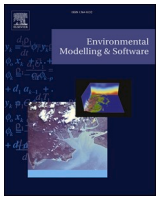
Environmental Modelling and Software 151 (2022) 105364

Contents lists available at ScienceDirect

Environmental Modelling and Software

journal homepage: www.elsevier.com/locate/envsoft



Toward automating post processing of aquatic sensor data

Amber Spackman Jones [a,b,*], Tanner Lex Jones [c], Jeffery S. Horsburgh [d]

[a] Department of Civil and Environmental Engineering and Utah Water Research Laboratory, Utah State University, 8200 Old Main Hill, Logan, UT, USA
[b] U.S. Geological Survey, Logan, UT, 84321, USA
[c] Space Dynamics Laboratory and Department of Electrical and Computer Engineering, Utah State University, 1695 North Research Park Way, North Logan, UT, USA
[d] Department of Civil and Environmental Engineering and Utah Water Research Laboratory, Utah State University, 8200 Old Main Hill, Logan, UT, USA

ARTICLE INFO

Keywords:
Aquatic sensors
Quality control
Anomaly detection
Python
Data management
Software and data availability

ABSTRACT

Sensors measuring environmental phenomena at high frequency commonly report anomalies related to fouling, sensor drift and calibration, and datalogging and transmission issues. Suitability of data for analyses and decision making often depends on manual review and adjustment of data. Machine learning techniques have potential to automate identification and correction of anomalies, streamlining the quality control process. We explored approaches for automating anomaly detection and correction of aquatic sensor data for implementation in a Python package (pyhydroqc). We applied both classical and deep learning time series regression models that estimate values, identify anomalies based on dynamic thresholds, and offer correction estimates. Techniques were developed and performance assessed using data reviewed, corrected, and labeled by technicians in an aquatic monitoring use case. Auto-Regressive Integrated Moving Average (ARIMA) consistently performed best, and aggregating results from multiple models improved detection. pyhydroqc includes custom functions and a workflow for anomaly detection and correction.

Name of software

pyhydroqc.

Description

A Python package for automated detection and correction of anomalies in aquatic sensor data.

Developer and contact information

Amber Jones, amber.jones@usu.edu.

Year first available

2021.

Program language

Python 3.7.

Hardware required

Personal computer running Microsoft Windows, Apple MacOS, or Linux.

Software required

pyhydroqc uses the following Python packages, all of which are available via the Python Package Index (PyPI): numpy 1.19.1, pandas 1.1.0, matplotlib 3.3.0, scipy 1.5.2, pmdarima 1.6.1, tensorflow 2.3, keras 2.4.3, statsmodels 0.11.1, scikit-learn 0.23.2, os, warnings, pickle, random.

Software Availability

The pyhydroqc software is open-source and is released under the Berkeley Software Distribution Version 3 (BSD3) software license. It can be installed within a Python environment from the Python Package Index (PyPI) using the PIP utility. Source code, documentation, and examples for the software are freely available in GitHub at https://github.com/AmberSJones/pyhydroqc and Zenodo (Jones et al., 2022).

* Corresponding author. Department of Civil and Environmental Engineering and Utah Water Research Laboratory, Utah State University, 8200 Old Main Hill, Logan, UT, USA.
E-mail addresses: amber.jones@usu.edu (A.S. Jones), tanner.jones@sdl.usu.edu (T.L. Jones), jeff.horsburgh@usu.edu (J.S. Horsburgh).

https://doi.org/10.1016/j.envsoft.2022.105364
Received 2 July 2021; Received in revised form 21 December 2021; Accepted 2 March 2022
Available online 5 March 2022

Dataset Availability

A resource containing the input data, processing scripts, results, and code to generate plots in this manuscript is described and stored in Hydroshare (Jones et al., 2022).

Additional documentation

A resource containing an example Jupyter notebook with instructions is described and stored in HydroShare (Jones, 2022). All functions included in the package are documented here: https://ambersjones.github.io/pyhydroqc/

1. Introduction

Observation of environmental phenomena using in situ sensors is increasingly common as sensors and related peripherals become more affordable and as cyberinfrastructure and expertise to support their operation have grown (Hart and Martinez, 2006; Pellerin et al., 2016; Rode et al., 2016). Sensors are subject to environmental factors that affect measurements and their suitability for subsequent analyses. Data from environmental sensors include anomalous points and biases that are artifacts of instrument noise or drift, power failures, transmission errors, or unusual ambient conditions (Horsburgh et al., 2015; Wagner et al., 2006). Protocols for ensuring quality of environmental sensor data (quality assurance) and mechanisms for performing data post processing (quality control) are challenges and key components of sensor network cyberinfrastructure (Campbell et al., 2013; Gries et al., 2014; Jones et al., 2015). As the quantity of sensor data increases, there is a commensurate need for practices that ensure resultant data are of high quality for subsequent analyses and exploration (Campbell et al., 2013; Gibert et al., 2016).

In current practice, quality control post processing of sensor data is expensive and tedious. Tools exist to assist practitioners and technicians in reviewing data and performing corrections (Gries et al., 2014; Horsburgh et al., 2015; Sheldon, 2008); however, quality control remains a time consuming and manual process consisting of an interactive sequence of steps. Performing corrections generally requires expert knowledge about the sensor and the phenomena being observed as well as conditions at the monitoring location (Fiebrich et al., 2010; White et al., 2010). Furthermore, the quality control process involves subjectivity as individual technicians may make different correction decisions (Jones et al., 2018). As a result, it is difficult to transfer the institutional knowledge required to post-process data, and even for trained and experienced technicians, quality control remains a daunting task as datasets grow in size and complexity for environmental observatories with ongoing data collection. For one network, a substantial delay of approximately six months between data collection and availability of reviewed and processed datasets allowed for thorough review and correction (Jones et al., 2017). For cases where observations are used for real time decisions related to public health and water treatment, the impacts of anomalous data are costly.

As sensor datasets continue to grow, it is not tenable for scientists and technicians to manually perform quality control tasks (Gibert et al., 2018), neither is it advisable to use or publish data without performing corrections to mitigate for errors. As a result, there is a recognized need for automating and improving quality control post processing for high frequency in situ sensor data. In this vein, automated, data driven techniques to detect anomalies in streaming sensor data are documented in the realm of research (Hill and Minsker, 2010; Leigh et al., 2018; Russo et al., 2020; Talagala et al., 2019); however, they are unfamiliar to practitioners, generally lack robust and accessible software implementations, and are not typically reproducible. Furthermore, while basic checks and more complex algorithms may identify and flag potentially erroneous values (e.g., Dereszynski and Dietterich, 2007; Hill et al., 2009; Taylor and Loescher, 2013), these procedures are generally not capable of applying corrective actions. Thus, the specific questions we pursued with this research are: 1) how can data-driven methods be applied to automatically detect and correct anomalies in aquatic sensor data, and 2) how can these methods be packaged into an overall workflow and reusable software for general application?

Regression models are one class of data-driven techniques that can be used as anomaly detectors for time series data by making a prediction based on previous data (either univariate or multivariate) and comparing the residual of the modeled and observed values to a threshold. Because regression models produce an estimate, they are well-suited for detection and correction of anomalous data. Although it is a substantial step in quality control post-processing, automated anomaly correction has not been widely examined. A handful of studies replaced raw data with modeled forecasts to exclude anomalies from model input but did not generate a corrected version of the dataset (Hill and Minsker, 2010; Leigh et al., 2018). In this work, we implemented and compared several regression models for anomaly detection and explored new approaches for anomaly correction.

Although effectively implemented for specific case studies, none of the techniques described in the cited studies have been packaged as accessible software for broad application and dissemination. Without reusable code, the specifics of the algorithms as implemented with environmental data cannot be examined, further tested, or applied to other datasets. Rather than a model calibrated to a specific variable/site combination, practitioners need tools that can be applied to a broad suite of variables and/or monitoring locations documented in a reusable and reproducible way. Thus, we sought to package the tools we developed as open-source software that could easily be deployed in a commonly available analytical environment.

In this paper, we present a Python package (pyhydroqc) that implements a set of methods for data-driven anomaly detection and correction for aquatic sensor data observed with high frequency in time. Our approach includes machine learning algorithms for detection, labeling, and correction of anomalous points. Multiple years of aquatic monitoring data from the Logan River Observatory (LRO) that have been reviewed and corrected by trained technicians were used as a case study for developing and testing automated detection and correction methods. The algorithms are encapsulated in a Python package that is publicly available and open-source (see Software and Data Availability section). Example scripts are also shared as Jupyter Notebooks that can be run with case study data to demonstrate the functionality and performance of the tools we developed. As there are many potential approaches to anomaly detection, additional techniques can be incorporated by adding new functions to the package that can be intrgrated into the workflow. Thus, the specific contributions of this work include: 1) advancing the algorithms and methods for automated quality control of aquatic sensor data, and 2) developing and demonstrating software tools that can make the process more approachable for data technicians and scientists. We anticipate that this work will be of interest to researchers, practitioners, and technicians that maintain environmental monitoring networks. The Python package can be used in any Python environment, and potential users should be familiar with scripting in Python (or a similar language), but do not need specific training or expertise.

Section 2 outlines the methods we implemented for detecting anomalies and performing corrections in the context of the structure and design of the pyhydroqc Python package, including a description of the case study that drove the implementation. In Section 3, we report the performance of the techniques on case study data and offer recommendations for next steps, followed by conclusions in Section 4. Appendix A contains related background including an overview of relevant literature and additional motivation for the work reported.

2. Methods

2.1. pyhydroqc software design and implementation

This work implements methods for anomaly detection and correction for environmental time series data within a Python-based software package. A subset of data-driven regression models are situated within an overall workflow that includes practical steps to facilitate anomaly detection and correction. The following sections describe the approaches for anomaly detection and correction, including details of how the software supports the workflow.

While many classes of algorithms could be used for detecting anomalies in aquatic sensor data, we selected time series regression models that were relatively straightforward to implement and that we

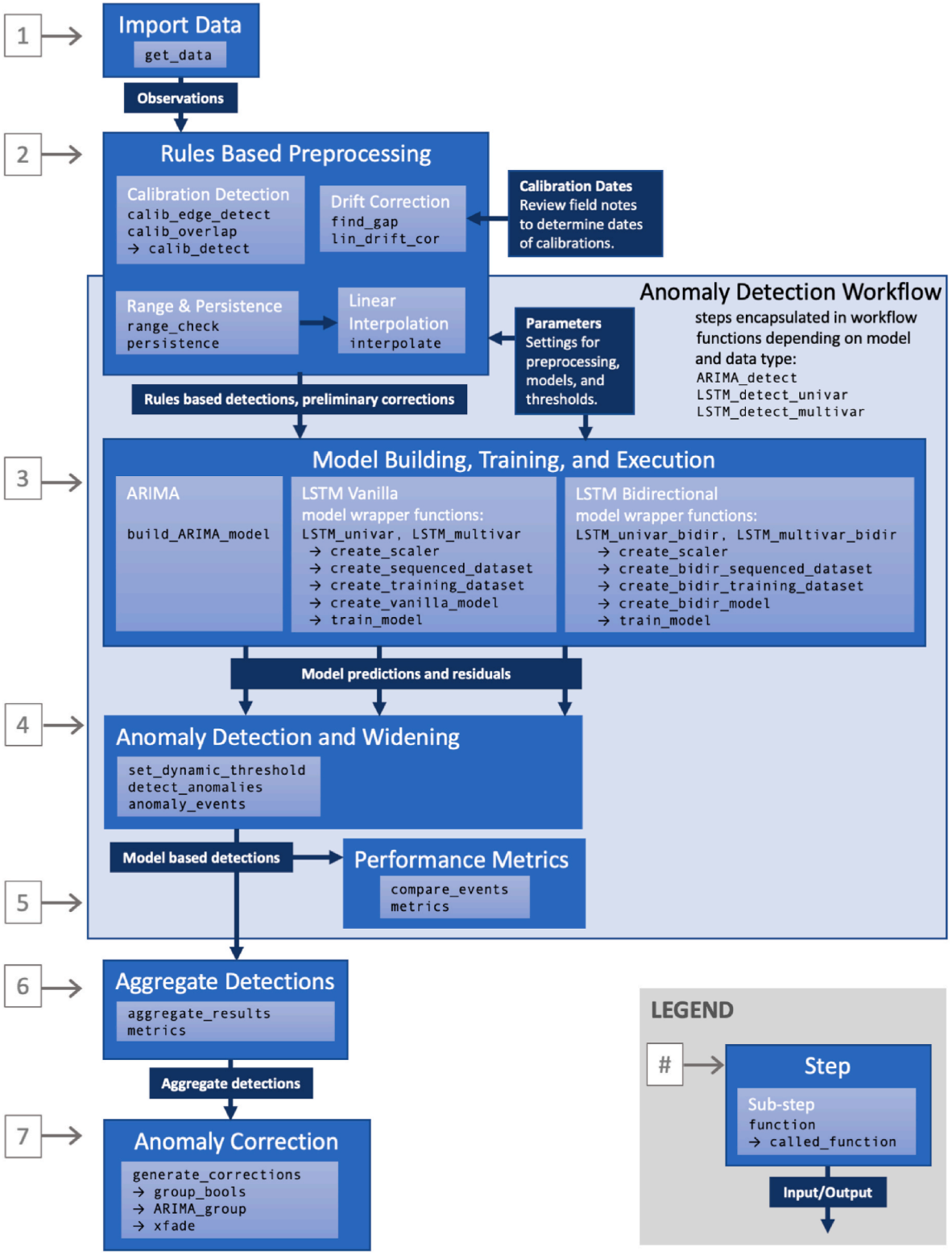


Fig. 1. Workflow for steps and functions in pyhydroqc. Numbers on the left correspond to steps in the process listed in Section 2.1.

anticipate will meet the needs and considerations of many applications. Specifically, we investigated auto-regressive integrated moving average (ARIMA), several types of long short-term memory (LSTM), and Facebook Prophet. ARIMA has been successfully implemented to detect anomalies in environmental data (Hill and Minsker, 2010; Leigh et al., 2018; Papacharalampous et al., 2019). LSTM is a class of Artificial Neural Networks (ANNs), and though applications to environmental data anomalies are limited, studies from other fields have detected anomalies with LSTM models (Hundman et al., 2018; Lindemann et al., 2019; Malhotra et al., 2016; Yin et al., 2020). Prophet was investigated but not included in the Python package. Because Prophet is geared toward social media and business applications (Taylor and Letham, 2018), we found that its applicability to environmental data is insufficient. It failed to capture seasonal shifts in the timing of daily cycles, and model features did not represent environmental phenomena. This paper focuses on a subset of models, but the modular design of the Python package allows for the implementation of additional techniques.

The software design and development were driven by the following steps as a workflow for anomaly detection and correction (Fig. 1), and each is described in more detail in the sections that follow.

1. Import raw sensor data into a memory-resident data structure.
2. Perform rules-based anomaly detection and correction as a first pass at quality control, including addressing sensor calibration.
3. Build one or more models for predicting observed values:
   a. Determine model hyperparameters.
   b. Transform and scale data if necessary.
   c. Build and fit models.
   d. Execute the model to determine model predictions and residuals.
4. Post-process model results:
   a. Determine dynamic thresholds based on model residuals and user-defined parameters.
   b. Detect anomalies where the absolute value of the model residual exceeds the defined threshold.
   c. Widen and index anomalous events.
5. Compare technician labeled and detected anomalous events (rules-based and model-based detections, inclusive) to assign confusion matrix categories and report metrics. (This step is only applicable if labeled data are available.)
6. Combine detections identified by multiple models for an aggregate anomaly detection (if rules-based detection has been performed, those detections are included).
7. Perform model-based correction for points identified as anomalous.

In addition to performing the workflow steps, requirements that drove our design included: 1) open-source software development to facilitate deployment and use by others; 2) cross-platform compatibility for use on Windows, MacOS, and Linux platforms; 3) modular and extensible architecture that enables each workflow step to be executed independently along with integration of new/additional functionality; and 4) simple deployment. A Python package was selected as the platform for software implementation. The Python language meets the open-source and cross-platform requirements, and existing tools and libraries in Python support steps in the workflow, including loading and manipulating large datasets and developing data-driven models. In a Python package, functions that comprise each step in the workflow can be called by scripts in a modular manner. Each of the steps can be performed independently, facilitating flexibility in use. A Python package also supports extensibility as new functions can be added without impacting existing functionality. Finally, Python packages can be published to the Python Package Index (PyPI, https://pypi.org/) making deployment straightforward and ensuring that algorithms can be applied in any Python coding environment.

The anomaly detection and correction workflow steps are encapsulated by functions in the pyhydroqc Python package described in the following sections. High level workflow wrapper functions ('ARIMA_detect', 'LSTM_univar_detect', and 'LSTM_multivar_detect') call more granular functions specific to each data and model type to perform steps 2–7 (Fig. 1) and generate objects of the 'ModelWorkflow' class. For clarity, each function is named and described in this paper; however, most users will use the overarching workflow function calls. Example Python scripts and Jupyter Notebooks (see Software Availability section) illustrate how the workflow functions are implemented for the data use case described in this paper. A full list of functions with inputs and outputs is found in Appendix B and with the package documentation.

2.1.1. Data format and import

pyhydroqc operates on pandas data frames, which are high performance, two-dimensional, tabular data structures for representing data in memory (pandas Development Team, 2008). Data frames can be created and saved or output as comma separated values (CSV) files. For pyhydroqc to perform anomaly detection and correction, input data need to be formatted as a data frame for each variable of interest indexed by date/time with a column of raw data. If technician labels or corrections are available (for determining anomaly detection metrics), they are included as additional columns in the data frame. In general, most date/time formats reported by sensor systems will be interpreted by pandas as date/time objects. In the rare case that the date/time format is not supported, some pre-processing may be required.

pyhydroqc also supports environmental sensor data formatted as one table or file with a single date/time column and multiple columns of measurements – one for each sensor output. For flat files with this structure, the pyhydroqc 'get_data' function wraps the 'read_csv' function from the pandas library to import data into Python and parse into separate pandas data frames for each variable as required by the anomaly detection and correction functions.

2.1.2. Rules-based detection and correction

Rules-based detection is an important precursor to detection using models (Leigh et al., 2018; Taylor and Loescher, 2013), and the results of this step contribute to the overall set of detected anomalies. Whether a result of sensor failure or another cause, some anomalies are "low hanging fruit" that can be detected by rules-based preprocessing that performs a first pass of the data. Preprocessing the data is motivated, in part, by the need to train models on a dataset absent of extreme outliers or artifacts that models cannot capture. By applying rules-based anomaly detection and correction, a first degree of correction is made for subsequent input into data driven models. We created Python functions to detect and correct out of range and persistent data. Furthermore, some aquatic sensors commonly exhibit drift, which requires sensor calibration and subsequent data correction. Because calibration shift and the preceding drift are subtle and difficult for any type of model to detect, we developed a rules-based routine that attempts to identify these events. Basic correction methods for these anomaly types were also implemented as Python functions.

2.1.2.1. Range and persistence checks. The function 'range_check' adds a column to the data frame and populates it with an anomalous label if the observation is outside of user-defined thresholds or a valid label if it is within the thresholds. Ranges should be determined specific to each sensor based on physics and the environment in which the sensor is deployed and can be refined based on site specific patterns. Data persistence refers to the sensor reporting a repeated value, which is unlikely in natural systems, although sensors may report repeated values due to limitations in resolution. For the 'persistence' function, the user defines a minimum duration of repeated values for data to be considered anomalous. If repeated values exceed that duration, the points are classified as anomalous by populating the column from the 'range_check' function. Beyond these basic checks, additional rules of increasing complexity could be added to the pyhydroqc package and the

anomaly detection workflow. Examples include ranges that vary seasonally, rate of change checks, and differencing checks.

Once anomalous points are identified by the Python functions that implement these rules, labels are carried through to the model-based detection steps. Labeled points are omitted from model training, either by logical exclusion, or, for models requiring an unbroken time series for training, by interpolating between valid points. Linear interpolation is performed (using the 'interpolate' function) over the entire time series as a preliminary correction step so that model input is more valid. If the complete workflow is followed, values initially corrected using linear interpolation are replaced by the model-based correction described in Section 2.1.9.

2.1.2.2. Calibration and drift correction. Environmental sensors commonly drift, and many aquatic sensors (specific conductance, pH, dissolved oxygen) require regular calibration to known standards. Drift causes a gradual increasing or decreasing trend separate from daily and seasonal patterns, and a calibration event manifests as a localized shift that corrects subsequent data up or down. These trends and shifts can be subtle and difficult to identify without a detailed record of calibration dates. In preliminary work, the model-based detectors described in subsequent sections were unable to consistently identify these data patterns. Detected shifts due to calibration events were undiscernible from other localized anomalies. Thus, it is important to address calibration events early in the quality control process because it is preferable that model-based detectors be trained on data that are free from drift.

For calibration and drift correction, we implemented functions to mimic a typical manual workflow. Performing post-processing correction for drift and calibration involves review of data, comparison of field records to data shifts to identify points corresponding to calibrations, and application of a drift correction that uses start and end points and the gap of the calibration shift to retroactively correct data between two calibrations. In our experience, calibration events are typically reviewed and corrected one at a time.

While recognizing the difficulty of definitively identifying calibration events in an automated way, we designed functions for detection (functions 'calib_edge_detect', 'calib_detect', 'calib_overlap') and correction (functions 'find_gap', 'lin_drift_cor') of data affected by drift and calibration. The algorithms take advantage of characteristics of calibration events, specifically that events only occur during certain hours of the day, they may involve a shift in observed data, and that when returned to the water, sensors may report the same values for several time steps until the sensor stabilizes. Two separate approaches identify calibration events: 1) where there is a discernible shift in the data, or 2) persistence occurs over a limited window of points. Both are restricted to hours and days when technicians would be in the field.

Given dates of calibration, a gap value needs to be specified for correcting past data. A function 'find_gap' identifies the greatest shift for a given window of time to determine a gap value and the precise point that should be shifted while accounting for outlier spikes commonly associated with calibrations. A function for linear drift correction, 'lin_drift_cor', corrects for drift and calibration events given start and end dates and a gap value of the calibration shift. While the calibration event detectors may not adequately identify events, requiring technician review or input, this process is a step toward automation as it evaluates gap values according to a set of rules rather than arbitrary determination by technicians (as illustrated in Jones et al. (2018)) and allows for bulk correction of calibration events.

2.1.3. Model-based detection using ARIMA

ARIMA is a time series forecasting model where inputs correspond to past time steps of the variable of interest, and the output is a predicted value for that variable at the next time step. ARIMA uses three parameters to define a linear model (Equation (1)):

$$y_t = \sum_{i=1}^{p} \varphi_i y_{t-i} + \varepsilon_t - \sum_{i=1}^{q} \theta_i \varepsilon_{t-i} \quad (1)$$

where y_t is the model output or the prediction for time step t, p is the number of previous points in the series to be used in the model, q is the number of moving average terms to include, φ_i are the fitted coefficients for auto-regression, θ_i are fitted model coefficients for the moving average, and ε_t is the moving average error term. Not shown in the equation is the term d, which is the order of differencing applied to the data y before this equation is evaluated. The parameters (p, d, q) can be determined manually or automatically. Manual parameter determination involves time series decomposition and the review of autocorrelation plots, which is tedious for numerous data series. Automatic determination of the parameters is effective but can be computationally demanding. pyhydroqc includes a function 'pdq' for automated determination using the pmdarima package (Smith, 2017). Given (p, d, q), model training involves determining the values of the coefficients for the terms in the linear equation (φ_i and θ_i) based on actual data.

In pyhydroqc, the function 'build_arima_model' constructs and trains an ARIMA model given input time series data and input parameters (p, d, q). It relies on the sarimax function from the statsmodel package (Seabold and Perktold, 2010) to fit an ARIMA model (based on Equation (1)), make model predictions for each time step, and compare predictions to observations. Input data should be free from gaps, so the anomaly detection workflow uses output of the rules-based detection with linear interpolation of any identified anomalies as input for ARIMA modeling.

2.1.4. Model-based detection using LSTM

LSTM is a type of neural network model architecture specifically designed for time-dependent and sequenced data. LSTM models consist of recurrent "cells" or units, each corresponding to one time step. A cell uses "gates" to control the flow of information in and out of the cell and how much of the past data that the cell "remembers" for computing output. To train an LSTM model, the weights of the connections within and between the gates are iteratively refined based on training data.

There are many variations of LSTM architecture (Greff et al., 2017). For our implementation, we compared several LSTM model types that are appropriate to time series data modeling for anomaly detection: vanilla and bidirectional, univariate and multivariate. In contrast with other neural network architectures, for which many layers are advised for fitting data, more shallow LSTM have been used because of the internal complexity of LSTM cells (Géron, 2017; Greff et al., 2017; Hundman et al., 2018). Other model types could be constructed, model layers and complexity could be added, and the input parameters could be tuned to each time series. Parameters can be defined by users and can be adjusted to investigate sensitivity, and we describe our approach for parameter selection in Section 3.1.4. The objective of this work was not to achieve the best time series model, but rather to detect anomalies, so fine-tuning models was not required or pursued. Instead, comparisons were made between a few basic LSTM variations with the same parameter settings.

As mentioned, pyhydroqc workflow functions call multiple lower-level functions. For LSTM models, each type is implemented within the workflow function by an associated model wrapper function ('LSTM_univar', 'LSTM_multivar', 'LSTM_univar_bidir', 'LSTM_multivar_bidir'), which calls functions specific to that model type for preprocessing, model building, model training, and model evaluation (shown in Fig. 1 and described in the Jupyter Notebook example script). The model wrappers return objects of the class 'LSTMModelContainer,' containing model predictions and residuals for each time step, similar to the output of 'build_arima_model.' The model wrapper functions also include an option for saving LSTM models for future use, which is important because LSTM model training and development is stochastic, resulting in a new model each time the algorithm is run. We developed models for a particular sensor deployed to

a certain location, so the models are variable and location specific and can be reused for that data series after training.

2.1.4.1. Vanilla and bidirectional LSTM. pyhydroqc implements the "vanilla" type of LSTM model (Greff et al., 2017), which consists of a single layer LSTM in a sequence-to-one manner, i.e. the model returns a single output based on a sequence of inputs. Given a user-specified number of past time steps, the model output is a single value for the next point in time. "Bidirectional" LSTM models use observations both before and after the point of interest to provide information for model prediction. By encoding a vanilla LSTM model with a bidirectional wrapper, input data are traversed both forward and backward in sequence, and model output is the value to have occurred in the middle of the sequence. In pyhydroqc, parallel functions structure input data to contain a user specified number of time steps prior to the point of interest for vanilla LSTM and prior to and following the point of interest for bidirectional LSTM (functions further described in Section 2.1.5.3).

2.1.4.2. Univariate and multivariate LSTM. Either univariate or multivariate input data may be used for vanilla and bidirectional LSTM through the LSTM workflow functions and model wrapper functions. The workflow functions ('LSTM_detect_univar' and 'LSTM_detect_multivar') prepare data and report results for univariate or multivariate data and call the associated model wrapper functions ('LSTM_univar' and 'LSTM_univar_bidir' for univariate, 'LSTM_multivar_bidir' and LSTM_multivar' for multivariate). For multivariate data, the models use data for all observed variables as input and output estimates of the same variables for the point of interest. Model errors are examined for each variable, and independent thresholds are set for anomaly detection.

2.1.4.3. LSTM preprocessing, model building, and training. The functions for preprocessing, model building, and model training are compiled as sequenced steps in the LSTM model wrapper functions (Fig. 1). Preprocessing for LSTM models involves scaling, reshaping, and ensuring that training data are valid, which is facilitated by using the output of the rules-based detection. Data must be scaled so that extreme values do not have an outsized impact on the model, and pyhydroqc includes a function for scaling ('create_scaler') based on the standardscaler function from the scikitlearn package, which subtracts the mean and divides by the standard deviation to scale the data (Pedregosa et al., 2011). Reshaping data creates a sequence of immediately previous points (i.e., model input) for each data value (i.e., model output). pyhydroqc functions ('create_sequenced_dataset' and 'create_bidir_sequenced_dataset') reshape data based on a user-defined number of past time steps.

To build a model structure, the pyhydroqc functions 'create_vanilla_model' and 'create_bidir_model' use the Sequential model from the Keras package (Keras Development Team, n.d.) with model layers (LSTM, Dense, and Bidirectional) and the suite of user-specified hyperparameters accepted by the Sequential model. To train the model, the functions 'create_training_dataset' and 'create_bidir_training_dataset' select a subset of data based on a user-defined number of random points, ensuring that none were identified as anomalous by the rules-based detection. These points are reshaped and used for training the LSTM model. The function 'train_model' uses the Keras early stopping feature so that model training ceases when the error of the test and validation sets (randomly selected by the algorithm) are approximately equal.

2.1.5. Post processing: dynamic threshold determination and anomaly detection

A key component of model-based anomaly detection using regression approaches is determination of the threshold that regulates whether a point is marked as anomalous or valid. Aquatic data vary seasonally, daily, and with environmental events, changes that may not be adequately captured by a model. A dynamic threshold has the potential to improve detection accuracy by applying a narrower range (i.e., higher sensitivity) when the model predictions are more precise and a wider range when model predictions are more variable. In particular, by using a dynamic threshold, we hoped to identify localized outliers that are within the absolute expected range of values but are relatively distinct for a narrower time window and which were undetectable with a constant threshold.

pyhydroqc implements a dynamic threshold following the format of confidence intervals and prediction intervals used in other studies (Hundman et al., 2018; Leigh et al., 2018). For each data point, a threshold is determined based on a moving window of points (Equation (2)):

$$T = \begin{cases} \mu \pm z_{\alpha/2}\sigma, & if\ z_{\alpha/2}\sigma < min \\ \mu \pm min, & otherwise \end{cases} \quad (2)$$

where T is the threshold, μ is the mean of the user-defined moving window model residuals, σ is the standard deviation of the moving window model residuals, α is a user-defined value to adjust the width of the threshold, $z_{\alpha/2}$ is the $\alpha/2$ quantile of a normal distribution, and *min* is a user-defined parameter for the minimum threshold value. Note that *min* may be set to zero (having no effect) or to a non-zero value to prevent too many false positives - i.e., detections that are not anomalies. This can occur when model residuals are low over an extended period and the dynamic threshold is smaller than the resolution or uncertainty inherent in the sensor.

Given a time series of model residuals, the 'set_dynamic_threshold' function in pyhydroqc determines upper and lower thresholds for each point in a series using Equation (2) with a user-defined moving window – the number of points used to calculate μ and σ. The 'detect_anomalies' function then compares the dynamic threshold values to the residuals for each time step to determine whether a point is anomalous. If rules-based detection was performed, the anomalies detected in that step are propagated through the workflow and are included in the detections output by this step.

2.1.6. Post processing: anomaly events and widening

In comparing anomalies identified by the model-based detectors to anomalies labeled by technicians, we observed mismatches related to resolution and lags in model approximations related to model smoothing. When an anomaly is identified, either the technician or the algorithm must determine how many points to label. To address this in a systematic way, pyhydroqc generalizes anomalies into numbered "events" consisting of groups of anomalous points. By widening the detection window to include points before and after anomalies detected by the algorithm as well as points labeled by the technician, overlap between the two is more likely. In pyhydroqc, the 'anomaly_events' function groups contiguous anomalous points as events by adding a column to the data frame with incrementing numbers as an index for each anomalous event. To perform widening for each anomalous event, the function assigns the event's index to points before and after the event (the number of points is user-defined), effectively adding those points to the event.

2.1.7. Performance metrics

For data with technician labels, the function 'compare_events' determines valid and invalid detections by comparing events detected by the algorithm to those labeled by the technician. Each point is classified as true positive, true negative, false positive, or false negative. When there is any overlap between detected events and labeled events (i.e., any portion of a labeled event is detected), all points are classed as true positives to indicate that the labeled event was detected. For accuracy, the points assigned as anomalous on the edges of events by widening are removed from the event as part of this step.

A confusion matrix compares model classifications to actual data to

evaluate overall performance by reporting total true positives, true negatives, false positives, and false negatives (Leigh et al., 2018; Tan et al., 2019). Additional metrics that are commonly reported include positive predictive value (precision), negative predictive value, accuracy, recall, and F scores (Li et al., 2017). In pyhydroqc, the function 'metrics' determines the performance metric outputs in Table 1. As aggregates of precision and recall, F scores combine true positives, false positives, and false negatives into a single assessment score to assess models (Cook et al., 2020). The F1 score gives equal weight to false positives and false negatives while the F2 score gives greater weight to false negatives. F scores range from 0 to 1, with 1 being the upper bound.

Because anomalies are sparse relative to the total number of data points, the datasets are considered imbalanced (Chandola et al., 2009). Counts of true negatives are overwhelming, resulting in high accuracy, which may make it difficult to compare between models (Tan et al., 2019). As a result, anomaly detection focuses on true positives, false positives, and false negatives. Anomaly detection requires a balance between increasing true positives while reducing both false negatives and false positives, objectives that may be mutually exclusive and depend on model sensitivity. Our preferred approach is to err on the side of sensitivity in the detector to minimize false negatives (along with maximizing true positives) even at the expense of increased false positives. The F2 score supports this aim by more heavily weighting false negatives while the F1 score equally weights true negatives and false negatives (Cook et al., 2020).

2.1.8. Aggregate detections

In applying multiple models, rather than select the single best performing model, a robust approach is to aggregate results so that a point identified by any of the models as anomalous is considered a detection. To address this, pyhydroqc includes a function 'aggregate_results' for combining anomalies detected by the different model types into a single column of detected anomalies. The outcome of aggregation is that a point is classed as anomalous if it was detected by any of the considered models. When a point is identified as anomalous by either rules-based detection or by any of the models, it is denoted with a Boolean in columns of output data frames (one that corresponds to each model), so the source of the anomaly may be traced through the process. Because rules-based detections are propagated through the workflow and are present in the detections associated with each model, the aggregation automatically includes the rules-based detections.

Table 1
Performance metrics calculated in pyhydroqc and associated equations.

Metric	Definition	Equation
True Positives (TP)	Count of data points from valid detection events where model detection events overlap with labeled anomalous events.	
False Positives (FP)	Count of data points from invalid detections where model detection events did not overlap with labeled anomalous events.	
True Negatives (TN)	Count of data points which did not belong to either labeled events or model detection events.	
False Negatives (FN)	Count of data points from labeled events which were not detected by model(s).	
Positive Predictive Value (PPV)	Ratio of true positives to total positives.	$PPV = \frac{TP}{TP + FP}$
Negative Predictive Value (NPV) (or Specificity)	Ratio of true negatives to total negatives.	$NPV = \frac{TN}{TN + FN}$
Accuracy	Ratio of correctly identified points to all data points.	$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$
Recall (or Sensitivity)	Ratio of True Positives to the total number of labeled anomalies.	$Recall = \frac{TP}{TP + FN}$
F1	Assessment score that combines true positives, false positives, and false negatives. Perfect score = 1.	$F1 = \frac{2*PPV*Recall}{PPV + Recall}$
F2	Assessment score that combines true positives, false positives, and false negatives. Gives greater weight to false negatives than does F1. Perfect score = 1.	$F2 = \frac{5*TP}{5*TP + 4*FN + FP}$

2.1.9. Model-based correction

A primary goal of this work was to suggest corrections for anomalous points, which is enabled by using time series regression methods for anomaly detection. While the model predictions used to determine anomalies could be simply substituted as corrections, the prevalence of consecutive anomalous points means that anomalous points would be used to determine corrections. To prevent this, correction models were implemented at a more granular scale. A function 'generate_corrections' was developed that implements piecewise ARIMA models using the following steps:

1. Given a data frame of observations with anomalies detected, assign consecutive points with either valid or anomalous labels to alternating groups. The function 'group_bools' adds a column populated with 0 for valid points and assigns each anomalous event a unique integer.
2. Ensure that sets of valid data points are large enough to generate forecast predictions. Where valid data points are in between anomalous points and the duration is too small to use as model input, the function 'ARIMA_group' merges them with previous and subsequent anomalous points into one anomalous group by resetting the group's incrementing index.
3. For each anomalous group, beginning with the group of shortest duration and progressing in order of increasing duration, develop 2 ARIMA models: one based on the preceding valid points and one based on subsequent valid points (using a specified maximum number of points for model development). Use the piecewise models to make forecasts and backcasts and blend them using the function 'xfade' to get a single correction estimate for each point in the anomalous group.
4. In the data frame, populate a new column with the correction estimates for points in anomalous groups and with the observations for the points in valid groups.

To blend the forecast and backcast, the values are weighted according to the proximity to each end point of the anomalous event, as shown in Equation (3), which is encoded in the function 'xfade':

$$y_k = A_k \frac{N-k}{N+1} + B_k \frac{k+1}{N+1} \tag{3}$$

where y_k is the correction estimate for each time step k in the anomalous group, N is the total number of data points in the anomalous group to be corrected ($k = 0 \ldots N\text{-}1$), and A_k and B_k are the ARIMA forecasted and backcasted values, respectively. Examples in Section 3.4 illustrate this concept. Because the ARIMA correction is based on points immediately proximate, instead of using the hyperparameters and model generated for the dataset as a whole, each forecast and backcast is an individual ARIMA model with hyperparameters and model fit based on the window of valid data. Using more granular models allows models to be tuned to that local time window and helps prevent errors that might arise from not having enough valid data points to estimate a point (e.g., if $p = 9$ for the time series as a whole, at least 9 valid data points are required). To avoid overfitting and to conserve computational resources, the 'generate_corrections' function includes a user-defined limit on the duration of data used to develop and train piecewise models to generate the

forecasts and backcasts.

Instead of applying corrections sequentially, the correction function first corrects the events of shortest length and then corrects events of increasing duration. In this manner, corrected estimates are available as model inputs when needed for correcting longer events. This helps ensure that the period of valid data before or after an anomalous event is sufficient to capture patterns.

2.2. Experimental use case: Logan River Observatory data

The primary objective of this work was to advance automation of quality control post processing specifically for environmental sensor data. As an extensive test case, we used data collected within the LRO where high frequency monitoring is conducted at several climate and aquatic sites within the Logan River watershed in northern Utah, USA (http://lro.usu.edu, (Neilson et al., 2021). Monitoring sites were established and infrastructure was originally deployed using protocols described by Jones et al. (2017). The LRO is similar to many research sites throughout the world where in situ monitoring of aquatic, climatic, and terrestrial variables is performed in support of research activities. Utah State University manages the monitoring network including site maintenance and data dissemination (available at http://lrodata.usu.edu/).

The upper Logan River watershed consists of mountainous forest and rangeland with limited development while the lower watershed is agricultural and urban with multiple agricultural diversions. Hydrology is generally driven by snowmelt, and the upper watershed is characterized by karst topography. Aquatic monitoring sites are located in both the upper mountain/canyon and lower urban/agricultural sections and include sensors for water level, water temperature, pH, dissolved oxygen, specific conductance, and turbidity (Fig. 2).

Raw sensor observations are recorded on field dataloggers, streamed to a central base station, and loaded to an operational database (Jones et al., 2015). Technicians perform quality control post processing on collected data using a suite of interactive tools to generate a quality controlled copy of data (Horsburgh et al., 2015). In this process, technicians review data and consult with the record of field activities to identify, label, and correct anomalous points or events in the data. LRO data exhibit a number of anomaly types including outliers, artificial

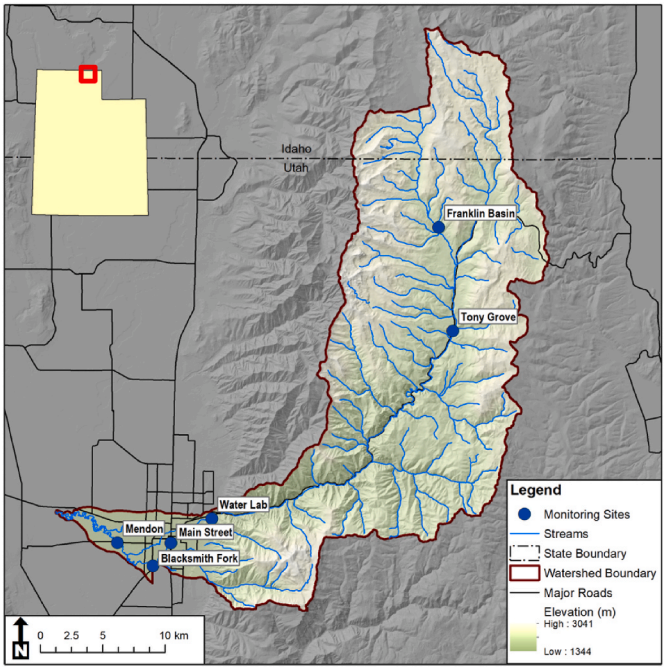


Fig. 2. Logan River Observatory showing locations of aquatic monitoring sites.

persistence, drift, and others described by Horsburgh et al. (2015). Currently, post processing consumes approximately half of a full-time technician's time with additional support from hourly assistants. We sought to move toward automated methods to reduce the time and resources required to perform quality control post processing.

To test pyhydroqc, we used data from the six aquatic sites shown in Fig. 2 for four variables common to aquatic monitoring and measured at all LRO sites: temperature, pH, specific conductance, and dissolved oxygen. Most of the sites include over 6 years of data at 15-min intervals with few to no gaps for both raw and labeled/corrected data. To assess performance, we used the metrics implemented in pyhydroqc to compare automated anomaly detection with the manual results produced by technicians. LRO sensor data were exported from a relational database (Observations Data Model, Horsburgh et al., 2008) to flat CSV files corresponding to each site indexed by a single date/time column with columns for the measurements output by each aquatic sensor. The pyhydroqc 'get_data' function was used to read the CSV files into individual pandas data frames for subsequent analyses. Testing the software against case study data was performed on a 2017 MacBook Pro laptop with 16 GB RAM and a 3.1 GHz quad-core Intel Core i7 processor.

3. Results and discussion

3.1. Preprocessing and settings

The following subsections present the parameters, configuration, and settings used by each anomaly detection and correction procedure. Anomalies detected by the combination of rules (range and persistence) and models with thresholds (ARIMA and LSTM) are reported together in Section 3.3.

3.1.1. Rules-based detection and correction: range and persistence checks

For the LRO data, range thresholds were determined specific to each sensor based on manufacturer reported ranges and were further refined according to past observations at each site (Table C1). The maximum allowable persistence durations were also based on review of raw observations and varied with sensor. Initially, persistence durations were set lower (~5–10 time steps); however, those durations resulted in many false positives as sensors regularly reported repeated values for more than 10 time steps. We observed that repeated values are often caused by limitations in sensor resolution, so persistence durations were increased (30–45 time steps; Table C1). Anomalies detected by these functions retained labels through subsequent steps, so the metrics resulting from rules-based detection are reported with the overall anomaly detection results in Section 3.3.

Anomalies detected by the range and persistence checks were initially corrected by linear interpolation, which is identical to the LRO protocol used by technicians to manually correct over short periods. However, in the pyhydroqc anomaly detection and correction workflow, the linear interpolation correction is an intermediate step to facilitate more accurate model development. These points retain an anomalous label through subsequent steps of the workflow and are eventually corrected using the model correction algorithm. Consequently, the final correction is performed by the model overwriting the interpolated points in the final, corrected dataset.

3.1.2. Rules-based detection and correction: calibration and drift correction

Results from the calibration detection algorithms were compared to calibration events identified and corrected by technicians for all sensors at one site (Main Street). The persistence functions ('calib_detect' and 'calib_overlap') identified about 25% of the calibration events with a high false positive rate (5X). The persistence we observed following a calibration may be specific to the sensors used in the LRO (YSI multiparameter sondes) and not broadly applicable. The edge detection function ('calib_edge_detect') identified about 40% of calibrations for pH but was less successful (<10%) for specific conductance and

dissolved oxygen. Additional effort could be applied to improve calibration event detection and to refine the parameters of the edge detection function (threshold and width). In theory, the model algorithms should identify these local shifts as anomalies; however, although the observed values may deviate from the modeled, the residuals were often within the dynamic thresholds (as defined in Table C1) and so were not detected as anomalies. Adjusting threshold settings may identify more calibration events but cause oversensitivity. Furthermore, the corrective action required for calibration events is different from that of other anomaly types, so the detection step should be separate.

Although calibration events were not automatically detected with high accuracy, the function for finding gap values was effective at determining valid gap values and end times for calibration shifts. In a review of the results of the 'find_gap' function, out of 100 distinct calibrations (the total for all variables at Main Street), revision was made for only 6 instances. With calibration dates and gap values as inputs, the function for linear drift correction was executed for all calibrated sensors (specific conductance, pH, dissolved oxygen) for the Main Street site. Many of the automatically determined gap values approximated the values used by the technician for correction, in which case the linear drift correction was comparable to the technician correction. Some automatically determined values were judged as preferable to the technician selected gap value (e.g., Fig. 3).

In our experience, selecting a viable gap value and performing drift correction can be the most time-consuming aspect of manual quality control. So, although the algorithms we designed were not successful in identifying a majority of calibration events, technicians typically record the dates of calibration, and automatically determining the gap value and performing drift correction in batch is a significant improvement. Furthermore, using an algorithm for this step increases consistency – the range of gap values selected by multiple technicians was the primary source of quality control subjectivity identified by Jones et al. (2018).

Based on our testing using the LRO data, our recommended workflow for addressing drift and calibration events is to: 1) identify a list of calibration dates (generally from field notes, although the pyhydroqc functions may be useful); 2) determine gap values and associated times using the 'find_gap' function; 3) review those shifts and make any adjustments; and 4) use the dates and gap values as inputs to the linear drift correction function. Code for performing these steps including generating plots of gap values for review are demonstrated in example notebooks.

3.1.3. Model-based detection and correction: threshold determination

The dynamic threshold used to evaluate differences between simulated and observed values directly impacts which observations are detected as anomalous or valid. For the LRO data, we used trial and error to settle on window sizes, alpha values, and minimum range values for determining thresholds (Table C1). The same threshold settings were used for all model types. We found that moving windows longer than a single day resulted in too much smoothing to the threshold and introduced artifacts due to daily patterns in model residuals. In general, window sizes of 5–10 h (corresponding to 20–40 time steps) were selected to balance between over-smoothing of longer windows and highly dynamic thresholds of shorter windows. An added benefit of smaller window sizes is that fewer computational resources are required to determine thresholds. Relatively small alpha values were selected (0.001–0.00001) to create a sufficiently high threshold range. With larger alpha values, the narrow threshold range was overly sensitive, resulting in too many false positives. Minimum values were similar for all sensors across sites, with a few exceptions. As illustrated in Fig. 4, the pattern of spread in thresholds tracks with the variability in model residuals, and residuals that exceed the threshold are detected anomalies.

3.1.4. Model-based detection and correction: model parameters and settings

To create ARIMA models, (p, d, q) were determined for each LRO data series over the full duration of data using the 'pdq' function (Table C1). To build, compile, and train LSTM models, consistent parameters and settings were used for all of the LRO data series and the several varieties of LSTM models (Table C2). Default settings and commonly used parameters (Géron, 2017; Keras Development Team, n. d.) were selected with minimal tuning to achieve the goal of satisfactory rather than perfect models. Models were trained with a randomized subset of 20,000 data points from each data series, corresponding to approximately 10% of the dataset. This number of data points was determined to be robust given that increasing the number of training points did not change the overall results, and training with 20,000 data points was less resource intensive compared with larger training sets. Anomalous events in both technician-labeled data and model-detected data were widened by a single point (widening factor = 1). This setting was used for all data series and all model types.

3.2. Anomaly detection example

Examples help demonstrate the performance of the workflow for both successful and unsuccessful anomaly detection (Fig. 5; additional examples in Appendix D). On 2018-11-11, the ARIMA model detected an event that was not labeled by the technician (false positive). Although this is a false positive, the model with a dynamic threshold behaved as

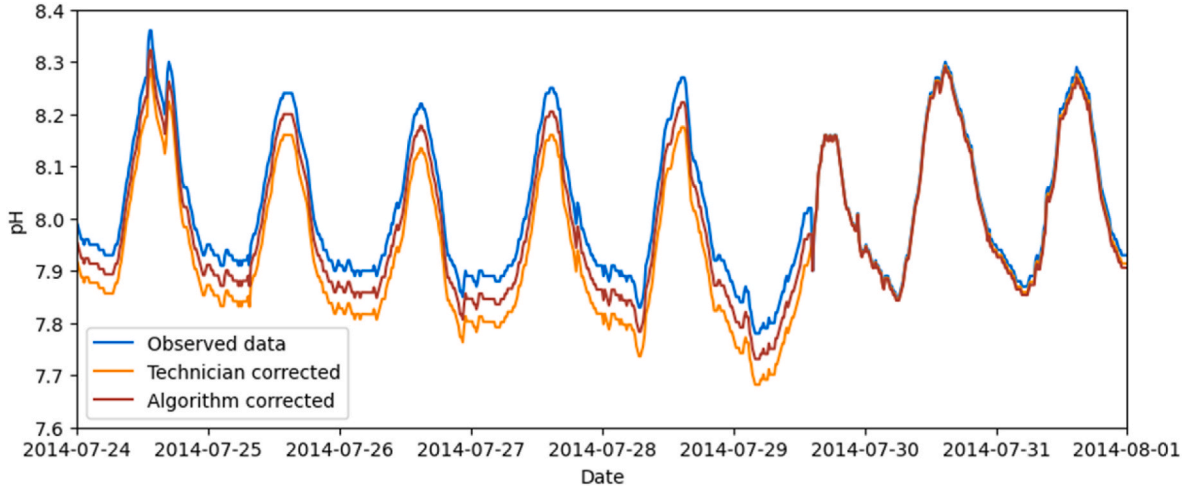


Fig. 3. Example of gap values and linear drift correction for pH at Main Street. A calibration shift occurred 2014-07-29. The data at the calibration were shifted by a gap value – determined either by the algorithm or by the technician, and data before the calibration were adjusted proportionately.

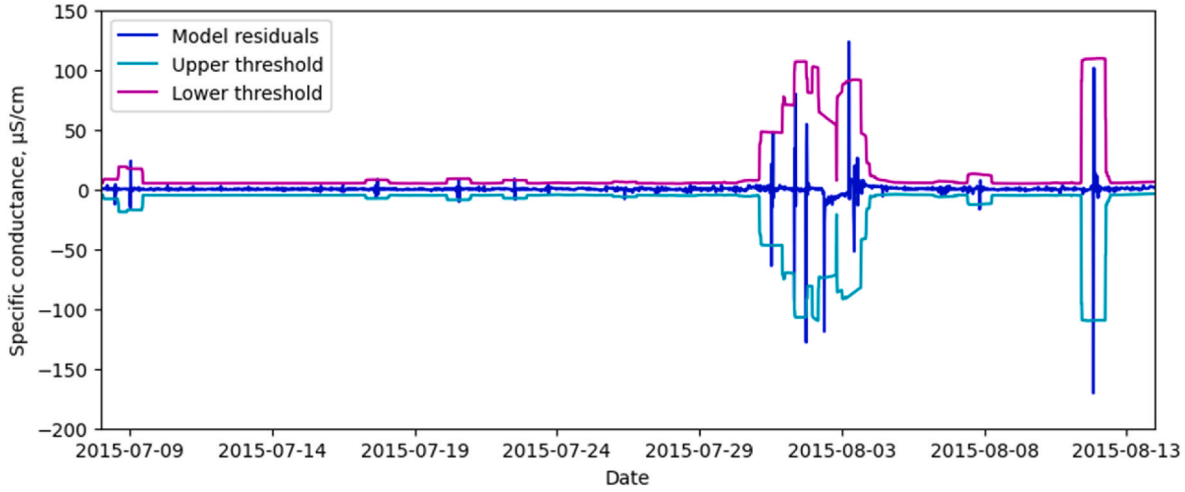


Fig. 4. Example of model residuals and dynamic thresholds for specific conductance at Main Street.

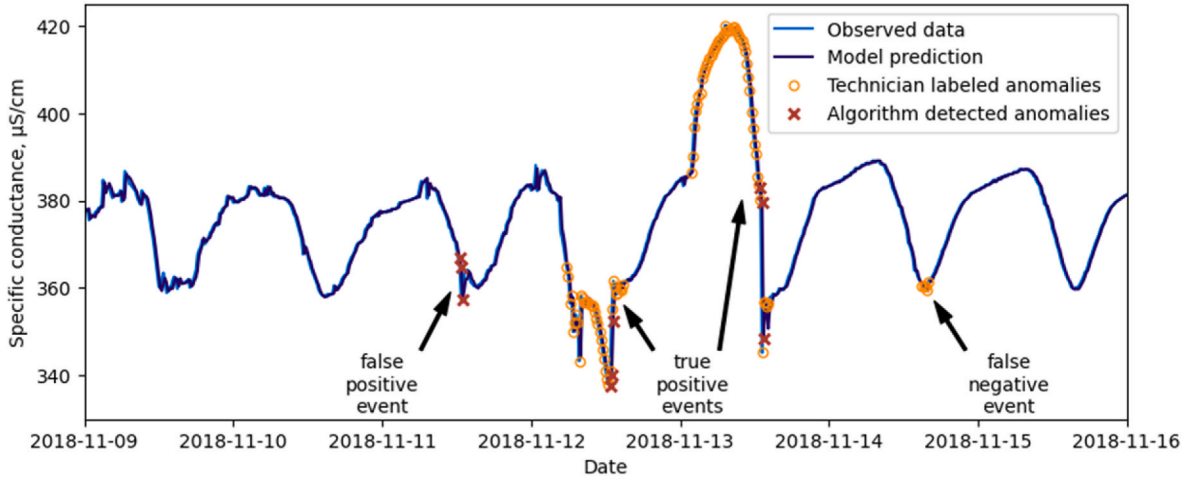


Fig. 5. Examples of anomalies detected using an ARIMA model for specific conductance at Tony Grove.

designed in detecting a localized outlier. The events on 2018-11-12 and 2018-11-13 consist of points both detected by the algorithm and labeled by the technician (true positive). Not all points labeled by the technician were detected as anomalies by the model; however, performing widening and considering the overlapping sets of points as anomalous events resulted in true positives for all of these points. The event on 2018-11-14 was not detected by the algorithm but was labeled by the technician (false negative). There is nothing in the original data to indicate that something was amiss, so it is unclear why the points were labeled as anomalous by the technician. The technician has expert knowledge or is following protocol that the algorithm is unable to discern. In assessing algorithm performance, we defer to technician labels as a benchmark. However, the quality control process is subjective (Jones et al., 2018) and data are not perfectly labeled, making reliance on technician labels as a gold standard problematic (Russo et al., 2020). In the LRO data, we identified numerous cases where it was unclear why some data points were labeled and others were not (see Appendix D), which may be due to multiple technicians and evolving protocols, among other reasons.

3.3. Combined anomaly detection results

The F2 scores for all time series (Table 2) combine true positives, false positives, and false negatives to indicate overall performance for each model type, rules-based detection, and an aggregate of all models. Higher scores indicate better model performance (F2 = 1 would be a perfect score). Fig. 6 is a visual illustration of the confusion matrix where each panel corresponds to a time series and each bar to a model type. The bottom portion of each bar (light blue) represents true positives, the middle portion (orange) represents false negatives, and the sum of those is equivalent to all technician labeled points. The top portion of each bar (purple) represents false positives. The dashed lines distinguish the proportion of anomalies identified by rules-based detection. True positives below the lower dashed line (black) were detected by rules while those above it were only detected by models. Likewise, false positives below the upper dashed line (gray) were detected by rules, and false positives above it were detected by only models. Anomalies detected by rules (those below each line) may have also been detected by models, so there may be overlap. The results illustrate some general trends regarding the performance of both rules-based and model-based detection.

3.3.1. Detections due to rules and threshold settings

For several time series, the rules-based algorithm accounts for the majority of anomaly (true positive) detections (e.g., temperature at several sites, dissolved oxygen at Franklin Basin). In these cases, the model detection did not provide many additional detections. In other cases (e.g., temperature at Tony Grove, all pH time series, most specific

Table 2
F2 score comparisons. Scores are reported for ARIMA and LSTM models for each time series as well as rules-based detection and the aggregate of all of the models. The aggregate column combines model results by classifying a point as anomalous if it was detected by any of the models. F2 = 1 would be a perfect score.

Monitoring Site	ARIMA	LSTM univar	LSTM univar bidir	LSTM multi	LSTM multi bidir	Rules-Based	Aggregate
Temperature							
Franklin Basin	0.926	0.840	0.842	0.840	0.841	0.764	0.920
Tony Grove	0.966	0.966	0.966	0.966	0.966	0.066	0.966
Water Lab	0.970	0.909	0.922	0.895	0.923	0.888	0.975
Main Street	0.546	0.571	0.650	0.569	0.625	0.548	0.709
Mendon	0.992	0.992	0.992	0.991	0.992	0.867	0.992
Blacksmith Fork	0.615	0.605	0.605	0.607	0.607	0.448	0.616
Specific Conductance							
Franklin Basin	0.985	0.403	0.410	0.977	0.723	0.176	0.986
Tony Grove	0.978	0.383	0.264	0.884	0.501	0.127	0.978
Water Lab	0.952	0.809	0.810	0.822	0.919	0.370	0.957
Main Street	0.935	0.876	0.884	0.872	0.904	0.155	0.928
Mendon	0.945	0.836	0.836	0.943	0.856	0.424	0.966
Blacksmith Fork	0.845	0.736	0.776	0.839	0.807	0.134	0.806
pH							
Franklin Basin	0.967	0.852	0.849	0.945	0.839	0.317	0.968
Tony Grove	0.946	0.654	0.638	0.658	0.632	0.064	0.945
Water Lab	0.966	0.954	0.932	0.934	0.929	0.175	0.969
Main Street	0.983	0.982	0.982	0.983	0.980	0.186	0.984
Mendon	0.995	0.983	0.848	0.849	0.847	0.396	0.995
Blacksmith Fork	0.989	0.983	0.982	0.958	0.955	0.125	0.990
Dissolved Oxygen							
Franklin Basin	0.496	0.467	0.457	0.470	0.459	0.429	0.497
Tony Grove	0.705	0.404	0.256	0.263	0.256	0.140	0.827
Water Lab	0.892	0.879	0.880	0.967	0.881	0.064	0.980
Main Street	0.967	0.943	0.942	0.946	0.944	0.194	0.968
Mendon	0.873	0.736	0.823	0.750	0.735	0.107	0.879
Blacksmith Fork	0.912	0.964	0.918	0.919	0.963	0.204	0.965
Average	0.889	0.780	0.769	0.827	0.795		

conductance and dissolved oxygen time series), the true positives are split between rules-based and model-based, indicating that the models captured anomalous events that the rules-based detection missed. This demonstrates the value of using both approaches in tandem.

In some cases, the success of the model(s) in detecting anomalies (true positives) is offset by a large number of false positives. Particularly high counts of false positives indicate oversensitivity, due to either persistence durations that are too short or to thresholds that are too tight, both of which may result in too many detections. In particular, dissolved oxygen at Franklin Basin and Mendon and specific conductance at Blacksmith Fork exhibit high rates of false positives. Given that most are under the rules-based line, the false positives are attributable to oversensitivity in rules (range check or persistence duration) rather than inadequate threshold settings. The similar rates of false positives between models for many time series indicates that using the same threshold settings for all model types is acceptable.

Cases with a large portion of false negatives (undetected anomalies) across models indicate that the models were not sensitive enough (e.g., temperature at Main Street and Blacksmith Fork). Better detection might occur with tighter thresholds or adjusted rules-based settings. Practitioners need to consider the tradeoffs with model sensitivity in determining threshold settings. Under the assumption that anomalies identified by the algorithm would be further reviewed by a technician, the thresholds can be set to capture more potential anomalies, erring on the side of false positives. However, sensitivity must be balanced to avoid excessive false positives from narrow thresholds.

3.3.2. Model comparison

The detections between all models were generally comparable (e.g., temperature at most sites, pH at most sites, dissolved oxygen at several sites), although, for a few time series, there were distinct variations in results between models (e.g., specific conductance at Franklin Basin and Tony Grove, dissolved oxygen at Tony Grove). ARIMA models gave the best average F2 score (Table 2) – they generally outperformed LSTM models for the cases with differences in model performance and were often slightly better than the LSTM models for the time series with comparable results. ARIMA was generally more sensitive – detecting more true positives than the LSTM models at the expense of detecting more false positives. Results from the LSTM models varied without a discernible pattern. In one case, the univariate bidirectional model excelled (temperature at Main Street), while in other cases the multivariate vanilla was preferred (specific conductance at Franklin Basin and Tony Grove, dissolved oxygen at the Water Lab). For some of these series, the more successful models detected a few points that were part of long events that were labeled anomalous by technicians, which improved the performance of those model types.

Although the multivariate and bidirectional models include more information in their input, either with additional variables or additional points in time, these models did not broadly outperform their simpler counterparts. With regard to the multivariate model, while there is some physical relationship between the variables of interest, the LSTM model has no explicit physical drivers. We might expect that the behavior of other variables related to diurnal cycle or ambient events to offer some predictive information; however, most types of anomalies occur for a single variable, independent of other sensors. Some conditions, such as sediment or ice built up around a multi-parameter sensing sonde, or an issue with power to the sensors, may impact more than one sensor simultaneously. In these cases, there is value in noticing concurrent anomalies in multiple sensors. Because not all studies measure all variables, single sensor algorithms are more versatile, and can perform well, as shown in these results. There may be systems in which variables are so physically related that a more directly dependent relationship/model can be more reliable.

Differences in anomaly detection between the model types could be due to several factors. ARIMA and LSTM models have inherently different structures with distinct processes for hyperparameter tuning

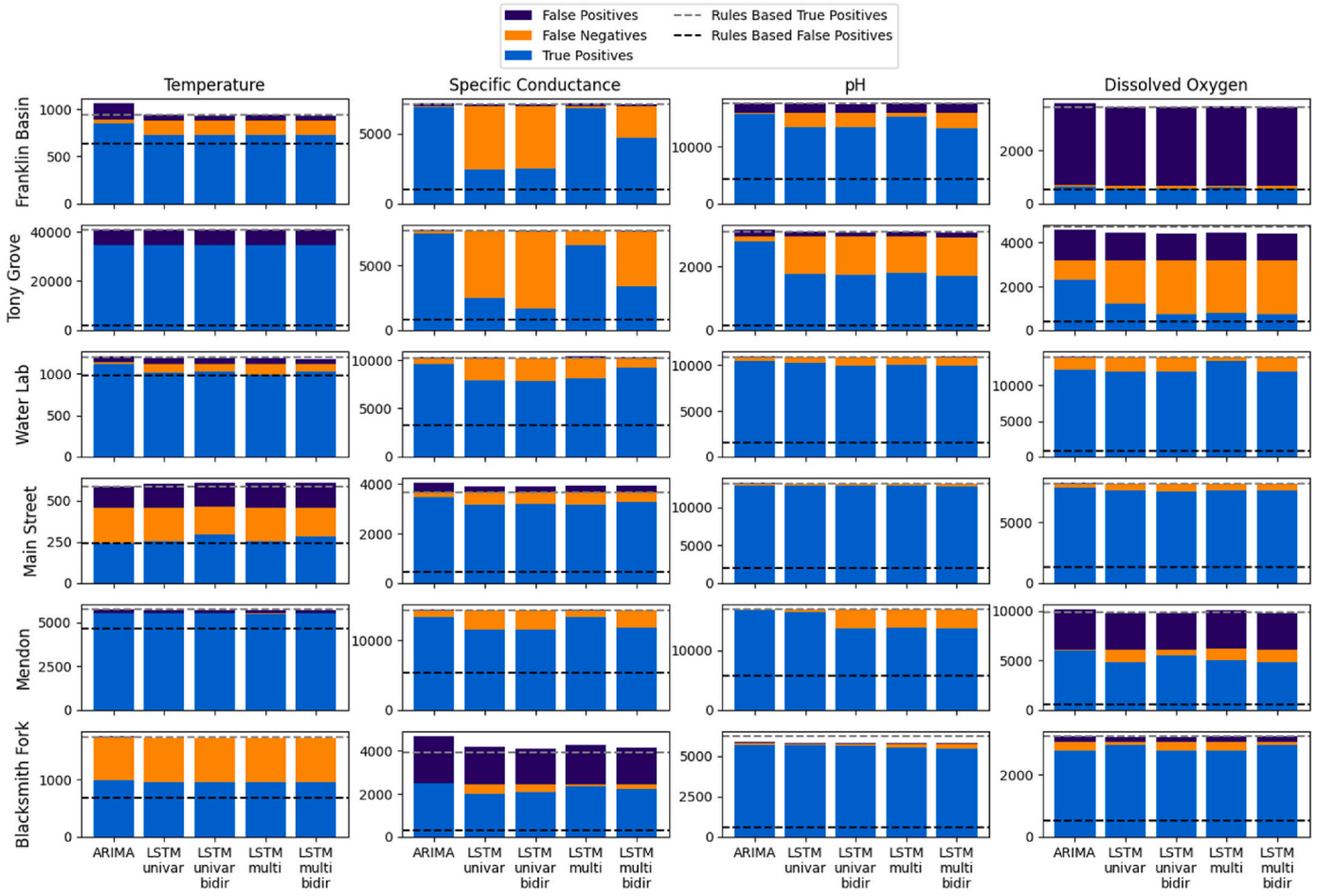


Fig. 6. Detection confusion matrix values for all time series (panels) and models (bars). y-axis values represent the count of observations that fall within each category shown in the legend. Dashed lines differentiate the proportions of detections from the rules-based detection and the model-based detection.

and model training. ARIMA models use a limited number of hyperparameters (three), which were tuned by automated optimization, while LSTM models include several hyperparameters for which minimal tuning was performed. It is possible that LSTM models could be improved with additional tuning; however, the process may not be worth the effort given that the objective of modeling was to detect anomalies rather than generate a perfect model. As one example, we observed LSTM models consistently biased toward the overall time series mean, which was reduced when developed with input sequences containing fewer previous data points (5 versus 10).

Another possible explanation for the poorer performance of LSTM models is a result of the training process. LSTM models were trained on a randomized subset of available data. Due to the stochastic nature of training data selection and initialization of weights, a new model is developed each time the algorithm is run (although pyhydroqc can save models for future use). If a distinct set of training data was used or learning converges to a local minimum, it may cause the seemingly arbitrary failure of some LSTM models on certain time series. To test this, LSTM models were regenerated. The resulting metrics were similar to those reported in Table 2. This indicates that the size of the training sets is sufficient so that the strength of the model does not depend on the specific, randomized subset of data used for training. Independently developing and training multiple models on the same time series is a straightforward check for training data robustness.

Although we tested across a range of sites that span elevation, land use, and hydrologic regime within the LRO, these locations do not represent the full spectrum of sites across the world. Investigating the suitability of the algorithm to additional physical settings is an important next step. More directly examining the performance of each model type related to physical characteristics of locations may help inform transferability of the techniques.

3.3.3. *Model aggregation*

The comparability of most of the results suggests that using any one of the models may be acceptable; however, rather than select a single model, aggregating detections by the multiple models may improve results. F2 scores of aggregated anomaly detection (Table 2) indicate overall good performance for most time series (F2>0.8), also illustrated by confusion matrix plots (Fig. 7). For some time series, the aggregation does not add high value, presumably because the same points were detected by multiple models. However, for a few time series in particular, aggregating detections of multiple models had a synergistic effect such that the aggregate F2 score is higher than that of any single model (e.g., temperature at Main Street, dissolved oxygen at Tony Grove). Lower F2 scores (<0.8) that persist after aggregating model detections are a result of either high rates of false positives (dissolved oxygen at Franklin Basin) or false negatives (temperature at Blacksmith Fork), both of which could be addressed by tuning rules and threshold settings as described rather than perfecting models.

The results affirm that time series regression methods with dynamic thresholds and widening are an effective tool for automating anomaly detection and correction, and implementing these techniques can streamline the quality control process. Without the models, a technician would need to review 200,000+ data points for each of the time series used in this case study. By using the pyhydroqc anomaly detection workflow, the number of data points for review (referring to combined

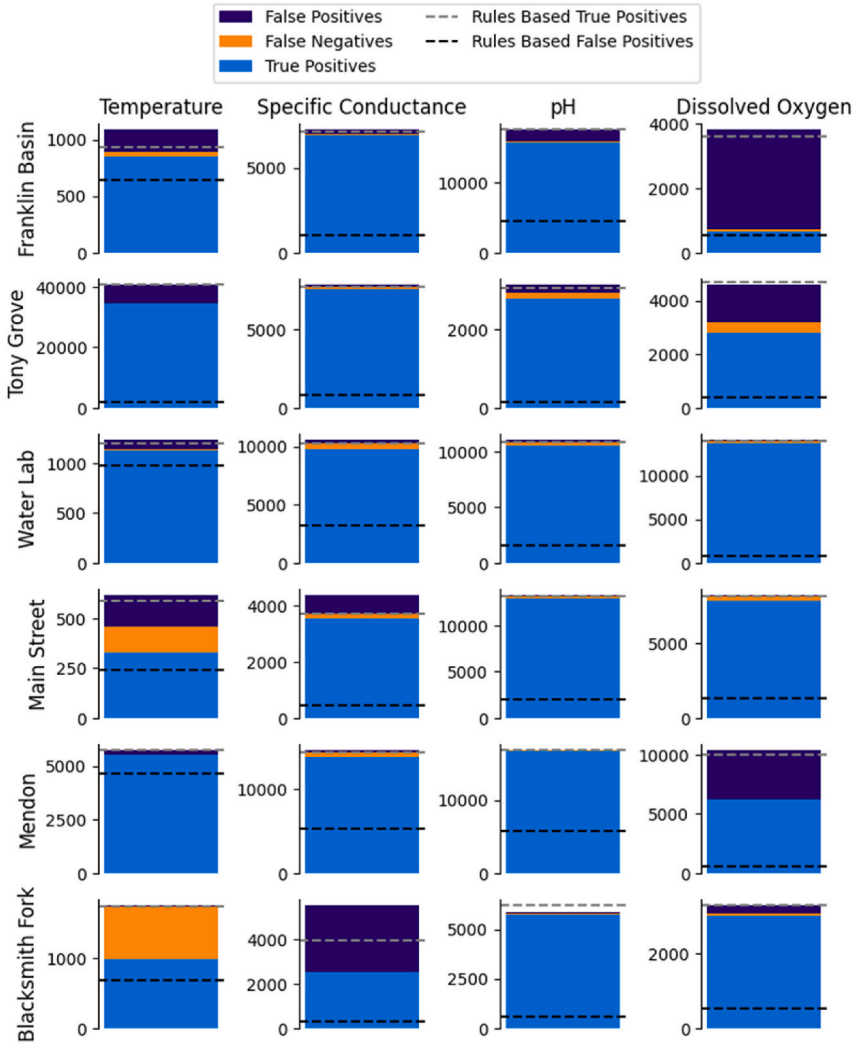


Fig. 7. Detection confusion matrix values for aggregate results for all time series. Symbology is as described for Fig. 6.

rules and model detections) is reduced by at least an order of magnitude (e.g., ~20,000 for pH at Franklin Basin), even for cases with high rates of false positives (e.g., ~4000 for dissolved oxygen at Franklin Basin).

3.4. Model-based correction examples

The model-based anomaly correction implemented in pyhydroqc generally resulted in smooth data profiles without outstanding non-linearities (Fig. 8). The method offers a viable path for correcting many anomalous events, although results varied depending on the duration, the variable, the season, and the reliability of anomaly detection. For shorter durations (e.g., approximately 2 h, Fig. 8a), the model-corrected data are similar to the technician correction (i.e., linear interpolation). For longer periods, the blended forecasts and backcasts can estimate patterns (diurnal cycles, Fig. 8b and c) that would not be practical for a technician to approximate. In these cases, technicians did not attempt corrections but set data to a "no data value" (−9999). In other cases, the model did not capture data patterns, particularly for extended periods (see Appendix D for examples). Some models overgeneralized and missed patterns while others focused on a single dominant feature. Overall, the correction algorithm better captured diurnal patterns in temperature and pH data while regular patterns in specific conductance and dissolved oxygen were less consistently approximated.

3.5. Combined correction results

Quantifying the overall performance of the correction algorithm for each time series is impractical because no gold standard exists for comparison. Algorithm-corrected data cannot be quantitatively compared to technician corrected data because the technician corrected data are subjective, contain correction and labeling errors, and include many periods where the values were set to a designated “no data value” (e.g., −9999 for the LRO). For correcting LRO data, technicians followed one of the following paths: 1) linear interpolation for periods less than 4 h, or 2) setting values to −9999 for longer periods where interpolation was deemed unreasonable. Technicians also performed linear drift correction between identified calibration events. The model-based correction algorithm is not designed to correct for drift, which was performed as part of the rules-based steps (Section 3.1.2).

Without a benchmark, correction algorithm performance cannot be definitively measured for each time series, leaving evaluation to be done qualitatively on a case-by-case basis (Section 3.4 and Appendix D). We considered simulating artificially introduced anomalies, which are then corrected and compared to valid raw data; however, it is unclear what frequency and duration of artificial anomalies would be appropriate and how to propagate artificial anomalies through multiple concurrently measured variables (i.e., in the case of multivariate models). We

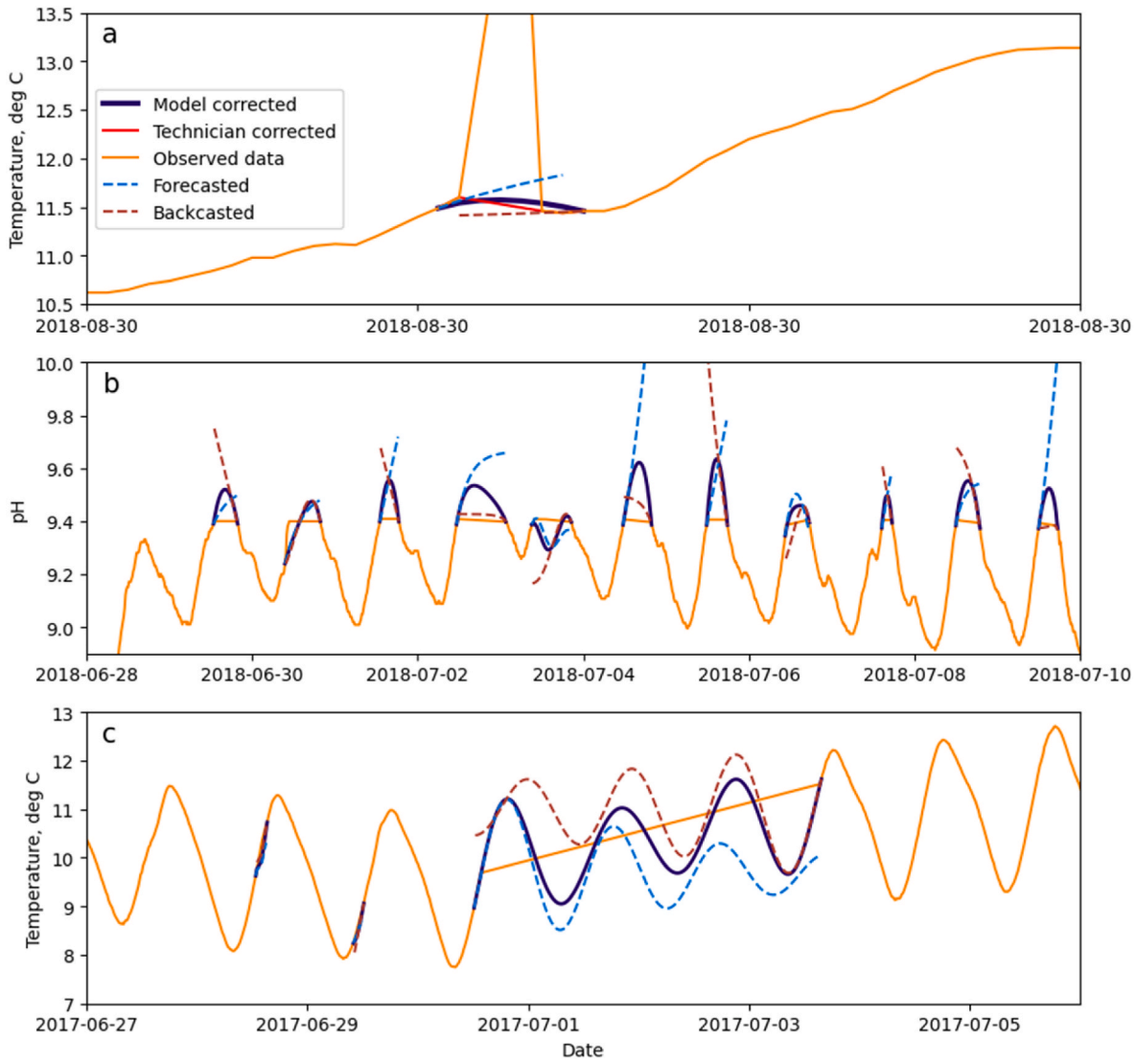


Fig. 8. Examples of successful correction using piecewise ARIMA models and the cross-fade technique. 8a: temperature at Water Lab, 8b: pH at Main Street, 8c: temperature at Water Lab.

determined that analysis to be outside the scope of this work. In an attempt to assess the value of the correction algorithm in terms of relative accuracy, we considered the total number of points in each series that were altered from the raw data by the technician or the algorithm and that were set to values outside of a valid range (Table 3). Ranges specific to each time series were adopted from the range checks in rules-based preprocessing (Table C1) to determine whether altered points were valid. Technician corrections resulting in invalid values generally correspond to data changed to the "no data value" of −9999. Causes of invalid values produced by the correction algorithm may include periods where anomaly detection was not adequately inclusive, so the points corrected by the algorithm were overly influenced by anomalous points that were not labeled as such (Figure C5). In another scenario, anomalous data may be close to the range limits resulting in forecasts, backcasts, and corrections outside of the valid range (e.g., the estimations of peaks in Fig. 8b exceed the upper limit for that time series).

Table 3
Technician and algorithm invalid changed data points. Counts represent the number of points where raw data were corrected to values outside of the valid range for that time series. The total number of data points for each series is ~200,000.

Monitoring Site	Temperature		Specific Conductance		pH		Dissolved Oxygen	
	Technician	Algorithm	Technician	Algorithm	Technician	Algorithm	Technician	Algorithm
Franklin Basin	584	8	3123	92	11,259	837	568	1656
Tony Grove	44	8	1517	13	482	0	692	1185
Water Lab	22	0	7527	59	4169	35	906	0
Main Street	168	0	632	0	6454	121	1171	271
Mendon	1459	2339	8541	0	8187	0	1678	3149
Blacksmith Fork	502	0	1202	0	1208	0	385	507

For most cases, the algorithm correction resulted in significantly fewer invalid values than the technician correction. For 16 out of 24 time series (most of the temperature, specific conductance, and pH series), the number of invalid points produced by the algorithm correction was less than 100 (out of 200,000+ total points) while the number of invalid points produced by the technician was significantly higher (ranging from 22 to 8541). For five of the time series (primarily dissolved oxygen), the algorithm correction resulted in a higher number of invalid values. For some of these series, the anomaly detection was also less performant (e.g., dissolved oxygen at Franklin Basin, Tony Grove, and Mendon – Fig. 7). These results highlight the need to review anomaly detections and refine settings to improve anomaly detection. Although the corrections classed as valid were within an acceptable range for that time series, the correction may not have approximated observed data patterns, so review of proposed algorithm corrections is necessary.

The overarching benefit of the correction in pyhydroqc is that the algorithm may capture diurnal patterns to suggest values that a technician could not estimate. However, anomalous events need review prior to correction, as do correction suggestions. Adjacent data may be inadequate to generate correction estimates for the full duration of an anomalous event. A more complete workflow could offer correction options for each anomalous event for review and selection by a technician.

4. Conclusions

We developed a new Python package, pyhydroqc, that enables application of rules-based and time series regression techniques coupled with dynamic thresholds as part of a workflow to detect and correct anomalies in aquatic sensor data. Functions to implement the models and supporting steps in the workflow are contained in the Python package and documented within the GitHub repository. Available functions include rules-based anomaly detection, calibration detection and drift correction, model development and estimation, threshold determination, anomaly detection and widening, performance metrics reporting, and model-based correction. Although this workflow advances the automation of sensor data post processing and can be implemented in any Python environment, a Python package and scripts may not be intuitive tools for some technicians. A graphical user interface offering more interactive review could be built on top of the underlying functionality contained in pyhydroqc. Another potential next step is investigating execution of the software on an edge computing device to streamline the process and reduce reliance on centralized systems.

We tested the methods on 24 time series of aquatic data from the Logan River in northern Utah. The case study sites varied in physical characteristics and spanned 5–6 years of high frequency data. Based on our case study, the anomaly detection workflow enabled by pyhydroqc was successful with high detection rates. ARIMA models were most performant, likely due to differences in model structure and development. Rather than using constant thresholds, dynamic thresholds allowed for responsiveness to data variability. Adjusting threshold settings impacts the sensitivity of model detection. We suggest erring on the side of oversensitivity and then reviewing detections. A correction algorithm used blended forecasts and backcasts of local models to make correction estimates that follow data patterns for events of up to several days for some observed variables. These approximations surpass a technician's ability to correct anomalous data, but each corrected event needs review. A rules-based approach was successful in determining calibration gap values and performing linear drift correction with calibration dates as input. Though not completely automated, this work helps to streamline the process of quality control related to sensor drift and calibration. Beyond the case study data, applying the techniques to datasets from other locations and environmental variables outside of the aquatic domain is an important next step.

Manual detection and correction performed by technicians is an extended process that overlaps with other tasks. To perform quality control for 3–6 month durations of a single time series takes multiple days of dedicated effort. In comparison, implementing the complete pyhydroqc workflow for anomaly detection and correction for all variables at a single site for a single year of data takes a few hours to run in the background on a personal computer. A technician will still need to review results; however, we submit that the package and workflow offer significant resource savings.

Throughout this process, the technician was treated as an "oracle" with technician labels dictating algorithm performance. The subjectivity inherent in manual quality control and uneven application of labels by technicians highlight the need for improving consistency in quality control, which is an important driver of automating post processing given that computers are not subjective in their decisions.

As the volume of environmental sensor data continues to increase, so does the need for performing post processing quality control. This work contributes tools and approaches that can be used to streamline and automate the quality control process to reduce the costs of manual quality control; facilitate a post processing workflow that is reproducible, defensible, and consistent; and provide reliable data for analysis and decision making.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research was primarily funded by the United States National Science Foundation under grant number 1931297. Any opinions, findings, and conclusions or recommendations expressed are those of the authors and do not necessarily reflect the views of the National Science Foundation. Additional funding support was provided by the Utah Water Research Laboratory at Utah State University. Ongoing data collection and management in the LRO is supported by funding from the Utah State Legislature administered by the Utah Division of Water Resources. Additional funding for the LRO has been provided by Utah State University, Logan City, and the Cache Water District. We gratefully acknowledge the work of the many technicians and students who have participated in maintaining and operating the LRO instrumentation along with producing the quality-controlled sensor data that were used in testing the software we developed.

Appendix A

Background

Manual post processing by a technician remains the most commonly implemented approach for correcting anomalies in environmental sensor data. Software tools have been developed to assist technicians in performing quality control, wherein anomalies are identified visually or using filters or rules that are implemented based on user-input (Horsburgh et al., 2015; Sheldon, 2008). While initially straightforward to implement, manual post processing is resource-intensive, requires significant expertise, and may be implemented unevenly within and between sensor networks. Additionally, manual approaches may not be reproducible making it difficult to track the provenance of data from raw measurements to quality controlled products. Data driven anomaly detection has the potential to address the deficiencies of manual post processing by streamlining and standardizing the workflow.

Numerous data driven approaches have been documented for anomaly detection (Chandola et al., 2009; Cook et al., 2020; Tan et al., 2019). Basic approaches use rules to test data plausibility - e.g., range and variability checks (Taylor and Loescher, 2013), and even studies with complex workflows initially implement rules-based approaches (e.g., Leigh et al., 2018). Statistical approaches rely on the distribution of data to identify points outside of the expectation (Cook et al., 2020). Regression approaches estimate a value and compare it to the observation (Chandola et al., 2009). Feature based approaches apply numerous variables (or features) within one or more machine learning methods to determine if the data point should be grouped with valid or anomalous points (Talagala et al., 2019). In approaching data driven methods for anomaly detection, important considerations include:

- Data extent: What duration of data are available? Some methods require data partitioned into separate groups for training and testing models.
- Data labels: Do sufficient data exist in which anomalies have been identified by an expert? The availability of labeled data impacts which types of models can be used. Supervised model types require labeled data for training while unsupervised model types do not. For all model types, labeled data enable assessment of performance.
- Data quality: Do sufficient data exist in which anomalies have been corrected? Some methods require 'clean' data that are free from anomalies for training models.
- Variables: What variables are to be considered? Is a single variable/sensor observed or are multiple variables measured? Do sensors at nearby sites provide additional information?
- Anomaly types: What types of anomalies are of particular concern? Can rules-based detection effectively detect some of these cases?
- Online/offline detection: Does detection need to occur in real time online, or is a retrospective, offline approach acceptable?

In the following sections, we provide a brief description of several approaches and methods for detecting and correcting anomalies in environmental sensor data. We also illustrate gaps in the current state of practice for anomaly detection and correction in the quality control process.

A.1 Data Redundancy Approaches

Various types of data redundancy, including sensors, people, and models, are used to detect anomalies in environmental sensor data. The gold standard (World Meteorological Organization, 2008; Mourad and Bertrand-Krajewski, 2002) compares data from multiple sensors, requiring at least three sensors to determine which observation is erroneous. Increased cost, maintenance, power, and data storage requirements challenge observational networks to implement redundant sensors. Furthermore, multiple sensors may all exhibit the vagaries of environmental events, sensor malfunctions, and infrastructure failures, complicating assessment and correction of data quality. To improve the consistency of quality control, Jones et al. (2018) suggest another form of data redundancy in which multiple technicians collaborate to review and correct data. Finally, data redundancy may be achieved by modeling expected values for comparison with sensor measurements. A physically based model could be used; however, model availability and uncertainty are barriers (Moatar et al., 2001). Given the relative simplicity of implementation, ability to scale to large volumes of data, few input requirements, and potential for fast performance, statistical and data driven techniques may be more appropriate. Thus, we examined several classes of data driven techniques to model expected sensor behavior as data redundancy approaches.

A.2 Univariate or Multivariate Approaches

Some predictive time series models are based on data from a single sensor independent of the condition of other co-located sensors or data. Advantages of these univariate methods are that processing can be performed on multiple sensors independently and simultaneously, and gaps or errors in data from one sensor will not impact data from other sensors (Hill and Minsker, 2010). However, anomalies in one sensor stream may correspond to anomalies in a related sensor, so approaches that utilize the information from multiple sensors provide multiple lines of evidence toward anomaly detection (Li et al., 2017). Furthermore, when performing quality control post processing, technicians regularly consult the record of other variables simultaneously recorded at the same site to check for 'internal consistency' (Campbell et al., 2013) and to inform corrective actions. There is no clear best approach, and even the same authors simultaneously promote a univariate detector (Hill and Minsker, 2010) and a multivariate approach (Hill et al., 2009). Either method may yield acceptable results, although Leigh et al. (2018) report poor performance for multivariate time series regression compared to univariate. The data in question will drive whether a univariate method is required or if additional power could be achieved with multiple variables. In our work, we considered both univariate and multivariate approaches and compared the benefits and drawbacks related to the data we examined.

A.3 Spatial Dependency

'External consistency' refers to comparison with data from other locations (Campbell et al., 2013), and some data driven approaches are based on relationships between sites. In particular, spatial dependencies between weather sensors have been used to identify anomalies (Galarus et al., 2012). In another application, data driven models used weighted data from neighboring stream monitoring sites to infill daily mean flow records (Giustarini et al., 2016). One study included data at an upstream site offset by estimated travel time to detect anomalies in aquatic data (Conde, 2011). Spatial methods assume high correlation for a particular variable at sites having similar characteristics, which may not be clearly established for the data of interest. In this work, we focused models on data at a single site of interest so that detection and correction could be applied to sites independently.

A.4 Regression Approaches

Regression models are a class of data driven anomaly detectors for time series that predict the next anticipated value based on previous data (either univariate or multivariate). To detect anomalies with regression, the modeled value is compared to the observed, and a range of acceptability is determined for the residuals such that points outside of that range are classed as anomalous (and vice versa). Constant acceptability thresholds may be based on a user-defined range or determined as a prediction interval based on the model results (Leigh et al., 2018). Thresholds may also be dynamic, varying based on the range of the model residuals (Hundman et al., 2018). For example, in one study (Dereszynski and Dietterich, 2007), the threshold range for an observation varied based on the modeled state of the sensor (i.e., a narrower range when the sensor was classed as "Good" versus "Bad").

Auto-regressive integrated moving average (ARIMA) is a regression technique that uses a combination of past data to forecast the next point. ARIMA has been successfully implemented to predict environmental data and subsequently detect anomalies (Hill and Minsker, 2010; Leigh et al., 2018; Papacharalampous et al., 2019). Another regression technique based on a previous sequence of data is Long Short-Term Memory (LSTM), a class of Artificial Neural Networks (ANNs). Though applications to environmental data anomalies to date are limited, LSTM models have been used to reconstruct time series to detect anomalies in other fields (Hundman et al., 2018; Lindemann et al., 2019; Malhotra et al., 2016; Yin et al., 2020), and other ANN model types have been used for environmental anomaly detection (Hill and Minsker, 2010; Russo et al., 2020). Other algorithms that show promise for time series regression include Prophet, a time series forecasting method developed by Facebook with focus on business applications (Taylor and Letham, 2018), and Hierarchical Temporal Memory (HTM) (Ahmad et al., 2017). Another method that has been implemented for anomaly detection in environmental sensor data is Dynamic Bayesian Networks, which predict values in a time series based on assigned model states corresponding to temporal windows. Studies developed models based on a few previous points (Hill et al., 2009), thousands of previous points (Hill and Minsker, 2006), and multiple past years of data to give an output based on the day of year and hour of day (Dereszynski and Dietterich, 2007). These models assume that temporal states can be definitively assigned as well as consistently applied, and we did not attempt them due to complexity and obscurity of implementation.

Because regression models produce an estimate, they are well-suited for both detection and correction of anomalous data. The time series regression models we investigated were ARIMA, LSTM, and Facebook Prophet. While ARIMA has been commonly attempted for anomaly detection in time series data, other techniques are emergent in this field (e.g., LSTM), and there are few examples comparing multiple regression techniques for aquatic sensor data.

A.5 Feature Based Approaches

Feature based methods comprise another class of anomaly detectors commonly used for discrete data (Tan et al., 2019), which some authors have applied to environmental time series (Leigh et al., 2018; Russo et al., 2020; Talagala et al., 2019). Unlike regression methods, feature based methods do not make a prediction of the observation. Anomalies are detected either based on a supervised model trained to data labels (anomalous or valid) (Russo et al., 2020), or an unsupervised model that determines the likelihood of the point being anomalous based on distance to neighboring points. These methods rely on multiple variables as model input (features), which, in the case of aquatic sensor time series, may correspond to variables measured concurrently by adjacent sensors, past values of the variable of interest, or transformations of the relationships between these variables. Particularly for data with temporal correlation, it is not obvious which features should be selected, and complex feature engineering may be required (Christ et al., 2018). Another challenge is selecting an appropriate data transformation, a preprocessing step (e.g., taking the first derivative of the data) to highlight outlying points (Leigh et al., 2018; Talagala et al., 2019).

Almost any feature based machine learning method may be applied to anomaly detection problems, and approaches described in the literature include principal components analysis, support vector machines (Tran et al., 2019), HDOutliers (Leigh et al., 2018), k-nearest neighbor (Russo et al., 2020; Talagala et al., 2019), clustering (Hill and Minsker, 2010), random forest (Russo et al., 2020), xgboost, and isolated forest (Smolyakov et al., 2019). The success of feature based techniques in detecting anomalies from environmental sensor data is mixed (Hill and Minsker, 2010; Leigh et al., 2018; Russo et al., 2020). As they do not make predictions, feature based approaches are not well-suited to performing corrections. Given that our objectives were to both detect and correct anomalies, we did not pursue feature based approaches in the work reported here.

A.6 Anomaly Types

In most of the studies cited here, the emphasis is on anomalies that are outliers where the value of the variable is outside of expected ranges or rates of change. Detection of gradual bias that may occur due to drift in the sensor or ongoing fouling has not been successfully reported. The models implemented by Dereszynski and Dietterich (2007) identify some biases resulting from abrupt shifts in conditions; however, the authors acknowledge that complex anomalies are outside of the performance of their detector. Conde (2011) was unable to identify labeled anomalies with relatively small variation from the measured baseline. Leigh et al. (2018) intentionally prioritized outliers in development of anomaly detection techniques for aquatic sensors. Given that existing methods have not addressed anomalies caused by drift and fouling, there is significant room for improvement in methods for detecting these types of anomalies. We examined both outliers and more subtle anomaly types in our methods and software implementation.

A.7 Reproducibility

Although effectively implemented for specific case studies in the research realm, none of the techniques described in the cited studies have been packaged as easily accessible software for broad application and dissemination. Without reusable code, the specifics of the algorithms as implemented with environmental data cannot be examined, further tested, or applied to other datasets. Recent work in outlier detection was encapsulated in an R package (Talagala et al., 2019); however, a lack of documentation made it difficult to know how to install the package and apply the methods to our datasets. Provenance of data from raw field observations to quality controlled data products is vitally important yet rarely described in sufficient detail that the process used to arrive at final data products could be repeated (Horsburgh et al., 2015). Applying more automated techniques can help, and reusable software tools can overcome barriers related to understanding and implementing complex algorithms for practical application. Rather than a model calibrated to a specific variable/site combination, practitioners need tools that can be applied to a broad suite of variables and/or monitoring locations documented in a reusable and reproducible way. Thus, we sought to package the tools we developed as open source software that could easily be deployed in a commonly available analytical environment.

A.8 Anomaly Correction

Various techniques and past studies developed functionality for detecting anomalies, but few applied corrective actions, which is an important and time consuming step in quality control post processing. A handful of studies used modeled ARIMA forecasts to directly replace anomalies that were detected by the same ARIMA model, termed 'anomaly detection and mitigation' (ADAM) (Hill and Minsker, 2010; Leigh et al., 2018). However, the objective of ADAM was to improve detection by ensuring that model input data did not include detected anomalies, not to generate a corrected version of the dataset. Furthermore, the success of ADAM was mixed and resulted in high rates of false positives (Leigh et al., 2018). Given the general lack of available methods for automated correction, we explored new approaches for inclusion in the software package we developed.

Appendix B

List of Files and Functions

This appendix provides a listing of each of the Python files in the pyhydroqc package and describes the functionality that each provides. More detailed documentation is found in the GitHub repository and package documentation (see the Software Availability Section).

parameters.py

This file contains assignments of parameters for all steps of the anomaly detection workflow. Parameters are defined specific to each site and observed variable that are referenced in the detect script. LSTM parameters are consistent across sites and variables. ARIMA hyper parameters are specific to each site/variable combination, other parameters are used for rules-based anomaly detection, determining dynamic thresholds, and for widening anomalous events.

anomaly_utilities.py

Contains functions for performing anomaly detection and correction:

- **get_data:** Retrieves and formats data. Retrieval is based on site, observed variable, and year. To pass through subsequent steps, the required format is a Pandas data frame with columns corresponding to datetime (as the index), raw data, corrected data, and data labels (anomalies identified by technicians).
- **anomaly_events:** Widens anomalies and indexes events or groups of anomalous data.
- **assign_cm:** A helper function for resizing anomaly events to the original size for determining metrics.
- **compare_events:** Compares anomaly events detected by an algorithm to events labeled by a technician.
- **metrics:** Determines performance metrics of the detections relative to labeled data.
- **event_metrics:** Determines performance metrics based on number of events rather than the number of data points.
- **print_metrics:** Prints the metrics to the console.
- **group_bools:** Indexes contiguous groups of anomalous and valid data to facilitate correction.
- **xfade:** Uses a cross-fade to blend forecasted and backcasted data over anomaly events for generating data correction.
- **set_dynamic_threshold:** Creates a threshold that varies dynamically based on the model residuals.
- **set_cons_threshold:** Creates a threshold of constant value.
- **detect_anomalies:** Uses model residuals and threshold values to classify anomalous data.
- **aggregate_results:** Combines the detections from multiple models to give a single output of anomaly detections.
- **plt_threshold:** Plots thresholds and model residuals.
- **plt_results:** Plots raw data, model predictions, detected and labeled anomalies.

modeling_utilities.py

Contains functions for building and training models:

- **pdq:** Automatically determines the (p, d, q) hyperparameters of a time series for ARIMA modeling.
- **build_arima_model, LSTM_univar, LSTM_multivar, LSTM_univar_bidir, LSTM_multivar_bidir:** wrappers that call other functions in the file to scale and reshape data (for LSTM models only), create and train a model, and output model predictions and residuals.
- **create_scaler:** Creates a scaler object for scaling and unscaling data.
- **create_training_dataset**, **create_bidir_training_dataset:** Creates a training dataset based on a random selection of points from the dataset. Reshapes data to include the desired time_steps for input to the LSTM model - the number of past data points to examine or past and future points (bidirectional). Ensures that data already identified as anomalous (i.e., by rules-based detection) are not used.
- **create_sequenced_dataset**, **create_bidir_sequenced_dataset:** Reshapes all inputs into sequences that include time_steps for input to the LSTM model - using either only past data points or past and future data points (bidirectional). Used for testing or for applying the model to a full dataset.
- **create_vanilla_model, create_bidir_model:** Helper functions used to create single layer LSTM models.
- **train_model:** Fits the model to training data. Uses a validation subset to monitor for improvements to ensure that training is not too long.

rules_detect.py

Contains functions for rules-based anomaly detection and preprocessing. Depends on anomaly_utilities.py. Functions include:

- **range_check:** Scans for data points outside of user-defined limits and marks the points as anomalous.
- **persistence:** Scans for repeated values in the data and marks them as anomalous if the duration exceeds a user-defined length.
- **group_size:** Determines the maximum length of anomalous groups identified by the previous steps.
- **interpolate:** Corrects data points with linear interpolation, a typical approach for short anomalous events.
- **add_labels:** Enables the addition of anomaly labels (referring to anomalies previously identified by an expert) in the case that labels may have been missed for corrected data that are NaN or have been set to a no data value (e.g., −9999).

calibration.py

Contains functions for identifying and correcting calibration events. Functions include:

- **calib_edge_detect:** Identifies possible calibration event candidates by using edge filtering.
- **calib_persist_detect:** Identifies possible calibration event candidates based on persistence of a user-defined length.
- **calib_overlap:** Identifies possible calibration event candidates by finding concurrent events of multiple sensors from the calib_persist_detect function.
- **find_gap:** Determines a gap value for a calibration event based on the largest data difference within a time window around a datetime.
- **lin_drift_cor:** Performs linear drift correction to address sensor drift given calibration dates and a gap value.

model_workflow.py

Contains functionality to build and train ARIMA and LSTM models, apply the models to make predictions, set thresholds, detect anomalies, widen anomalous events, and determine metrics. Depends on anomaly_utilities.py, modeling_utilities.py, and rules_detect.py. Wrapper function names are: **ARIMA_detect, LSTM_detect_univar,** and **LSTM_detect_multivar**. LSTM model workflows include options for vanilla or bidirectional. Within each wrapper function, the full detection workflow is followed. Options allow for output of plots, summaries, and metrics.

ARIMA_correct.py

Contains functionality to perform corrections and plot results using ARIMA models. Depends on anomaly_utilities.py.

- **ARIMA_group:** Ensures that the valid data surrounding anomalous data points and groups of data points are sufficient forecasting/backcasting.
- **ARIMA_forecast:** Creates predictions of data where anomalies occur.
- **generate_corrections:** The primary function for determining corrections. Passes through data with anomalies and determines corrections using piecewise ARIMA models. Corrections are determined by averaging together (cross fade) both a forecast and a backcast.

Appendix C

Logan River Observatory Input Parameters and Settings

Table C1
Input parameters for each time series. Persistence duration and window size refer to the number of time steps: 20 = 5 h, 30 = 7.5 h, 40 = 10 h, 45 = 11.25 h.

Observed Variable	Parameter	Franklin Basin	Tony Grove	Water Lab	Main Street	Mendon	Blacksmith Fork
Temperature (degrees C)	Maximum range	13	20	18	20	28	28
	Minimum range	−2	−2	−2	−2	−2	−2
	Persistence duration	30	30	30	30	30	30
	Window size	30	30	30	30	30	30
	alpha	1E-04	1E-05	1E-04	1E-05	1E-04	1E-04
	Threshold minimum	0.25	0.4	0.4	0.4	0.4	0.4
	(p, d, q)	(1, 1, 3)	(10, 1, 0)	(0, 1, 5)	(0, 0, 0)	(3, 1, 1)	(1, 1, 0)
Specific Conductance (μS/cm)	Maximum range	380	500	450	2700	800	900
	Minimum range	120	175	200	150	200	200
	Persistence duration	30	30	30	30	30	30
	Window size	30	40	40	40	40	20
	alpha	1E-04	1E-05	1E-04	1E-06	1E-05	1E-02
	Threshold minimum	4	5	5	5	5	4
	(p, d, q)	(10, 1, 3)	(6, 1, 2)	(7, 1, 0)	(1, 1, 5)	(9, 1, 4)	(0, 0, 5)
pH	Maximum range	9.2	9	9.2	9.5	9	9.2
	Minimum range	7.5	8	8	7.5	7.4	7.2
	Persistence duration	45	45	45	45	45	45
	Window size	30	40	40	20	20	30
	alpha	1E-05	1E-05	1E-05	1E-04	1E-04	1E-05
	Threshold minimum	0.02	0.02	0.02	0.03	0.03	0.03
	(p, d, q)	(10, 1, 1)	(8, 1, 4)	(10, 1, 0)	(3, 1, 1)	(0, 1, 2)	(0, 1, 4)
Dissolved Oxygen (mg/L)	Maximum range	13	14	14	15	15	14
	Minimum range	8	7	7	5	3	2
	Persistence duration	45	45	45	45	45	45
	Window size	30	30	30	30	30	30
	alpha	1E-04	1E-04	1E-05	1E-05	1E-03	1E-04
	Threshold minimum	0.15	0.15	0.15	0.25	0.15	0.15
	(p, d, q)	(0, 1, 5)	(10, 1, 0)	(1, 1, 1)	(1, 1, 1)	(10, 1, 3)	(0, 0, 5)

Table C2
LSTM model parameters and settings selected for the LRO case study. Defaults were used for all other settings and parameters not listed here. See Géron (2017) and Keras Development Team (n.d.) for additional details.

Parameter	Function	Setting	Details
Time steps	model.add	5	The number of past data considered as input for prediction. For the LRO data, more time steps (10, 15, 20) biased results toward the mean. Reduced time steps (5) gave greater accuracy and improved computational time.
Units/cells	model.add	128	Number of cells or nodes in the model architecture. There is no rule for finding the perfect number of cells. We chose a high number and used early stopping and dropout to prevent overfitting. For processing purposes, it is generally preferred to have network dimensions in multiples of 32.
Dropout	model.add	0.2	A fraction of cells that are randomly ignored during training. Using dropout improves the model by reducing overfitting, but the number usually matters little. 20% is often used to balance accuracy and overfitting.
Optimizer	model. compile	adam	Algorithm for training. Adam (adaptive movement estimation) is commonly selected for training LSTM models for being computationally efficient, requiring little memory, and handling large amounts of data.
Loss	model. compile	Mean absolute error	The quantity to be minimized during training. Mean absolute error computes the mean of the difference between observations and predictions.
Epochs	model.fit	100	The number of rounds to train the model. We opted for a high number that is truncated by early stopping that ends training when the model is sufficiently fit.
Validation split	model.fit	0.1	Fraction of training data to be used as validation data on which the loss is evaluated at the end of each epoch.
Callbacks	model.fit	Early stopping	Interrupts training when performance on the validation set drops.
Patience	model.fit	6	Number of epochs with no improvement after which training will be stopped.
Shuffle	model.fit	False	Whether to shuffle training data before each epoch. Set to false because the order of training data matters for these data.

Appendix D

Anomaly Detection and Correction Examples

This appendix includes additional examples of anomaly detection and correction performed by the pyhydroqc workflow on LRO case study data. Figure C1 illustrates anomaly detection false positives and true positives. Peaks and troughs in the data were considered anomalies by the model (ARIMA), but only two of them (2017-12-18 and 2017-12-26) were labeled by the technician. It is unclear why certain peaks were labeled by the technician while others were not. Although this example includes several false positives, the algorithm behaved as expected.

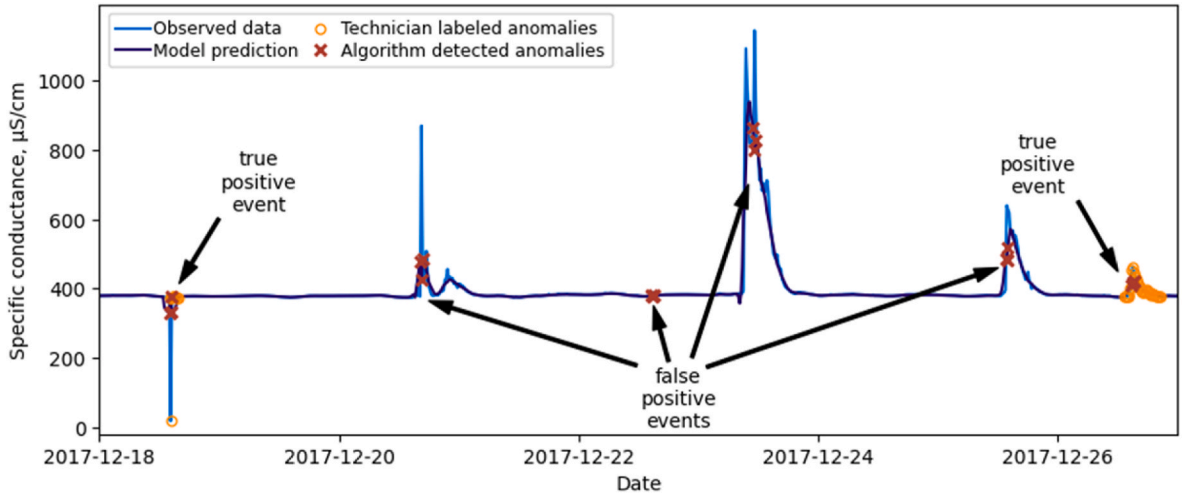


Fig. C1. Examples of anomalies detected using an ARIMA model for specific conductance at Main Street.

In some cases, the apparent success of the model results may be an artifact of both the generalization of detections in the 'compare_events' function and the liberal application of labels by technicians. Some time series contain extensive periods of data labeled as anomalous that correspond to concerns with sensor validity or site conditions (e.g., Figure C2). When comparing events to determine confusion matrix categories, any overlap in model detections results in all points of the anomalous period being identified as true positives. This is an example where large events may bias the

metrics toward true positives if any point in the event is detected or toward false negatives if the event goes undetected (less likely). This particular event contributes to the 13,000+ true positives for this time series (pH at Main Street).

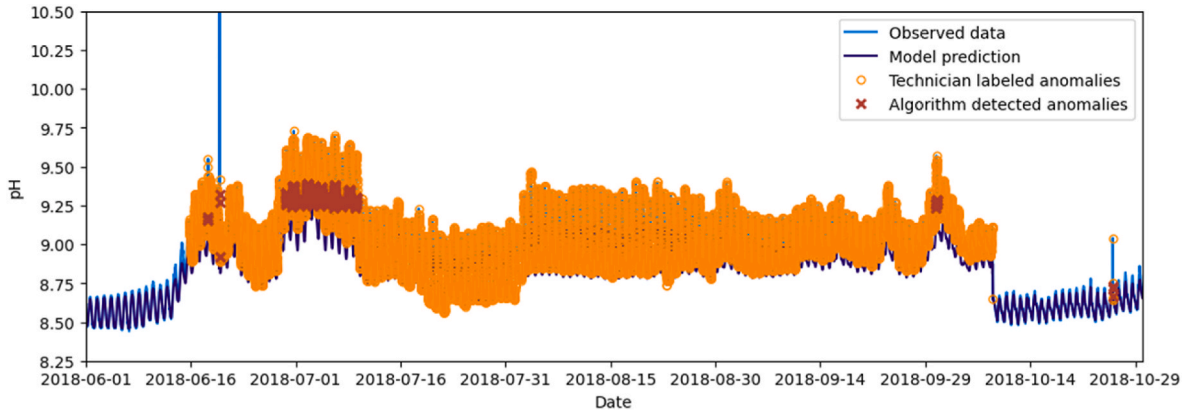


Fig. C2. Examples of anomalies detected using an LSTM multivariate bidirectional model for pH at Main Street for of an extended period of data labeled as a sensor malfunction.

We were interested in whether the models could detect calibration events. For one time series (pH at Main Street), one model type (LSTM multivariate bidirectional) detected approximately 20% of labeled calibration events. We found that the master list of calibrations recorded in the field notes differs from what technicians labeled in the data. Some calibrations recorded in the field notes were not labeled by technicians in the data, and other events labeled by technicians appeared to be calibrations but were not part of the master list derived from the field notes. These discrepancies point to deficiencies in the labeled data. The model predictions are erratic and do not track the observations at most calibration events (Figure C3a), even if the threshold was not sensitive enough to result in detections. In some cases, calibration events were detected as anomalous by the model (Figure C3b), but there was no mechanism to distinguish from other anomalies. These examples illustrate the challenge of using the model-based approach for detecting and correcting calibration events.

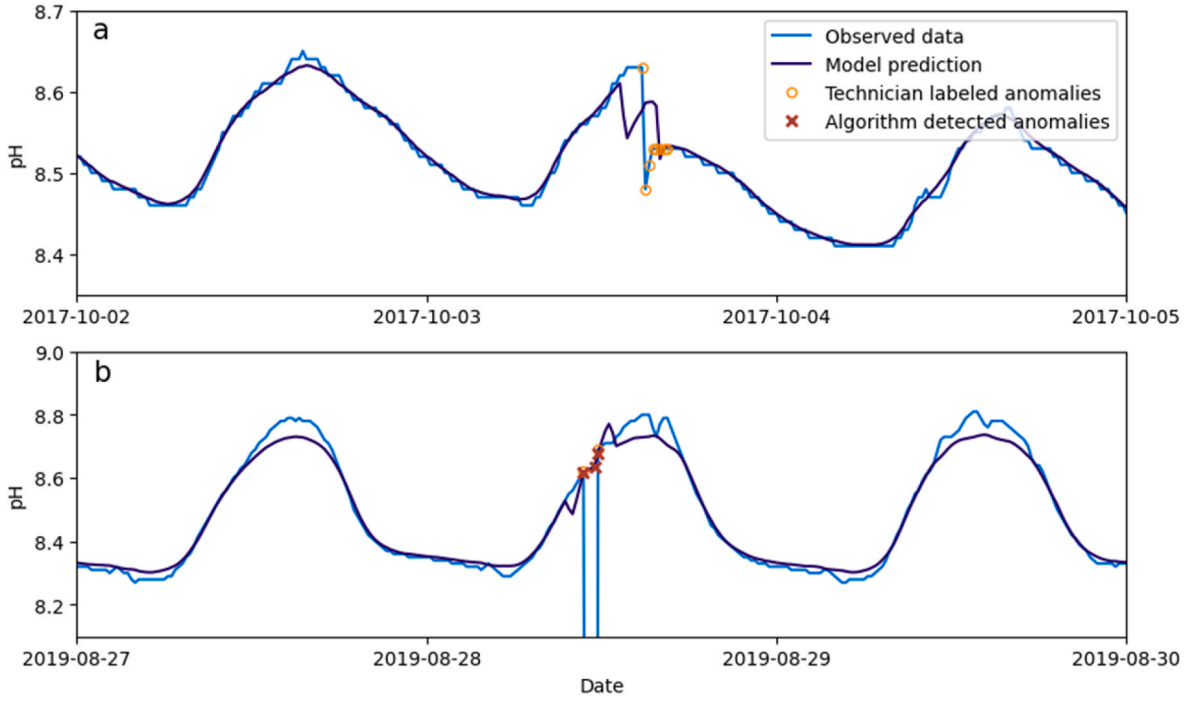


Fig. C3. Examples of anomalies detected using an LSTM multivariate bidirectional model on a pH sensor at Main Street with calibration events.

A direct comparison of results from each model type illustrates model behaviors and associated detections. For specific conductance at Tony Grove, where there was variability in performance between model types (see Section 3.4), the ARIMA and LSTM multivariate vanilla models detected points at the edges of long duration labeled events, improving their performance metrics relative to the other model types. Figure C4 further illustrates differences between model estimates and resulting detections. For the first date range, the estimates of both multivariate models deviate from the original data because they use other variables as input. In the absence of this information, only one univariate model detects an anomaly. In the second date range, models responded to the localized event in distinct ways, and none resulted in a detection. In the third date range, estimates from the multivariate models exhibit spikes around the detections illustrating that information is coming from other variables. It is likely that some of these labeled anomalies correspond to calibration events for which other variables exhibited greater shifts than did specific conductance.

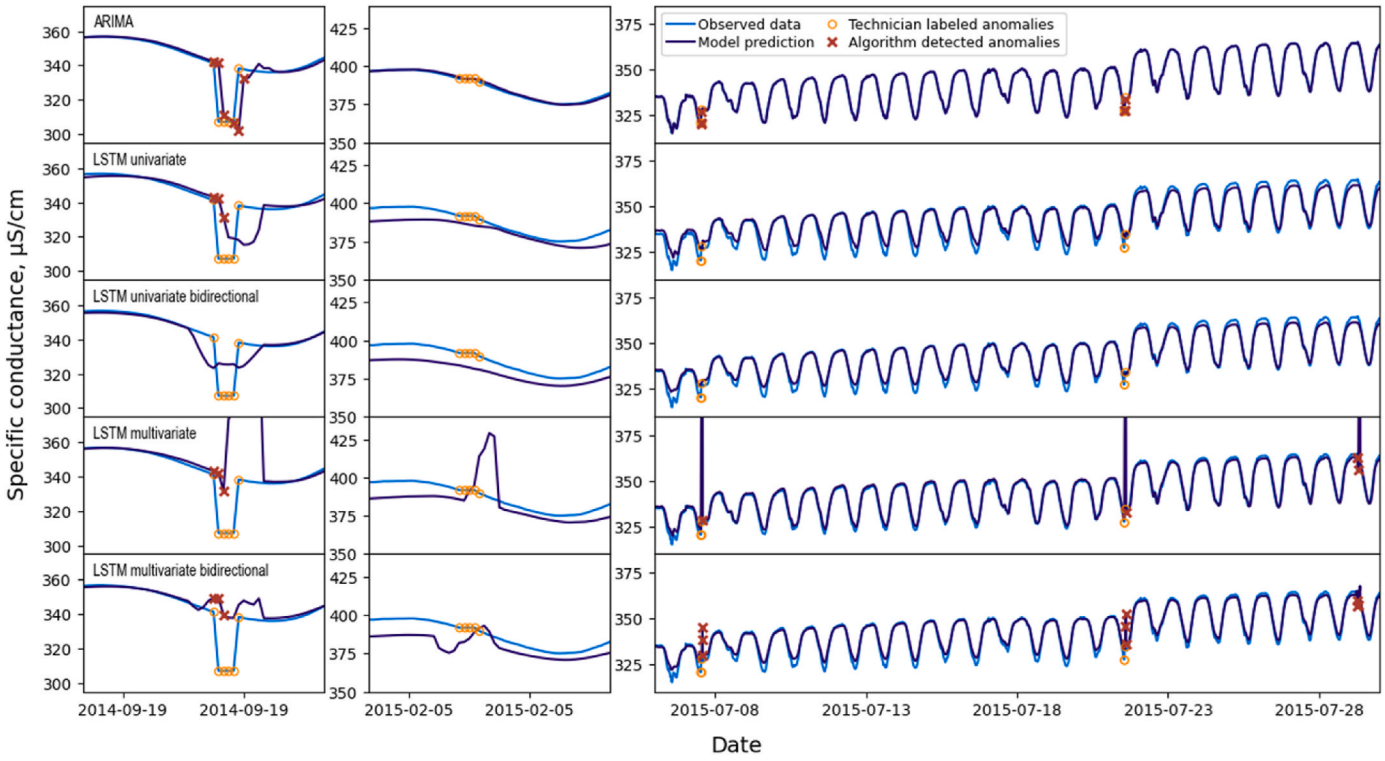


Fig. C4. Examples comparing model estimates and detected anomalies for all model types for specific conductance at Tony Grove.

Although the correction algorithm was capable of capturing diurnal oscillations, in some cases, data patterns did not translate and propagate through the corrections (e.g., Figure C5). Because each correction is based on individual, independent models trained for data immediately prior to and following an anomalous event, the number of data points considered can vary. Even though the adjacent data used for input is limited by the maximum duration parameter, some models may still overgeneralize (i.e., a straight line). Other models may use so little data that a pattern is missed, while still others are focused on a single dominant feature (i.e., an oscillation or a curve). Furthermore, a pattern may be damped over an extended time period. Explicitly incorporating seasonality into development of the ARIMA models may result in more consistent output of oscillations. However, developing seasonal ARIMA models is computationally demanding, and the correction algorithm already requires significant computational resources.

The correction algorithm is directly dependent on identified anomalies. In Figure C5c, an anomalous event (2018-06-19 – 2018-06-20) was detected by the model, but even with widening, the initial abrupt decrease was not labeled anomalous, so it was considered valid data, and it directly influenced the forecast. For the correction algorithm to be effective, anomalies should be reviewed and may need adjustment (e.g., further widening).

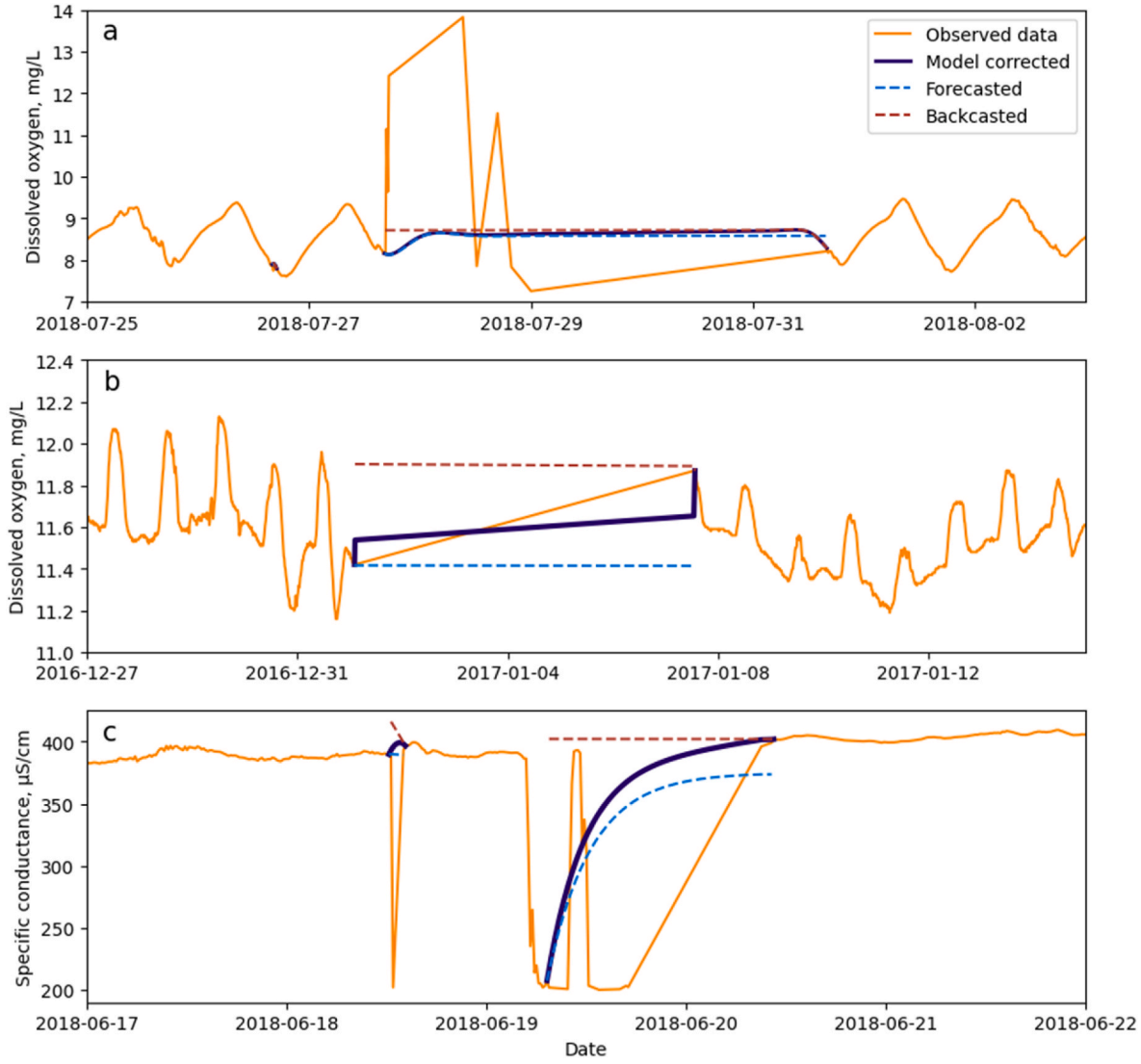


Fig. C5. Examples of problematic algorithm correction. a and b: dissolved oxygen at Tony Grove, c: specific conductance at Mendon.

References

Ahmad, S., Lavin, A., Purdy, S., Agha, Z., 2017. Unsupervised real-time anomaly detection for streaming data. Neurocomputing 262, 134–147. https://doi.org/10.1016/j.neucom.2017.04.070.

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, Wade, M., Boose, E.R., 2013. Quantity is nothing without quality. Bioscience 63, 574–585. https://doi.org/10.1525/bio.2013.63.7.10.

Chandola, V., Banerjee, A., Kumar, V., 2009. Survey of anomaly detection. ACM Comput. Surv. 41, 1–72. https://doi.org/10.1145/1541880.1541882.

Christ, M., Braun, N., Neuffer, J., Kempa-Liehr, A.W., 2018. Time series FeatuRe extraction on basis of scalable hypothesis tests (tsfresh – a Python package). Neurocomputing 307, 72–77. https://doi.org/10.1016/j.neucom.2018.03.067.

Conde, E.F., 2011. Environmental Sensor Anomaly Detection Using Learning Machines. Learning. Utah State University.

Cook, A., Misirli, G., Fan, Z., 2020. Anomaly detection for IoT time-series data: a survey. IEEE Internet Things J. 1–1 https://doi.org/10.1109/jiot.2019.2958185.

Dereszynski, E.W., Dietterich, T.G., 2007. Probabilistic models for anomaly detection in remote sensor data streams. In: Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence (UAI2007), pp. 75–82.

Fiebrich, C.A., Morgan, C.R., McCombs, A.G., Hall, P.K., Mcpherson, R.a., Morgan, Y.R., McCombs, A.G., Hall, P.K., Mcpherson, R.a., 2010. Quality assurance procedures for mesoscale meteorological data. J. Atmos. Ocean. Technol. 27, 1565–1582. https://doi.org/10.1175/2010JTECHA1433.1.

Galarus, D., Angryk, R., Sheppard, J., 2012. Automated weather sensor quality control. Proc. 25th Int. Florida Artif. Intell. Res. Soc. Conf. FLAIRS- 25, 388–393.

Géron, A., 2017. No Hand-On Machine Learning with Scikit-Learn & TensorFlow. O'Reilly.

Gibert, K., Horsburgh, J.S., Athanasiadis, I.N., Holmes, G., 2018. Environmental data science. Environ. Model. Softw. 106, 4–12. https://doi.org/10.1016/j.envsoft.2018.04.005.

Gibert, K., Sànchez-Marrè, M., Izquierdo, J., 2016. A survey on pre-processing techniques: relevant issues in the context of environmental data mining. AI. Commun. 29, 627–663. https://doi.org/10.3233/AIC-160710.

Giustarini, L., Parisot, O., Ghoniem, M., Hostache, R., Trebs, I., Otjacques, B., 2016. A user-driven case-based reasoning tool for infilling missing values in daily mean river flow records. Environ. Model. Softw. 82, 308–320. https://doi.org/10.1016/j.envsoft.2016.04.013.

Greff, K., Srivastava, R.K., Koutnik, J., Steunebrink, B.R., Schmidhuber, J., 2017. LSTM: a search space odyssey. IEEE Transact. Neural Networks Learn. Syst. 28, 2222–2232. https://doi.org/10.1109/TNNLS.2016.2582924.

Gries, C., Henshaw, D., Brown, R.F., Cary, R., Downing, J., Jones, C., Kennedy, A., Laney, C.M., Martin, M., Morse, J., Porter, J., Read, J.S., Rettig, A., Sheldon, W., Strachan, S., Zdravkovic, B., 2014. Sensor and Sensor Data Management Best Practices Released, 201. LTER Databits Spring.

Hart, J.K., Martinez, K., 2006. Environmental Sensor Networks: a revolution in the earth system science? Earth Sci. Rev. 78, 177–191. https://doi.org/10.1016/j.earscirev.2006.05.001.

Hill, D.J., Minsker, B.S., 2010. Anomaly detection in streaming environmental sensor data: a data-driven modeling approach. Environ. Model. Softw. 25, 1014–1022. https://doi.org/10.1016/j.envsoft.2009.08.010.

Hill, D.J., Minsker, B.S., 2006. Automated fault detection for in-situ environmental sensors. In: 7th International Conference on Hydroinformatics.

Hill, D.J., Minsker, B.S., Amir, E., 2009. Real-time Bayesian anomaly detection in streaming environmental data. Water Resour. Res. 45, 1–16. https://doi.org/10.1029/2008WR006956.

Horsburgh, J.S., Tarboton, D.G., Maidment, D.R., Zaslavsky, I., 2008. A relational model for environmental and water resources data. Water Resour. Res. 44 https://doi.org/10.1029/2007wr006392.

Horsburgh, J.S., Reeder, S.L., Jones, A.S., Meline, J., 2015. Open source software for visualization and quality control of continuous hydrologic and water quality sensor data. Environ. Model. Softw. 70, 32–44. https://doi.org/10.1016/j.envsoft.2015.04.002.

Hundman, K., Constantinou, V., Laporte, C., Colwell, I., Soderstrom, T., 2018. Detecting spacecraft anomalies using LSTMs and nonparametric dynamic thresholding. Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min. 387–395. https://doi.org/10.1145/3219819.3219845.

Jones, A.S., 2022. pyhydroqc Sensor Data QC: Single Site Example. HydroShare. https://doi.org/10.4211/hs.92f393cbd06b47c398bdd2bbb86887ac.

Jones, A.S., Aanderud, Z.T., Horsburgh, J.S., Eiriksson, D.P., Dastrup, D., Cox, C., Jones, S.B., Bowling, D.R., Carlisle, J., Carling, G.T., Baker, M.A., 2017. Designing and implementing a network for sensing water quality and hydrology across mountain to urban transitions. J. Am. Water Resour. Assoc. https://doi.org/10.1111/1752-1688.12557.

Jones, A.S., Horsburgh, J.S., Eiriksson, D.P., 2018. Assessing subjectivity in environmental sensor data post processing via a controlled experiment. Ecol. Inf. 46, 86–96. https://doi.org/10.1016/j.ecoinf.2018.05.001.

Jones, A.S., Horsburgh, J.S., Reeder, S.L., Ramírez, M., Caraballo, J., 2015. A data management and publication workflow for a large-scale, heterogeneous sensor network. Environ. Monit. Assess. 187, 348. https://doi.org/10.1007/s10661-015-4594-3.

Jones, A.S., Jones, T.L., Horsburgh, J.S., 2022. Supporting data and tools for "Toward automating post processing of aquatic sensor data". HydroShare. https://doi.org/10.4211/hs.a6ea89ae20354e39b3c9f1228997e27a.

Jones, A.S., Jones, T.L., Horsburgh, J.S., 2022. pyhydroqc v0.0.4. Zenodo. https://doi.org/10.5281/zenodo.6336536.

Keras Development Team. n.d. Keras [WWW Document]. URL. https://keras.io/about/. (Accessed 2 May 2021).

Leigh, C., Alsibai, O., Hyndman, R.J., Kandanaarachchi, S., King, O.C., McGree, J.M., Neelamraju, C., Strauss, J., Talagala, P.D., Turner, R.S., Mengersen, K., Peterson, E. E., 2018. A framework for automated anomaly detection in high frequency wate r-quality data from in situ sensors. Sci. Total Environ. 664, 885–898. https://doi.org/10.1016/j.scitotenv.2019.02.085.

Li, J., Pedrycz, W., Jamal, I., 2017. Multivariate time series anomaly detection: a framework of Hidden Markov Models. Appl. Soft Comput. J. 60, 229–240. https://doi.org/10.1016/j.asoc.2017.06.035.

Lindemann, B., Fesenmayr, F., Jazdi, N., Weyrich, M., 2019. Anomaly detection in discrete manufacturing using self-learning approaches. Procedia CIRP 79, 313–318. https://doi.org/10.1016/j.procir.2019.02.073.

Malhotra, P., Ramakrishnan, A., Anand, G., Vig, L., Agarwal, P., Shroff, G., 2016. LSTM-based Encoder-Decoder for Multi-Sensor Anomaly Detection.

Moatar, F., Miquel, J., Poirel, A., 2001. A quality-control method for physical and chemical monitoring data. Application to dissolved oxygen levels in the river Loire (France). J. Hydrol. 252, 25–36. https://doi.org/10.1016/S0022-1694(01)00439-5.

Mourad, M., Bertrand-Krajewski, J.-L., 2002. A method for automatic validation of long time series of data in urban hydrology. Water Sci. Technol. 45 (263). LP – 270.

Neilson, B.T., Tennant, H., Strong, P.A., Horsburgh, J.S., 2021. Detailed streamflow data for understanding hydrologic responses in the Logan River Observatory. Hydrologic Proces. 35 (8), e14268 https://doi.org/10.1002/hyp.14268.

Pandas Development Team, 2008. Pandas Documentation [WWW Document]. URL. https://pandas.pydata.org/docs/. (Accessed 2 May 2021).

Papacharalampous, G., Tyralis, H., Koutsoyiannis, D., 2019. Comparison of Stochastic and Machine Learning Methods for Multi-step Ahead Forecasting of Hydrological Processes, Stochastic Environmental Research and Risk Assessment. Springer Berlin Heidelberg. https://doi.org/10.1007/s00477-018-1638-6.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E., 2011. Scikit-learn: machine learning in Python. J. Mach. Learn. Res. 12, 2825–2830.

Pellerin, B.A., Stauffer, B.A., Young, D.A., Sullivan, D.J., Bricker, S.B., Walbridge, M.R., Clyde, G.A., Shaw, D.M., 2016. Emerging tools for continuous nutrient monitoring networks: sensors advancing science and water resources protection. JAWRA J. Am. Water Resour. Assoc. 20460, 1–16. https://doi.org/10.1111/1752-1688.12386.

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., Skeffington, R.A., Rozemeijer, J.C., Aubert, A.H., Rinke, K., Jomaa, S., 2016. Sensors in the stream: the high-frequency wave of the present. Environ. Sci. Technol. https://doi.org/10.1021/acs.est.6b02155.

Russo, S., Lürig, M., Hao, W., Matthews, B., Villez, K., 2020. Active learning for anomaly detection in environmental data. Environ. Model. Softw. https://doi.org/10.1016/j.envsoft.2020.104869.

Seabold, S., Perktold, J., 2010. statsmodels: econometric and statistical modeling with python. In: Proceedings of the 9th Python in Science Conference.

Sheldon, W.M., 2008. Dynamic, rule-based quality control framework for real-time sensor data. In: Gries, C., Jones, M.B. (Eds.), Proceedings of the Environmental Information Management Conference, pp. 145–150. Albuquerque, NM.

Smith, T.G., 2017. Pmdarima: ARIMA Estimators for Python.

Smolyakov, D., Sviridenko, N., Ishimtsev, V., Burikov, E., Burnaev, E., 2019. Learning ensembles of anomaly detectors on synthetic data. Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics) 11555 LNCS. https://doi.org/10.1007/978-3-030-22808-8_30, 292–306.

Talagala, P.D., Hyndman, R.J., Leigh, C., Mengersen, K., Smith-Miles, K., 2019. A feature-based procedure for detecting technical outliers in water-quality data from in situ sensors. Water Resour. Res. 55, 8547–8568. https://doi.org/10.1029/2019WR024906.

Tan, P.-N., Steinback, M., Karpatne, A., Kumar, V., 2019. Introduction to Data Mining, second. Pearson, New York.

Taylor, J.R., Loescher, H.L., 2013. Automated quality control methods for sensor data: a novel observatory approach. Biogeosciences 9, 18175–18210. https://doi.org/10.5194/bg-10-4957-2013.

Taylor, S.J., Letham, B., 2018. Forecasting at scale. Am. Statistician 72, 37–45. https://doi.org/10.1080/00031305.2017.1380080.

Tran, L., Fan, L., Shahabi, C., 2019. Outlier Detection in Non-stationary Data Streams 25–36. https://doi.org/10.1145/3335783.3335788.

Wagner, R.J., Boulger, R.W., Oblinger, C.J., Smith, B.A., 2006. Guidelines and Standard Procedures for Continuous Water-Quality Monitors: Station Operation, Record Computation, and Data Reporting. In: U.S. Geological Survey Techniques and Methods, 1-D3.

White, D.L., Sharp, J.L., Eidson, G., Parab, S., Ali, F., Esswein, S., 2010. Real-Time Quality Control (QC) Processing, Notification, and Visualization Services, Suporting Data Management of the Intelligent River, in: Proceedings of the 2010 South Carolina Water Resources Conference, p. 4.

World Meteorological Organization, 2008. Guide to Meteorological Instruments and Methods of Observation. WMO-No 8, Geneva.

Yin, C., Zhang, S., Wang, J., Xiong, N.N., 2020. Anomaly detection based on convolutional recurrent autoencoder for IoT time series. IEEE Trans. Syst. Man, Cybern. Syst. 1–11 https://doi.org/10.1109/tsmc.2020.2968516.