* **2007**, *65*, 158 – 175. doi:10.1016/j.jmarsys.2005.11.019.
28. Gourrion, J.; Szekely, T.; Killick, R.; Owens, B.; Reverdin, G.; Chapron, B. Improved statistical method for quality control of hydrographic observations. *Journal of Atmospheric and Oceanic Technology* **2020**, *37*, 789–806. doi:10.1175/JTECH-D-18-0244.1.
29. Dorigo, W.; Xaver, A.; Vreugdenhil, M.; Gruber, A.; Dostálová, A.; Sanchis-Dufau, A.; Zamojski, D.; Cordes, C.; Wagner, W.; Drusch, M. Global automated quality control of in situ soil moisture data from the International Soil Moisture Network. *Vadose Zone Journal* **2013**, *12*. doi:10.2136/vzj2012.0097.
30. Liao, W.; Wang, D.; Wang, G.; Xia, Y.; Liu, X. Quality control and evaluation of the observed daily data in the North American Soil Moisture Database. *Journal of Meteorological Research* **2019**, *33*, 501–518. doi:10.1007/s13351-019-8121-2.
31. Taylor, J.R.; Loescher, H.L. Automated quality control methods for sensor data: a novel observatory approach. *Biogeosciences* **2013**, *10*, 4957–4971.
32. Gourrion, J.; Szekely, T.; Killick, R.; Owens, B.; Reverdin, G.; Chapron, B. Improved statistical method for quality control of hydrographic observations. *Journal of Atmospheric and Oceanic Technology* **2020**, *37*, 789–806.
33. You, J.; Hubbard, K.G.; Nadarajah, S.; Kunkel, K.E. Performance of quality assurance procedures on daily precipitation. *Journal of Atmospheric and Oceanic Technology* **2007**, *24*, 821–834.
34. Hubbard, K.G.; You, J. Sensitivity analysis of quality assurance using the spatial regression approach—A case study of the maximum/minimum air temperature. *Journal of Atmospheric and Oceanic Technology* **2005**, *22*, 1520–1530.
35. Smith, D.E.; Metzger, S.; Taylor, J.R. A transparent and transferable framework for tracking quality information in large datasets. *PLoS One* **2014**, *9*, e112249.
36. Durre, I.; Menne, M.J.; Vose, R.S. Strategies for evaluating quality assurance procedures. *Journal of Applied Meteorology and Climatology* **2008**, *47*, 1785–1791.
37. Collier-Oxandale, A.; Feenstra, B.; Papapostolou, V.; Polidori, A. AirSensor v1. 0: Enhancements to the open-source R package to enable deep understanding of the long-term performance and reliability of PurpleAir sensors. *Environmental Modelling & Software* **2022**, *148*, 105256.
38. Metzger, S.; Durden, D.; Sturtevant, C.; Luo, H.; Pingintha-Durden, N.; Sachs, T.; Serafimovich, A.; Hartmann, J.; Li, J.; Xu, K.; et al. eddy4R 0.2. 0: a DevOps model for community-extensible processing and analysis of eddy-covariance data based on R, Git, Docker, and HDF5. *Geoscientific Model Development* **2017**, *10*, 3189–3206.
39. Urraca, R.; Sanz García, A.; Sanz-Garcia, I. BQC: A free web service to quality control solar irradiance measurements across Europe. *Solar Energy* **2020**, *211*, 1–10. doi:10.1016/j.solener.2020.09.055.
40. Long, C.N.; Dutton, E.G. BSRN Global Network recommended QC tests, V2. x **2010**.
41. Schmithüsen, H.; Sieger, R.; König-Langlo, G. BSRN Toolbox V2. 0–a tool to create quality checked output files from BSRN datasets and station-to-archive files **2012**.
42. Horsburgh, J.S.; Reeder, S.L.; Jones, A.S.; Meline, J. Open source software for visualization and quality control of continuous hydrologic and water quality sensor data. *Environmental Modelling & Software* **2015**, *70*, 32–44.
43. Fischetti, T. *assertr: Assertive Programming for R Analysis Pipelines*, 2022. https://docs.ropensci.org/assertr/ (website) https://github.com/ropensci/assertr.
44. Caveness, E.; GC, P.S.; Peng, Z.; Polyzotis, N.; Roy, S.; Zinkevich, M. Tensorflow data validation: Data analysis and validation in continuous ml pipelines. Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, 2020, pp. 2793–2796.
45. Scully-Allison, C.; Le, V.; Fritzinger, E.; Strachan, S.; Harris Jr, F.C.; Dascalu, S.M. Near real-time autonomous quality control for streaming environmental sensor data. *Procedia computer science* **2018**, *126*, 1656–1665.
46. Sheldon, W.M. Dynamic, rule-based quality control framework for real-time sensor data. Proceedings of the Environmental Information Management Conference. Citeseer, 2008, pp. 145–150.
47. Wilkinson, M.D.; Dumontier, M.; Aalbersberg, I.J.; Appleton, G.; Axton, M.; Baak, A.; Blomberg, N.; Boiten, J.W.; da Silva Santos, L.B.; Bourne, P.E.; et al. The FAIR guiding principles for scientific data management and stewardship. *Scientific data* **2016**, *3*, 1–9. doi:10.1038/sdata.2016.18.
48. Wollschläger, U.; Attinger, S.; Borchardt, D.; Brauns, M.; Cuntz, M.; Dietrich, P.; Fleckenstein, J.H.; Friese, K.; Friesen, J.; Harpke, A.; et al. The Bode hydrological observatory: a platform for integrated, interdisciplinary hydro-ecological research within the TERENO Harz/Central German Lowland Observatory. *Environmental Earth Sciences* **2017**, *76*, 1–25. doi:10.1007/s12665-016-6327-5.
49. Rebmann, C.; Claudia, S.; Sara, M.J.; Sebastian, G.; Matthias, Z.; Luis, S.; Matthias, C. Integrative measurements focusing on carbon, energy and water fluxes at the forest site 'Hohes Holz' and the grassland 'Grosses Bruch'. EGU General Assembly Conference Abstracts, 2017, p. 9727.
50. Rinke, K.; Kuehn, B.; Bocaniov, S.; Wendt-Potthoff, K.; Büttner, O.; Tittel, J.; Schultze, M.; Herzsprung, P.; Rönicke, H.; Rink, K.; et al. Reservoirs as sentinels of catchments: the Rappbode reservoir observatory (Harz Mountains, Germany). *Environmental earth sciences* **2013**, *69*, 523–536. doi:10.1007/s12665-013-2464-2.
51. Iglewicz, B.; Hoaglin, D.C. *How to detect and handle outliers*; Vol. 16, Asq Press, 1993.
52. Talagala, P.D.; Hyndman, R.J.; Leigh, C.; Mengersen, K.; Smith-Miles, K. A feature-based procedure for detecting technical outliers in water-quality fata from in situ sensors. *Water Resources Research* **2019**, *55*, 8547–8568. doi:10.1029/2019WR024906.

53. Talagala, P.D.; Hyndman, R.J.; Smith-Miles, K. Anomaly detection in high-dimensional data. *Journal of Computational and Graphical Statistics* **2021**, *30*, 360–374. doi:10.1080/10618600.2020.1807997.
54. Erhan, L.; Ndubuaku, M.; Di Mauro, M.; Song, W.; Chen, M.; Fortino, G.; Bagdasar, O.; Liotta, A. Smart anomaly detection in sensor systems: a multi-perspective review. *Information Fusion* **2021**, *67*, 64–79. doi:10.1016/j.inffus.2020.10.001.
55. Pang, G.; Shen, C.; Cao, L.; Hengel, A.V.D. Deep learning for anomaly detection: a review. *ACM Computing Surveys (CSUR)* **2021**, *54*, 1–38. doi:10.1145/3439950.
56. Porter, J.H.; Hanson, P.C.; Lin, C.C. Staying afloat in the sensor data deluge. *Trends in ecology & evolution* **2012**, *27*, 121–129. doi:10.1016/j.tree.2011.11.009.
57. Rode, M.; Wade, A.J.; Cohen, M.J.; Hensley, R.T.; Bowes, M.J.; Kirchner, J.W.; Arhonditsis, G.B.; Jordan, P.; Kronvang, B.; Halliday, S.J.; et al. Sensors in the stream: the high-frequency wave of the present. *Environmental Science & Technology* **2016**, *50*, 10297–10307. doi:10.1021/acs.est.6b02155.