SOUND: Sanity Checking of Pipelines for Uncertain and Sparse Data Series

Hermann Stolte[1], Iftach Sadeh[2], Elisa Pueschel[3], Avigdor Gal[4], Matthias Weidlich[1]
[1]*Humboldt-Universität zu Berlin* [2]*DESY Zeuthen* [3]*Ruhr University Bochum* [4]*Technion - Israel Inst. of Techn.*
{stolther,weidlima}@hu-berlin.de iftach.sadeh@desy.de Elisa.Pueschel@astro.rub.de avigal@ie.technion.ac.il

***Abstract*—The analysis of data series forms the basis of decision-making in various domains, so that it is essential to ensure data validity. Yet, current solutions for sanity checking of processing pipelines, such as GX, TFDV, Pandera or Deequ, fall short in accounting for data quality issues. In particular, irregular cadences, sparsity and value uncertainty limit the applicability of sanity checking and pose risks of false conclusions.**

In this paper, we present SOUND to enable sanity checking of pipelines in the presence of typical quality issues in data series. In particular, SOUND evaluates a set of sanity constraints that formalize validity expectations on the data, while incorporating data quality issues, i.e., uncertainty of individual data points and sparsity in a whole data series. To this end, it defines a statistical framework for constraint checking that is based on adaptive resampling and Bayesian hypothesis testing, minimizing computational costs while ensuring accurate results. If a constraint violation has been identified, SOUND also includes drill-down strategies to guide users in the identification of the root cause of the violation. We demonstrate the feasibility and utility of SOUND by applying it for pipelines developed in the domains of smart grid monitoring and astrophysics.

I. INTRODUCTION

Data series are generated in domains ranging from healthcare [1] and finance [2] to IoT [3] and Science [4], and drive decision-making processes. In these domains, the analysis of data series, i.e., traditional timeseries [5] and sequences of multivariate data points [6], is important to understand the evolution of a system over time. To this end, a data processing pipeline is designed, which comprises several operators that are sequentially applied over data series. To achieve scalability, such pipelines are implemented using big data frameworks [7], [8] or scientific workflow engines [9].

Similar to writing unit tests to ensure the reliability of a software system [10], it is essential to ensure the validity of processed data [11]–[14]. Changes in the inputs to a pipeline, or changes to the pipeline itself, may compromise data consistency and, hence, yield implausible results. To achieve reliable analysis, therefore, sanity constraints that formulate integrity expectations shall be evaluated on primary, intermediate, and final data products, in order to ensure their trustworthiness.

The evaluation of sanity constraints, however, remains challenging when faced with data quality issues. Real-world data series show temporal gaps and sparseness, e.g., due to costly data acquisition [15]. Also, the values in data series are often uncertain [16], due to the use of point estimates, limited precision of sensors, measurement errors, or data products being available only in aggregated or anonymized form.

Current solutions to ensure data validity in data processing pipelines, such as GreatExpectations [11], TFDV [12], or Deequ [13], largely neglect such data quality issues. While they support the specification of sanity constraints for data in various computational environments, the evaluation of these constraints assumes discrete, accurate, and complete data series.

Neglecting quality aspects in the assessment of data integrity, however, yields results that are biased. This is problematic for two reasons. First, false positives induce a severe manual overhead: Complex analysis pipelines need to be debugged, wasting countless hours of rerunning the pipeline and communicating with experts on a particular processing step or input data file. Second, false negatives hamper the validity of the result. Constraint violations are not noticed, which leads to potentially wrong conclusions and flawed decision-making.

In this paper, we present SOUND, a framework for sanity checking of processing pipelines for uncertain and sparse data series. By incorporating data quality aspects, it flags constraint violations more accurately, and clearly marks regions in the series, for which conclusions on data validity cannot be made. SOUND makes the following contributions:

1) *Constraint model.* To capture validity requirements on data series in a processing pipeline, we introduce a model for sanity constraints. It includes a taxonomy to systematically capture validity expectations within a certain context.
2) *Sanity checking for uncertain and sparse data.* We provide a statistical framework for the evaluation of sanity constraints in the presence of data quality issues. To cater for uncertainty of data values and sparsity of the series, we rely on bootstrapping and confirmatory hypothesis testing.
3) *Violation analysis.* We present means to support the identification of a root-cause of a constraint violation. We outline how changes in the validation result are traced back to data quality issues or changes in the handled data.

We evaluate SOUND using two real-world applications. We show that sanity checking with SOUND in Flink incurs only a modest computational overhead. Moreover, we illustrate that our framework indeed enables effective constraint evaluation under data quality issues of varying severity, and that the root-cause of a violation can be isolated successfully.

Below, we elaborate on our motivation (§II), before formalizing the problem setting (§III). We then introduce our approach to sanity checking (§IV), and elaborate on the analysis of violations (§V). We close with evaluation results (§VI), a discussion of related work (§VII), and conclusions (§VIII).

II. Motivation

Sanity constraints provide a general means to ensure the validity of processed data [12]–[14]. They formulate integrity assumptions on the processed data, which may stem from domain knowledge, such as physical laws and scientific models, and the semantic context of the data, including fairness aspects, compliance rules, or real-world quantities. While the manual specification of sanity constraints induces a certain overhead, this effort pays off through the early and accurate detection of validity issues, thereby saving debugging and development efforts. Once trustworthy data is available, various types of techniques to detect common structure and regularities in data may also help users in constraint definition. For instance, techniques for data profiling may determine common value ranges [17], correlation analysis and warping may reveal dependencies between different data series [18], and pattern detection may hint at recurring behavior over time [19].

The importance of sanity checking of processing pipelines is highlighted by various frameworks to define such constraints, which emerged in recent years, such as GX [11], TFDV [12], and Deequ [13]. However, these frameworks assume discrete, accurate, and complete data series, and neglect the impact of data quality issues, which creates misleading results.

As an example, consider Fig. 1, which shows an uncertain data series of variable rate (upper panel). The values are assigned standard deviations for upwards and downwards uncertainty, which is a common model, e.g., in astrophysics [20] and earth observation [21]. Here, a sanity constraint is defined that checks whether the values of four time windows stay below a certain threshold, which is indicated by the dashed line. Ignoring the uncertainty and rate differences as part of a naive approach, yields misleading and noisy results (middle panel). In the second time window, it wrongly decides on constraint violation, even though the measurement uncertainty suggests that the value is likely within the range. In the third window, it indicates a satisfied constraint (two of three values are within the range), whereas the measurement uncertainty suggests that this is unlikely. In the fourth window, a constraint violation is indicated, if though there is insufficient evidence for such a conclusion, due to having a single point with large uncertainty intervals above and below the threshold. Using the approach presented in this work, SOUND, we avoid these misleading results, as shown in Fig. 1 (lower panel).

The above properties of application scenarios for sanity checking are generic and materialize in a wide range of domains. We illustrate several of these domains in the remainder:

Smart Grid Analytics. Load forecasting in smart grids is based on series of measurements of energy consumption [22]–[24]. The respective pipelines include forecasting models, integrate contextual data, and detect abnormal user behavior. Examples for sanity constraints define plausible value ranges (load forecasts must stay below a threshold), limit value changes over time (accumulated work needs to increase monotonically), and enforce correlations (aggregates at different levels of the monitoring hierarchy need to be show consistent trends). Yet,

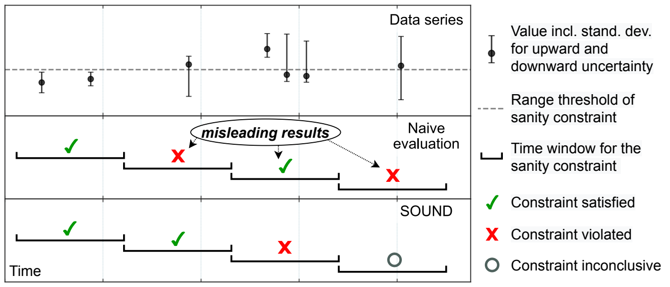
Data series
Naive evaluation
misleading results
SOUND
Time
Value incl. stand. dev. for upward and downward uncertainty
Range threshold of sanity constraint
Time window for the sanity constraint
Constraint satisfied
Constraint violated
Constraint inconclusive

Figure 1: Sanity check evaluation on a sparse and uncertain data series (top) using a naive approach (middle) and using SOUND (bottom).

constraint evaluation has to handle data quality issues: Readings of accumulated work are reported only in coarse-grained units such as kWh, while momentary load measurements are more fine-grained. Also, data series are sparse, since measurement devices show periods of unavailability [23].

Data Management in Astronomy. In data-driven astronomy, data series representing gamma-ray [25], x-ray [26], and optical [27] measurements are processed to detect anomalies and guide observation schedules for pointed telescopes. The respective pipelines involve multiple levels of data cleaning and normalization, training of background models, and outlier classification. Here, sanity constraints capture, for instance, the correlation of sliding averages computed over series with a different temporal granularity. However, data quality issues are omnipresent and need to be incorporated in constraint evaluation: Instruments are inherently noisy and measurement values come with an associated uncertainty (from statistical models) and the limited scope of observation per instrument leads to data series of varying cadences.

Urban Transportation. Systems for traffic management incorporate data series that capture traffic flow at junctions as well as positional information of buses and light rails [28]. The data is processed with normalization stages, and models to predict crowdedness and delay patterns. Sanity constraints relate to the inertia with which traffic flow can be expected to change over time, and the value ranges of the respective predictions. However, the respective data series show data quality issues: Flow measurements obtained with induction loops are inherently uncertain and the non-availability of positional information due to limited technical coverage of specific areas in a city shall be considered in constraint evaluation.

Finance Monitoring. In fraud detection, spending patterns and dependencies between financial transactions are assessed to identify fraudulent behavior [29], [30]. To this end, series of transaction events are processed in pipelines that include models for classifying transactions into spending classes and for learning the geo-spatial context of regular behavior, and deviations thereof. Sanity constraints capture invariants such as correlations between aggregates of spending volumes over time, but also over different classes. Yet, again, data quality issues impact constraint evaluation, as the series inherently show varying cadence as well as uncertainty stemming from the various classifiers and predictors included in the pipeline.

Table I: Overview of basic notions and notations.

Notation	Description
$\mathcal{D} = \mathbb{Q} \times V \times V \times V$	Domain of data points
$p = (t, v, \sigma_\uparrow, \sigma_\downarrow)$	Data point: Timestamp t, value v, stand. dev. $\sigma_\uparrow, \sigma_\downarrow$
$s = \langle p_1, p_2, \ldots, p_n \rangle$	Data series: Sequence of data points
$P = (S, E)$	Pipeline: DAG with series $S \subseteq \mathcal{D}^*$ as nodes; edges $E \subseteq S \times \mathcal{O} \times S$ denote data transformations based on operators from domain $\mathcal{O}$
${}^\bullet s$	Predecessors of data series s in the pipeline

III. Problem Setting

Below, we present a model for our problem context (§III-A), and characterize the problem of sanity checking (§III-B).

A. Data Series and Pipelines

Requirements. The above scenarios highlight two common types of quality issues that are often observed in practice:

Value uncertainty. Data points are often uncertain, with the reasons ranging from limited precision of physical instruments, to upstream data aggregation, or anonymization techniques. While our work is agnostic to the precise choice of an uncertainty model, below, we adopt a widespread model based on normal distributions [31]: A data point consists of a value and the information on two normal distributions that characterize the upward and downward uncertainty, each captured by the distributions' standard deviation.

Data sparsity. Series of data points are often irregularly scattered over time, e.g., due to high costs of data acquisition or the lack of continuous observability of a phenomenon. As such, the data model shall include explicit temporal information to capture the density of values over time.

Data points and data series. In the light of the above requirements, we rely on a model of a *data point* $p = (t, v, \sigma_\uparrow, \sigma_\downarrow)$ of domain $\mathcal{D} = \mathbb{Q} \times V \times V \times V$ that includes a timestamp t, a value v, and the standard deviations $\sigma_\uparrow$ and $\sigma_\downarrow$ with which the value shall be interpreted. As a shorthand, we use $p.t$, $p.v$, $p.\sigma_\uparrow$, and $p.\sigma_\downarrow$ to refer to the components of a data point p. Also, we may consider scenarios, in which timestamps and the standard deviations are not set as they are not available or not relevant for the analysis. A sequence $s = \langle p_1, p_2, \ldots, p_n \rangle \in \mathcal{D}^*$ of n data points is a *data series*. As a shorthand, we use $s.t$, $s.v$, $s.\sigma_\uparrow$, and $s.\sigma_\downarrow$ to refer to the sequences of the respective components of all data points in s. An overview of the notations used in the remainder is given in Table I.

To illustrate the model, consider the smart grid scenario from above with plug measurements as a data series:

t	1.0	2.0	4.0	8.0	9.0	10.0
v	1.0	3.0	2.0	4.0	8.5	6.0
$\sigma_\uparrow$	2.1	0.4	0.6	0.4	2.2	1.3
$\sigma_\downarrow$	1.6	1.8	1.1	0.2	1.6	1.1

The series contains six measurements. Value uncertainty is captured by two normal distributions with means given as the data point's value, and a standard deviation. Timestamps enable us to model data sparsity, e.g., there are no measurements between timestamps 4.0 and 8.0.

Pipelines. To capture the processing of data series, we rely on a graph-based model. A *pipeline* $P = (S, E)$ is a DAG with the nodes $S \subseteq \mathcal{D}^*$ being a set of data series and the edges $E \subseteq S \times \mathcal{O} \times S$ denoting their transformation using operators from domain $\mathcal{O}$. An edge $(s, o, s') \in E$ denotes that data series s' was derived from data series s through operator o. Operators are generally considered as some user-defined function (UDF), for which the exact semantics are not known. As a short-hand, for a data series $s \in S$, we write ${}^\bullet s = \{s' \in S \mid (s', o, s) \in E\}$ for the data series from which s was derived.

B. Problem Characterization

To achieve reliable analysis in the presence of changes in the data series or the pipelines defined over them, validity expectations on the data are formulated as sanity constraints. An evaluation of these constraints then increases the trustworthiness of the pipeline's results. Any realization of such sanity checking, however, shall address the following requirements:

Expressive modelling of data expectations: Users shall be enabled to express their validity expectations as sanity constraints on properties of the processed data. Such constraints may consider individual data points or parts of data series; they may refer to a particular context in which the data occurs; and they may link the data in terms of its provenance.

Robust assessment of expectations: Sanity constraints need to be evaluated in the light of value uncertainty and data sparsity, in order to avoid any bias or even entirely wrong results.

Effective analysis of violations: When validity expectations are not met, users shall be supported in the exploration of potential root-causes for the respective constraint violation. Here, a major concern is the separation of violations that are likely to be caused by data quality issues from those that go along with changes in the processed data.

IV. Sanity Checking

In this section, we first elaborate on the formulation of validity expectations as sanity constraints (§IV-A). Then, we show how the evaluation of constraints is phrased as a statistical test to enable sanity checking in the presence of data quality issues (§IV-B). Finally, we present a collection of constraint templates for processing pipelines over data series (§IV-C).

A. Constraint Definition

Constraint Taxonomy. To capture validity expectations on the processed data, we introduce a taxonomy of sanity constraints in Fig. 2. It includes the following dimensions:

Granularity refers to the selection of data points to which a constraint is applied. *Point-wise* constraints refer to individual points, whereas *window-based* constraints consider all points of a data series, i.e., a *global* window, or only a part of it that is selected based on the data points' *time* or *index*.

Orderedness specifies if a constraint is defined for a *sequence* of data points, either derived based on their *time* or their *index* in a data series; or if the constraint is defined for a *set* of data points, independent of their ordering.

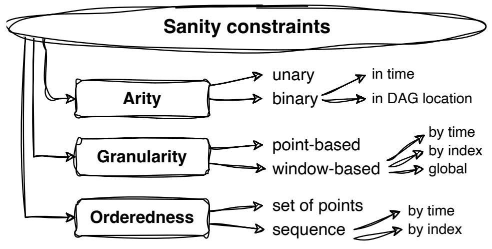

Figure 2: Taxonomy of sanity constraints.

Arity outlines over how many data inputs a constraint is defined, whether it is *unary*, defining some absolute condition over a single data input; or *binary*, defining a relative condition that compares two data inputs.

Constraint Formalization. We formally capture sanity constraints as a function that maps data to a Boolean value, i.e., true $\top$, if the constraint is satisfied; or false $\bot$, if it is violated. The function is defined over sequences of data values, thereby implicitly capturing also point-based constraints.

Definition 1. *A* sanity constraint *is a function* $\phi^k : (V^*)^k \rightarrow \{\top, \bot\}$ *that assigns a Boolean value to* k *sequences of data values.*

Pipeline Integration. To express sanity constraints within the context of a pipeline, a constraint needs to be mapped to a concrete data series. Moreover, with multiple long data series being processed in a pipeline, a constraint can often only be meaningfully expressed for some temporal context, such as a time window of a fixed size. To this end, we define a *sanity check* $\lambda = (\phi^k, s^k, \psi)$ with:

1) ϕ^k: a constraint function,
2) $s^k \in S^k$: k-tuple of data series from pipeline $P = (S, E)$,
3) $\psi : (S)^k \rightarrow ((D^*)^k)^*$: a windowing function mapping a k-tuple of data series to a sequence of k subsequences of the respective data series, e.g. by a sliding time window.

The semantics of a constraint check are defined as follows. The windowing function is applied to the selected data series, which yields a sequence of k tuples of windows:

$$\psi(s^k) \mapsto \langle w_1^k, \ldots, w_n^k \rangle.$$

For point-based constraints, each window comprises k data series of length one, i.e., each window has a single data point.

Next, we consider a naive approach for constraint evaluation that does not account for data sparsity or value uncertainty. It applies the constraint function directly to the data values selected by the windowing function, which results in a Boolean outcome at each sequence index. That is, with

$$(w_{(i,1)}, \ldots, w_{(i,k)}) = \psi(s^k)(i)$$

as the k-tuple of windows (each itself being a data series) at sequence index i of a k-tuple of data series, the constraint evaluation corresponds to

$$\phi((w_{(i,1)}.v, \ldots, w_{(i,k)}.v)) \in \{\top, \bot\}.$$

As demonstrated already in Fig. 1, such a naive approach yields unreliable and potentially misleading results.

Table II: Overview of SOUND notions and notations.

Notation	Description
ϕ^k	Sanity constraint function, assigning a Boolean value to k series of data values
$\lambda = (\phi^k, s^k, \psi)$	Sanity check, consisting of a constraint, k data series, and a windowing function
$\gamma(\phi^k, w_i^k, c, N)$	Sanity check evaluation, assigning an outcome to a k-valued window and a constraint with parameters c, N

Constraint Complexity. While the above definition allows for any constraint arity, in the remainder, we limit ourselves to unary and binary constraints, i.e., $k \in \{1, 2\}$, as this suffices to capture a wide range of validity expectations. In any case, the model covers all types of constraints of the taxonomy in Fig. 2. A constraint may assess only an individual data point (i.e., be applied to a single index of a single data series) or consider subsequences of the series; it may consider the points as a set or leverage their ordering; and it can be required to hold for all data points or adopt a statistical view.

Constraint Evaluation Time. Sanity constraints according to the above definition are applied over the data series S of a pipeline $P = (S, E)$. However, an important consideration is *when* a constraint is evaluated in terms of the progress of pipeline execution. To give timely feedback about constraint violations, the evaluation is performed as soon as the data is available and in parallel to the nominal data processing. For the densest coverage, a constraint is evaluated for every index (i.e. data point for point-based constraints or window for windowed constraints). This ensures high sensitivity in the checking, while we later show empirically that it results only in a modest performance overhead.

B. Constraint Evaluation under Uncertainty

To cater for data quality issues, we propose a robust constraint evaluation algorithm. It comprises a resampling function to capture the implicit variability of data series under the influence of data quality issues, and a dynamic decision rule to derive reliable outcomes with low computational cost. The approach is formalized in Alg. 1 and denoted as:

$$r_i = \gamma(\phi^k, w_i^k, c, N), \quad r_i \in \{\top, \bot, \dashv\}.$$

The evaluation of the constraint function ϕ^k is done per k-valued window $w_i^k = \psi^k(s^k)(i)$ independently. It takes two parameters, a credibility level c, and a max sample size N. The parameters are described below, before discussing the full algorithm. Based on c and N, the constraint evaluation assigns an outcome r_i to each w_i^k, which marks the check as either satisfied $\top$, violated $\bot$, or inconclusive $\dashv$.

Credibility level c: The credibility level, comparable to a significance level in traditional hypothesis testing, defines of the degree of belief in a sanity check outcome that is required to conclude it. Specifically in SOUND, it represents the minimum probability of an evaluation outcome given the variability from data quality issues, before defining a sanity check outcome as satisfied or violated. A reasonable default value is $c = 0.95$.

Algorithm 1: Constraint Evaluation γ.

```
input  : constraint function $\phi$, k-tuple of windows $w_i^k$,
         credibility level $c$, maximum sample size $N$
output : satisfaction ($\top$), violation ($\perp$) or inconclusive result ($\dashv$)
1  count_satisfied $\leftarrow 0$;
2  $\alpha \leftarrow 1, \beta \leftarrow 1$
3  for i_sample in range(1, $N$) do // For each sample iteration
       // Draw random sample to capture uncertainty and
          evaluate constraint
4      sample $\leftarrow$ resample($w^k$);
5      is_satisfied $\leftarrow \phi$(sample);
       // Update the posterior distribution
6      count_satisfied $\leftarrow$ count_satisfied + is_satisfied;
7      $\alpha_{\text{post}} \leftarrow \alpha$ + count_satisfied;
8      $\beta_{\text{post}} \leftarrow \beta$ + i_sample − count_satisfied;
       // Calculate current posterior credible interval
9      [lower, upper] $\leftarrow$ BetaDistribution($\alpha_{\text{post}}, \beta_{\text{post}}, c$);
       // Decision rule based on credible interval
10     if lower > 0.5 then return $\top$;        // Constr. satisfied
11     else if upper < 0.5 then return $\perp$;  // Constr. violated
       // Continue if no decision was made yet
12 return $\dashv$ ;                              // Inconclusive Outcome
```

Maximum sample size N: Resampling in SOUND is motivated by the need to capture data variability. However, the max sample size N limits the computational effort in inconclusive cases. If N is reached before the check is concluded to be satisfied or violated, the outcome is inconclusive $\dashv$. The value of N is set based on resource constraints.

To evaluate a sanity check, first, variables are initialized (lines 1-2 in Alg. 1). Then, until the max number of samples is reached, the k-valued window is resampled (line 4). The constraint function is evaluated on the resampled window (line 5). Based on the outcome, the belief in the outcome is updated, modeled by a Bayesian binomial test (lines 6-9). Finally, a decision rule is applied to terminate the evaluation as soon as a result materializes, or the max sample size is reached (lines 10-12).

Resampling. SOUND includes the `resample` method (line 4) not to improve performance, but to account for the inherent variability in data series caused by data sparsity and value uncertainty. Specifically, we employ techniques based on Monte Carlo Simulation [32] and bootstrapping [33]. The specific approach depends on the granularity and orderedness of the constraint. We differentiate the following three cases.

Point-based checks: For the simplest case with k data points, each sample is created by randomly adding either an upward or downward uncertainty value to the data point. The uncertainty value is sampled from the corresponding uncertainty distribution. If there is no uncertainty, the resampling instead yields the unaltered value of the data point.

Window-based set checks: When resampling k data series independent of their ordering, data sparsity needs to be handled additionally to value uncertainty. A data series with very few data points may be uninformative with respect to the expectation expressed in the constraint function. Then, the samples of the data series should reflect the uncertainty from the low data size. To address this, statistical bootstrapping [33] is used to estimate statistics of the original data series from several samples obtained through random sampling of data points with replacement. Here, SOUND operates under the assumption of bootstrapping: Data points in each w_i are i.i.d, without specific distributional assumptions. For each sample obtained from bootstrapping, the value uncertainty is incorporated by adding a random sample of the upward or downward uncertainty to the point's value.

Window-based sequence checks: In contrast to the previous case, the ordering of data points within each of the k data series matters. Standard bootstrapping does not maintain the ordering of data points and is therefore unsuitable for sequence checks. However, bootstrapping can be lifted to sequence data [34]. In SOUND, we employ block-bootstrap [35]: Instead of sampling from all data points independently, each of the k data series is first divided into subsequent blocks of size b. The sampling with replacement is then carried out on blocks, such that sequential dependencies within blocks are maintained. The block size b is a parameter that, in practice, can be set automatically based on the length of a data series as $b = \sqrt{n}$. While block-bootstrap may alter some sequential patterns, it can be expected to maintain patterns in the lengths that are most relevant for window-based checks. We later discuss how to handle potential false positives from edge cases. To maintain the association across indices of k data series, it is crucial to sample all data series using the same random indices, such that the data series remain aligned.

Bayesian Binomial Test. We model the evaluation of a sanity check using a Bayesian binomial test (lines 7-8) , assuming that both satisfactions and violations are possible. Mathematically, we use an uninformative (flat) prior represented by a Beta distribution with parameters $\alpha = 1$ and $\beta = 1$, which is uniform over $[0, 1]$ and indicates complete uncertainty. The Beta distribution is defined as follows:

$$\text{Beta}(x;\alpha,\beta) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha,\beta)}, \quad B(\alpha,\beta) = \int_0^1 x^{\alpha-1}(1-x)^{\beta-1}dx.$$

Although we start with an uninformative prior, this approach allows for the inclusion of prior knowledge in the evaluation by adjusting the α and β parameters of the Beta distribution. The posterior distribution is also a Beta distribution. After observing n satisfied and m violated constraint evaluations, it is given by:

$$\text{Posterior} = \text{Beta}(\alpha + m, \beta + n).$$

This posterior distribution represents the updated belief about the constraint evaluation and forms the basis for concluding an outcome of sanity check evaluation in a decision rule. Specifically, based on the credible interval of the posterior distribution and a neutral probability threshold at $t = 0.5$, we make the following decision after each resampling step: If the lower bound is above t, we conclude a violation. If the upper bound is below t, we conclude the check is satisfied. Otherwise, the evaluation remains inconclusive (lines 9-12).

Complexity. Evaluating a sanity check (Alg. 1) has the following complexity: Let N be the max sample size (line 3), R be the cost of resampling k data series (line 4), and C the cost of evaluating the constraint function (line 5). Then, the total cost lies in $\mathcal{O}(N \cdot R \cdot C)$. In practice, the actual number of required samples depends on the impact of data quality and the credibility level c. We analyze this empirically in §VI.

C. Constraint Templates

Our approach (§IV-B) is applicable to any definition of a constraint that follows our above model (§IV-A). To illustrate the range of validity expectations that can be formulated and checked, we now elaborate on common types of constraints.

Numeric Ranges. A data series may be expected to respect a numerical range. For instance, an expectation of non-negativity may stem from the series representing distance metrics or occurrence counts. In our taxonomy, this is an *unary*, *point-wise* constraint over an *unordered* data series. Also, when normalizing a data series, the expectation may be that a large fraction of data points falls into the unit interval, which would be a *unary*, *windowed*, and *set-based* constraint.

Monotonic Trends. A data series may be expected to be monotonously increasing or decreasing, e.g. as it contains accumulated data over time or describes the depletion of a resource. Such constraints are *unary*, *window-based*, *discrete*, and require a *sequence* data series.

Linear Correlations. Two unrelated data series may be expected to *not* be correlated, as if they were, this would hint at some issue in the pipeline. Two related data series, in turn, may be expected to be correlated. Either way, a sanity constraint is based on a correlation measure (e.g., Pearsson) and an upper/lower bound on the correlation coefficient, which is a *binary*, *window-based*, and *index-ordered sequence* constraint.

Explained Variances. If a pipeline processes predictions and ground-truth values in data series, it may be expected that the predictions are plausible. While this can be modelled in various ways, a common assessment for regression models is to compare the residuals against the inherent variation of the ground truth series. This can be expressed in a constraint function based on the coefficient of determination, i.e., R^2, which measures the explained variance of a mapping function of an independent variable and dependent variable. A sanity constraint checking that the explained variance between two series is in a given range is *binary*, *window-based* and requires *index-ordered sequence* series.

Equal Distributions. A common integrity expectation is that a set of data series follows a similar distribution. Examples include series of measurements performed by two redundant sensors or the load over time for different nodes ingested through a load balancing system. There exist a plethora of similarity scores for empirical distributions, such as the Kullback-Leibler divergence or Kolmogorov Smirnov test statistic. To express the respective expectation, a check tests that a distance metric is below or above a threshold, which is a *binary*, *windowed*, and *set-based* constraint.

V. Violation Analysis

When a sanity check is violated, an explanation is required that clarifies the violations' root-cause and its implications. Pinpointing exact root-causes, however, is difficult in practice. The search space is large and cannot be explored exhaustively. Similar to a failing software test, the identification of root-causes is a manual process, requiring expert knowledge.

In Sound, we assist users in the identification of violation root-causes by (i) gathering evidence to confirm or exclude potential explanations, and (ii) narrowing down the search space of root-causes in a pipeline. These techniques build upon generic concepts and, therefore, are independent of a specific constraint definition and applicable to any user-defined formulation of validity assumptions.

Below, we first describe our general approach to violation analysis (§V-A). Then, we outline potential explanations and their relevance assessment (§V-B). Finally, we describe an algorithm to guide the analysis of potential root-causes (§V-C).

A. Change Points

We analyze violations of a sanity check based on change points over time. That is, the transition of a sanity check from being satisfied to being violated, or vice versa, provides an opportunity for a direct comparison of related positive and negative data points. In context of a sanity check λ, a change point is a position in the series of constraint evaluation results:

Definition 2. *A change point is an index i of a constraint evaluation results $r_i = \gamma(w_i^k, \phi, c, N)$, for which the condition $(r_i = \top \wedge r_{i-1} = \bot) \vee (r_i = \bot \wedge r_{i-1} = \top)$ holds.*

We denote the pair of subsequent windows located directly at a change point as $(w_\top^k, w_\bot^k)$, with $w_\top^k$ and $w_\bot^k$ being the windows that are evaluated positively and negatively, respectively. To find an explanation of a violation change point, the ordering of the windows (i.e. whether it changes from satisfied to violated or vice versa) does not matter.

B. Potential Explanations

Our analysis of constraint violations is based on a set of possible explanations, see Table III. The root-cause of a change in the check outcome may lie in a difference between the respective data inputs $w_\bot^k$ to $w_\top^k$, or the sanity check λ itself. In the former case, shifts in data distributions may result in changes in the data values or the data quality. We differentiate between scenarios where explanations lie in the data values (**E1**), or in a change of data quality. Specifically, $w_\bot^k$ may have a higher data sparsity (**E2**), lower data sparsity (**E3**), higher value uncertainty (**E4**) or lower value uncertainty (**E5**). Regarding the sanity check $\lambda = (\phi^k, s^k, \psi)$ itself, the constraint function ϕ^k and the s^k data series are identical for both $w_\bot^k$ to $w_\top^k$. Moreover, we consider ϕ^k to be deterministic (which applies to all examples above). Hence, the only element potentially causing a change in the sanity check outcome is the evaluation function ψ (Alg. 1). Specifically, it may cause a difference in the evaluation outcome, if the contained `resample` method alters the sequence structure within $w_\bot^k$ (**E6**).

Table III: Overview of SOUND root-cause candidates.

Name	Description	Linked root-cause	Counterfactual explanation
E1	Difference in Data Values	Data values v	*case-specific, see §V-C*
E2	High Data Sparsity	Temporal sparsity (t)	If $w_\bot$ were less sparse, the check would not be violated.
E3	Low Data Sparsity	Temporal sparsity (t)	If $w_\bot$ were more sparse, the check would not be violated.
E4	High Value Uncertainty	Uncertainty σ	If $w_\bot$ had lower value uncertainty, the check would not be violated.
E5	Low Value Uncertainty	Uncertainty σ	If $w_\bot$ had higher value uncertainty, the check would not be violated.
E6	Resampling False Positive	`resample` (Alg. 1)	If `resample` left the sequence structure of $w_\bot$ unaltered, the check would not be violated.

Next, we show how to systematically examine potential explanations around a violation change point, to either confirm or exclude them. Here, we assume that one of the potential explanations materializes as the main root-cause. Moreover, the assessment is carried out for each of the k input windows separately. If an explanation applies to one of the k inputs, it is valid in the context of the entire check. Hence, we use $w_\top, w_\bot$ as shorthand for one of the k input windows.

E1: Difference in Data Values. A difference in the data values is the most difficult root-cause to identify. Typically, there exist many differences between the values of $w^k_\bot$ and $w^k_\top$, while their relevance for the violation depends on the constraint semantics. It can, however, remain as the only possible explanation:

$$\text{E1} \Leftrightarrow \neg(\text{E2} \vee \text{E3} \vee \text{E4} \vee \text{E5} \vee \text{E6}). \tag{1}$$

If value differences are expected to yield a violation of a sanity check, the root-cause is traced back to data series that are upstream in the processing pipeline. To this end, we introduce a heuristic search for relevant upstream changes in §V-C.

E2: High Data Sparsity. When the data sparsity of $w_\bot$is high, there may have been no violation, if the data was more dense. In that case, the violation's root-cause is not a change in the actual population distribution, but lies in $w_\bot$ being a small, unrepresentative sample of a population distribution. To diagnose E2, the first criterion is that the violated window is sparser than the neighbouring satisfied one, $|w_\bot| < |w_\top|$. Second, we test the aforementioned case directly by increasing the size of $w_\bot$. We carry out an inverse what-if analysis to determine the applicability of E2 to a change point with confidence. Precisely, we test whether the sanity check would fail for $w'_\top$, a version of $w_\top$ with increased sparsity to the level of $|w_\bot|$ by means of random downsampling. Then, E2 can be determined as root-cause based on the following condition:

$$\text{E2} \Leftrightarrow (|w_\bot| < |w_\top|) \wedge (\gamma(\phi, w'_\top, c, N) = \bot).$$

E3: Low Data Sparsity. In analogy to E2, with low data sparsity in $w_\bot$, it is possible that there would have been no violation, if the sparsity was as high as in $w_\top$. In this case, a change in the sanity check outcome is associated with a transition from sparse data (check satisfaction) to more dense data (check violation). We diagnose this root-cause based on two criteria, similar to E2. First, the data has to be more dense when the check is violated. Second, a what-if analysis needs to show that the check would have been satisfied if the sparsity of $w_\bot$ were increased to the level of $w_\top$ through random downsampling:

$$\text{E3} \Leftrightarrow (|w_\bot| > |w_\top|) \wedge (\gamma(\phi, w'_\bot, c, N) = \top).$$

E4: High Value Uncertainty. If the value uncertainty is high and a sanity check is violated, the check may have not been violated under less uncertainty. To identify high value uncertainty as a root-cause, we first assess whether the *relative* value (upward or downward) uncertainty has increased:

$$(\delta^\uparrow_\bot > \delta^\uparrow_\top) \vee (\delta^\downarrow_\bot > \delta^\downarrow_\top) \quad \text{with} \quad \delta^{\uparrow/\downarrow}_{\top/\bot} = \frac{1}{|w_{\top/\bot}|} \sum \frac{w_{\top/\bot}.\sigma_{\uparrow/\downarrow}}{w_{\top/\bot}.v}.$$

Further, we test whether the check result on $w_\bot$ becomes satisfied under downscaling of the uncertainty by the relative difference in the mean relative uncertainties of $w_\bot$ and $w_\top$:

$$\gamma(\phi, w', c, N) = \top \text{ with } w' = w_\bot \text{ and } w'.\sigma^{\uparrow/\downarrow} = w_\bot.\sigma^{\uparrow/\downarrow} \cdot \frac{\delta^{\uparrow/\downarrow}_\top}{\delta^{\uparrow/\downarrow}_\bot}.$$

Combined, E4 is identified as a violation's root-cause, if the following condition holds:

$$\text{E4} \Leftrightarrow (\delta_\bot > \delta_\top) \;\wedge\; (\gamma(\phi, w', c, N) = \top)$$
$$\text{with} \quad \delta_{\top/\bot} = \frac{1}{|w_{\top/\bot}|} \sum \frac{w_{\top/\bot}.\sigma^\uparrow + w_{\top/\bot}.\sigma^\downarrow}{2w_{\top/\bot}.v}.$$

E5: Low Value Uncertainty. With a low value uncertainty at a sanity check violation, it may be that the check would not have been violated if there value uncertainty was higher. Intuitively, there might be a relevant difference in the data that is only detectable when the uncertainty is low enough. To detect such cases, we perform a test analogous to E4. Specifically, it checks if the check on $w_\bot$ becomes satisfied if the uncertainty is upscaled by the relative difference in the mean relative value uncertainties between $w^k_\bot$ and $w_\top$, which leads to the following condition for explanation E5:

$$\text{E5} \Leftrightarrow (\delta_\bot < \delta_\top) \wedge (\gamma(\phi, w', c, N) = \top)$$

E6: Resampling False Positive. In the evaluation of sequence constraints, a spurious violation (i.e. false positive) may occur, if the sequence structure is altered by the block-bootstrap resampling in a way that affects the outcome. If this is the case, the constraint function is true when evaluated on each resampling block b_i within $w_\bot$ individually. We cater for this case with the following condition:

$$\text{E6} \Leftrightarrow \forall b_i \; (\phi(b_i) = \top).$$

C. Change in Data Values

When a constraint violation can only be explained by a change in the data values, a relevant difference must be present in one or more of the k data series evaluated by the sanity check. Yet, the difference's root-cause may be a change in the primary input series to the pipeline, or a change in the intermediate data series produced by upstream operators.

To guide the manual drill-down, we analyse relevant data series for differences. In addition to the k series targeted by the check, we also inspect upstream data series, when relevant by their provenance in the pipeline DAG and the temporal context of a change point. Ultimately, SOUND produces an annotation of the pipeline DAG that limits the search space of the required manual analysis. Next, we first define a notion of a change in data values and then describe the approach for producing the annotated pipeline DAG.

Change Constraint. The criteria of what renders a data difference relevant to a violation change point depends on the semantics of the constraint and position of the data series under investigation. While this motivates flexibility in defining the method for assessing data changes, per-default, we perform a non-parametric statistical test for empirical distribution equality, namely the 2-sample Kolmogorov-Smirnov test [36], denoted as `ks_test_2samp`:

$$\phi^2_{\text{change}}(w_1, w_2) : (\texttt{ks\_test\_2samp}(w_1, w_2).\texttt{p\_value} < \alpha).$$

Here, the significance level α is set to match the evaluation's credibility level, to set a min certainty, i.e., $\alpha \leftarrow (1 - c)$.

Upstream Pipeline Annotation. Based on a data change constraint ϕ^2_{change}, we search for changes in each of the k data series of a check λ and their respective upstream data series, i.e. in the context of pipeline DAG P. The approach is detailed in Alg. 2 and consists of the following general steps:

1) For each of the k change point windows $(w_\perp, w_\top)$, the change constraint is evaluated. The respective data series is annotated, if the evaluation found a change (lines 2-4).
2) For every upstream data series of $u \in {}^\bullet s$, the windows matching those of the change points windows are determined (lines 6-7). The change constraint is evaluated on the window pair and the upstream series is annotated, if a change is found (lines 8-9).

The resulting annotation R, which is a set of data series in P, limits the search space for the violation's root-cause. Any data series or operator that is upstream of $s \in R$ in P is excluded.

Complexity. For each change point in the violation outcomes, the analysis of potential explanations has the following complexity. Let C be the cost of evaluating the constraint function, and let A be the cost of altering the data quality of k series (see E2-E5). The cost of assessing explanations is $\mathcal{O}(C \cdot A)$. The cost of detecting upstream change points (Alg. 2) is $\mathcal{O}(U \cdot T)$, with U being the number of upstream series of all k series targeted by a sanity check λ (lines 2, 5), and T being the cost of evaluating a change constraint ϕ^2_{change} (line 3).

Algorithm 2: Upstream data change point detection.

```
input  : check $\lambda$, pipeline $P = (S, E)$, change point $(w_\perp, w_\top)$,
         upstream change constraint $\phi^2_{\text{upstream}}$
output : set of local and upstream series with present changes
1  $R \leftarrow \emptyset$;
   // For each of the $k$ local series at change point
2  for $s, w_\top, w_\perp$ in zip($s^k, w^k_\top, w^k_\perp$) do
       // Assess difference in local series
3      has_change $\leftarrow \phi^2_{\text{upstream}}(w_\top, w_\perp)$;
       // Mark change in upstream series if present
4      if has_change $= \top$ then $R \leftarrow R \cup \{s\}$;
5      for $u$ in ${}^\bullet s$ do              // For each upstream series
           // Select time ranges of change point in
              upstream series
6          $u_\perp \leftarrow u[u.t \in \text{minmax}(w_\perp.t)]$ ;
7          $u_\top \leftarrow u[u.t \in \text{minmax}(w_\top.t)]$;
           // Assess difference in upstream series
8          has_change $\leftarrow \phi^2_{\text{upstream}}(u_\top, u_\perp)$;
           // Mark change in upstream series if present
9          if has_change $= \top$ then $R \leftarrow R \cup \{u\}$;
10 return $R$
```

VI. EXPERIMENTAL EVALUATION

In the following, we evaluate how SOUND catches implausibilities in realistic pipelines with modest overheads. Moreover, we show the influence of data quality and the parameters c and N on the cost and the confidence in the sanity check evaluation. Finally, we evaluate the automated checks that provide the user with guidance in the drill-down of potential root causes.

A. Evaluation Setup

Hardware/Software. Generally, SOUND can be applied in batch or stream processing contexts. We implemented it, along with the applications, in Apache Flink 1.14.0. The experiments are carried out on a MacBook Pro (2021) with a 10-core M1 Pro processor and 16GB of memory on macOS Ventura (13.6). Experiments run for at least 90 seconds and are repeated 5 times. Unless noted otherwise, plots show the resulting average and 95% confidence interval (shaded region or error bar). Flink applications are executed with 4 parallel worker slots and an operator parallelism of 4. Our evaluation setup is based on that of Erebus [37] and all artifacts are publicly available [38].

Pipelines. We use two applications with real-world data, for which Fig. 3 shows the pipeline DAGs including sanity checks.

Smart Grid Analytics (S) The dataset stems from the DEBS Grand Challenge 2014 [23] and the Flink pipeline is taken from the scenario SGA in [37]. It is an analytics pipeline to detect faulty plugs in a smart grid environment. It computes a minute average of the household load and compares it to the corresponding plug loads at the start of each minute. A load comparison forms the basis for alerts to the user.

Astrophysics (A) The dataset is real-world, public data of over 40 astronomical sources observed from the *Fermi* gamma-ray telescope [39]. The pipeline captures anomaly detection, which first filters the incoming data and then detects short-term anomalies in comparison to a smoothed local baseline.

Sanity checks. Our evaluation covers a wide range of sanity constraints, see Table IV. They include constraints of varying granularity (i.e. point-based and windowed), arity (i.e. unary and binary) and orderedness (set and sequence). In terms of complexity, the checks range from simple cases of unary numerical range to more complex cases like an expectation of correlation in two different series of the pipeline DAG.

Table IV: Sanity checks and their classification in the taxonomy, as implemented in the smart grid (top) and astrophysics (bottom) pipelines.

Name	Description	Arity	Granularity	Orderedness	$\phi(x,(y))$
S-1	load in plausible range	unary	point-wise	-	$a \le x \le b$
S-2	monotonous increase in work	unary	windowed in tuples	sequence	$x_i < x_{i+1}$
S-3	plug count $\geq$ household count	binary in DAG location	windowed in time	set	$\lvert x \rvert \geq \lvert y \rvert$
S-4	usage $>$ 0.5 in alerts	unary	point-wise	-	$x > 0.5$
S-5	max delta in household usage	unary	windowed in time	set	$(\max(x) - \min(x)) < a$
A-1	flux in plausible range	unary	point-wise	-	$a \le x \le b$
A-2	input pipeline did not freeze	unary	windowed in tuples	set	$\mathrm{std}(x) \neq 0$
A-3	lower delta on average	binary in DAG location	windowed in time	sequence	$(x_i - x_{i-1}) < (y_i - y_{i-1})$
A-4	has correlation	binary in DAG location	windowed in time	sequence	$\mathrm{corr}(x,y) > 0.2$

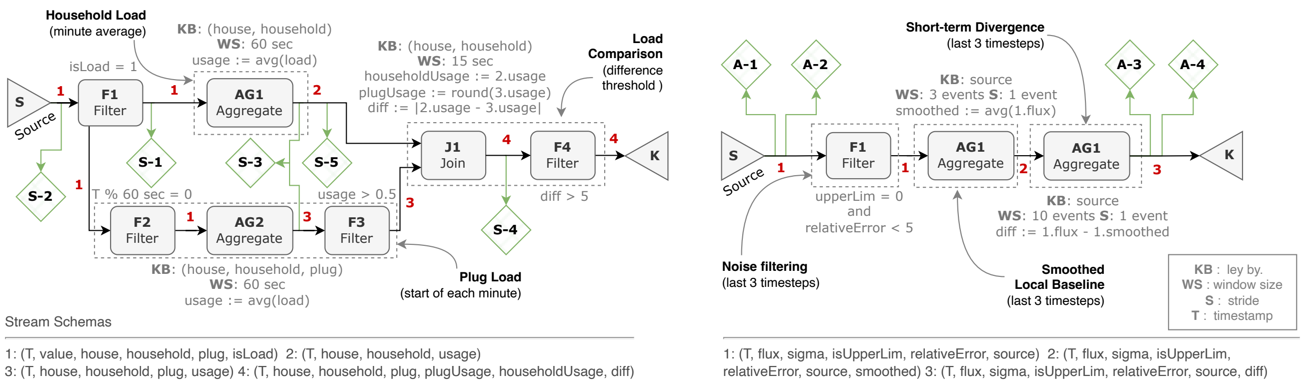

Figure 3: Pipelines for Smart Grid Analytics (left) and Astrophysics (right) along with the corresponding sanity checks.

Baselines. We evaluate SOUND against three distinct baselines:

BASE_NOM The nominal, uninstrumented data processing pipelines, used to assess any performance overhead.

BASE_CHECK A simplistic validation approach ignoring data quality issues. It is equivalent to employing existing techniques, such as TFDV [12] or Deequ [13], without accounting for the impact of data qualiy issues. Specifically, the outcome of sanity check λ is considered identical to the outcome of the associated constraint function ϕ^k.

BASE_VA A baseline approach to the analysis of violation change points, based on existing provenance techniques [37]. Data quality is ignored, s.t. a potential root cause is identified as a change in the local data values (E1) and traced back to upstream series. Change constraints ϕ^2_{change} are evaluated proactively and their results are propagated through the pipeline to the corresponding sanity check.

Evaluation metrics. Performance overhead is captured in terms of throughput (data points processed per second) and latency (delay in seconds between the creation of a data point and the ingestion of all required data points). To analyze the impact of data quality issues on sanity check outcomes, we measure and compare the accuracy of evaluation outcomes from BASE_CHECK and SOUND. Moreover, as part of a sensitivity analysis, we capture the probability distribution of constraint violation along with the concluded sanity check outcome. In the violation analysis, for each explanation E1-6 from SOUND, we report the corresponding number of explained change points. To quantify the explanation quality of SOUND, we also measure the false positive rate of explanations obtained from BASE_VA.

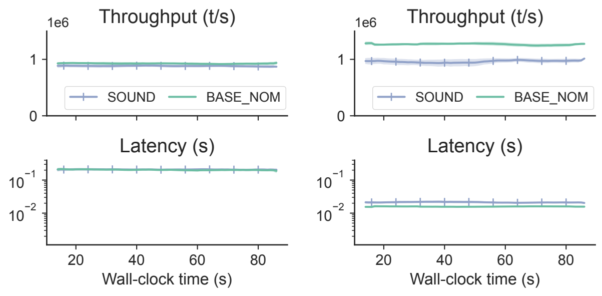

Figure 4: Performance overhead of SOUND measured in throughput and latency for the original and instrumented version of the smart grid application (left) and astrophysics application (right).

B. Performance Overhead

As sanity checks are additional computations that need to be carried out on top of a nominal pipeline, we evaluate the performance overhead of SOUND. Overall, our results show that sanity checking is feasible with modest overheads.

Costs compared to nominal pipeline. An assessment of the overhead in terms of latency and throughput once sanity checks are included is shown in Fig. 4. For the smart grid scenario (left) and the astrophysics scenario (right), the latency and throughput are recorded over wall-clock time, after a warm-up period of 15% of the experiment's duration. SOUND is configured with $c = 0.95$ and $N = 100$. For the smart grid scenario, the overhead is minimal. The throughput decreases to 95% of BASE_NOM, while the latency is on par, i.e., 0.2041 seconds for BASE_NOM and 0.2089 seconds for SOUND. For the astrophysics case, the throughput with SOUND is 76% of BASE_NOM; latency increases from 0.016 to 0.021 seconds.

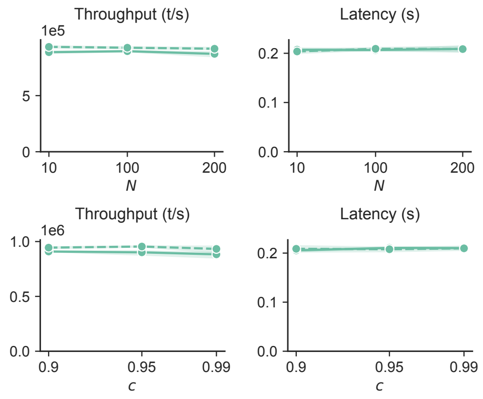

Figure 5: Smart grid scenario: Overhead as a function of the maximum number of samples and credible interval. The dashed line is BASE_NOM and the solid line SOUND.

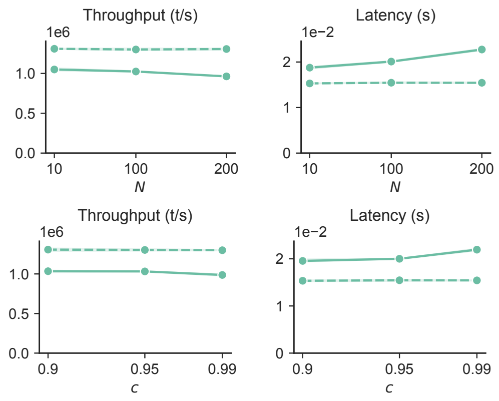

Figure 6: Astrophysics scenario: Overhead as a function of the maximum number of samples and credible interval. The dashed line is BASE_NOM and the solid line SOUND.

For both applications, the overhead is constant over time, hence SOUND and the application pipelines are in a stable state. The overhead for the smart grid scenario is smaller in comparison to the one in the astrophysics scenario. This is expected: First, data quality issues are more pronounced in the astrophysics case, leading to more samples being required to evaluate sanity checks. Second, the constraint functions in the astrophysics application are computationally more expensive. Finally, the smart grid pipeline is more complex, resulting in lower relative overhead of SOUND.

Overhead and framework parameters. The maximum number of samples N and the credible interval c control a trade-off between confidence and computational cost. In the following, we evaluate their effect on the performance overhead. The overhead in throughput and latency for both scenarios is shown in Fig. 5 and Fig. 6. With varying c, the maximum sample sized N is fixed to 100, and with varying N, the credible interval c is fixed to 0.95. A general expectation is that with increasing N and c, the throughput decreases and the latency grows. When varying $N \in [10, 200]$ and $c \in [0.9, 0.99]$, in the smart grid case, the changes in latency and throughput are negligible. In the astrophysics case, the effect is visible more clearly. Here, the mean latency increases from 0.0196 to 0.0219 seconds and the mean throughput decreases from 1.007M, to 947k tuples per second, when increasing $c = 0.90$ to $c = 0.99$. When changing $N = 10$ to $N = 200$, the mean latency increases from 0.0188 to 0.0227 seconds, and the mean throughput decreases from 1.05m to 962k tuples per second.

Table V: Scenario A: Outcomes of BASE_CHECK.

	A-1	A-2	A-3	A-4	Combined
Satisfied Outcome Acc.	0.961	0.071	0.628	1.0	0.634
Violated Outcome Acc.	0.993	-	0.107	0.001	0.161
Inconcl. Outcome Ratio	40.1%	0.0%	24.0%	0.4%	27.8%

C. Effectiveness of Sanity Checking

Next, we demonstrate that data quality issues may indeed have a large impact on the outcome of the evaluation of sanity checks. To this end, we compare the evaluation outcomes of SOUND to a naive approach. The latter ignores value uncertainty and data sparsity by directly considering the result of the constraint function ϕ as the sanity check outcome. Here, we control for spurious violations in sequence checks by considering an outcome to be satisfied when condition E6 holds (see §V). For this experiment, we focus on the astrophysics scenario, as it features extensive data quality issues that are directly grounded in the respective data sources.

As illustrated in Table V, we analyze the accuracy of the naive approach in the detection of satisfied and violated sanity checks. The combined accuracy for all constraints lies at 0.634 for satisfied and 0.161 for violated outcomes. The result highlights that neglecting data quality issues may drastically change the result of sanity checking, i.e., a naive approach yields numerous checks having the opposite outcome compared to their evaluation under data quality issues. Moreover, the accuracy varies notably among the constraints, so that this issue cannot be addressed by simply tuning the general sensitivity of the naive approach.

The results are further underpinned by an assessment of the cases, where SOUND yields an inconclusive outcome. Here, our approach determines that the uncertainty arising from data quality issues is too large to derive a reliable conclusion. Yet, the naive approach yields either satisfied or violated check outcomes with false confidence, potentially resulting in wrong conclusions. This is the case for $\approx 27.8\%$ of the outcomes.

D. Confidence in Constraint Evaluation

The confidence in a sanity check being satisfied or violated depends on several factors, including (i) a data value's proximity to the decision threshold modelled in a constraint, (ii) the extent of value uncertainty and data sparsity and their relevance for the constraint, as well as (iii) the number of samples drawn in the evaluation. Next, we investigate these influencing factors.

Framework parameters. To assess the impact of N and c on the constraint evaluation quality, we compare four cases of high/low parameter combinations in the evaluation of constraint S-4, see Fig. 7. The overall certainty in the violation probability decreases, as expected, when c is increased, as can be observed by the shorter intervals between the left and the right cases.

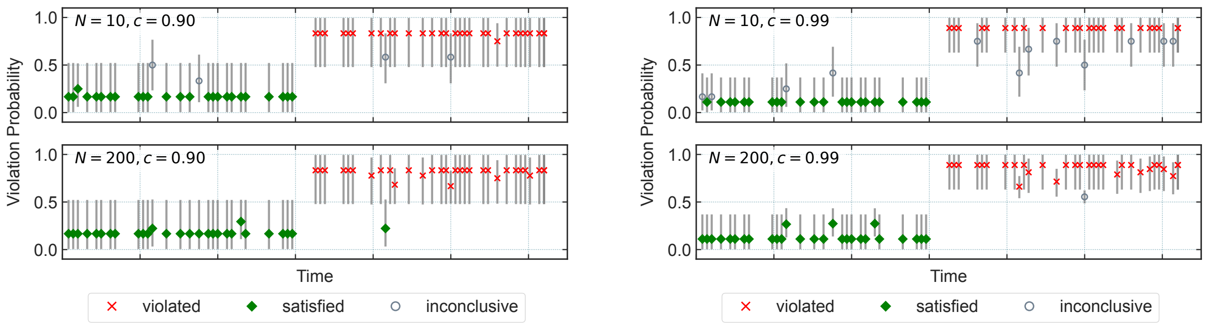

Figure 7: Smart grid scenario: Sanity check evaluation cases with representative parameter pairings, being high/low maximum sample size N and credibility level c. Error bars show 95% confidence interval of final violation probability.

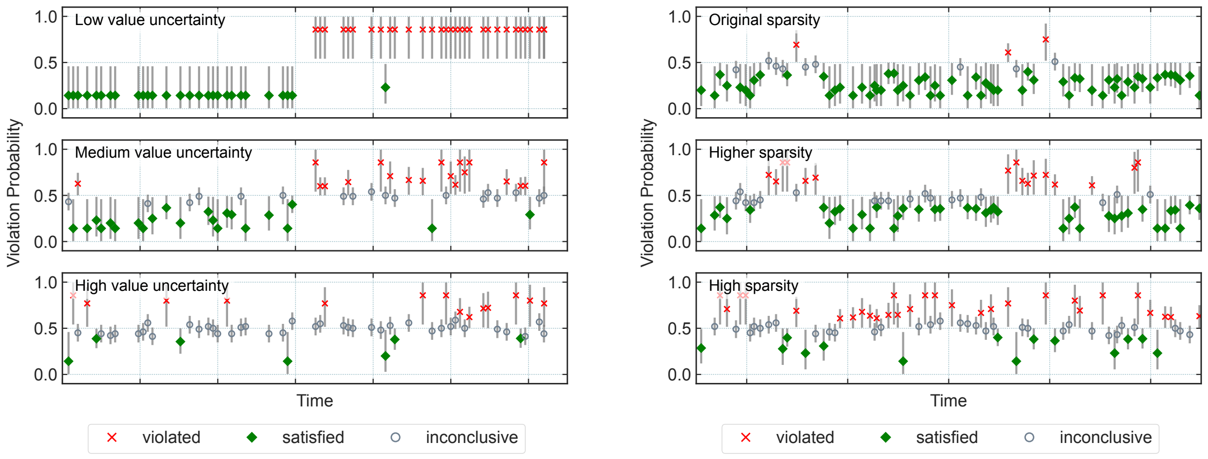

Figure 8: Astrophysics scenario: Sanity check evaluation change point with original and amplified value uncertainty (left) and sparsity (right).

However, in scenarios where the data quality issues render the evaluation of a sanity check not straightforward, a larger sample size is required to reach a conclusive outcome. This can be observed when comparing the top right and the top left panel, where multiple inconclusive outcomes occur with a high c and low N, and mostly disappear when N is increased. Here, it is beneficial to increase N, as more samples make it possible to reach a conclusive outcome. When c is lower, the tolerance for wrong conclusions becomes higher. This is illustrated by the false positive in the bottom left panel, that is not present in the right panels, where c is larger.

Value uncertainty. To study the impact of value uncertainty, we compare three variants of a constraint evaluation (S-4), while manually altering the value uncertainty. Fig. 8 (left) shows cases of the value uncertainty being low (top), moderate (middle), or high (bottom) regarding the decision criteria of the constraint S-4. Low value uncertainty (top panel) results in an overall low variability in the violation probability, and a clear distinction between violation and satisfaction. With increasing value uncertainty in the middle panel, there is more variety in the result, with more inconclusive outcomes and larger spread in violation probability mass. With high value uncertainty (bottom panel), there is large variability in the outcome, and clear cases of high-confidence constraint violations or satisfactions become rare. Instead, the violation probability distribution falls closer to 0.5 and outcomes become inconclusive.

Impact of data sparsity. To evaluate the impact of data sparsity, Fig. 8 (right) shows the evaluation of constraint A-4 with the original data distribution and with amplified data sparsity. Compared to the original series, higher sparsity (middle panel) potentially changes the outcome to be either inconclusive or the opposite. With higher sparsity (bottom), the outcomes frequently change compared to the original series. Opposite outcomes are contexts where the data series with amplified sparsity is unrepresentative of the original, but clearly suggests a wrong outcome. When outcomes change to be inconclusive, the series is ambiguous with respect the sanity check. Here, the amplified data sparsity disturbs the constraint evaluation results, which shows that the propagation of uncertainty from data sparsity works as expected.

E. Violation Analysis

Turning to root-cause analysis of constraint violations, we analyze the effectiveness and efficiency of our automated checks aimed to guide a user's drill-down of potential root causes. In the following, we focus on the constraints A-3 and A-4 in Astrophysics pipeline, as they located far downstream in the DAG and data quality issues are prevalent in general. Hence, they pose the most interesting scenario for violation analysis. We compare SOUND to the BASE_VA baseline based on existing provenance-based techniques.

Table VI: Number of identified explanations (E1-6) and false positive outcome ratio (FPR) of BASE_VA for change points in the violation analysis of the Astrophysics scenario.

constraint	E1	E2	E3	E4	E5	E6	BASE_VA FPR
A-3	28243	0	0	9936	957	1	0.278356
A-4	3179	0	0	452	0	0	0.124484

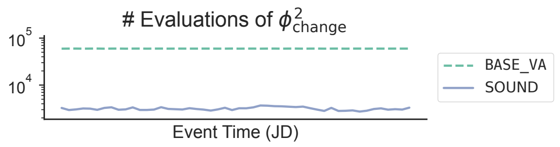

Figure 9: Number of evaluated change constraints for A-3 and A-4 of the Astrophysics scenario, comparing SOUND and BASE_VA.

In Table VI, we show the number of change points associated with each potential explanation in SOUND. Also, we show the false positive rate of BASE_VA. The results demonstrate that SOUND effectively finds explanations for change points, and that a naive approach (BASE_VA) yields spurious explanations, if a change in data quality can explain a violated check. This holds despite data sparsity not being a root-cause, as both investigated constraints operate on fixed-size count-windows.

In Fig. 9, we evaluate the efficiency of SOUND compared to BASE_VA, which proactively carries out checks of data changes in local and upstream data series to isolate root-courses in the pipeline DAG. The results show that SOUND is substantially more efficient, as the reactive approach allows to avoid performing unnecessary data change checks, which on average constitute to about 95% of the checks carried out in BASE_VA.

VII. Related Work

Sanity checking of pipelines links to work on debugging data-driven systems [40], [41], data validation [12], and data provenance [42], [43]. Our work is also related to models developed for probabilistic databases [44] and time series anomaly detection [45]. Yet, SOUND focuses on the impact of data quality issues, whereas probabilistic databases and anomaly detection typically do not model those explicitly.

Debugging Data-Driven Systems. In recent years, various approaches to debug the pipelines executed by data-driven systems have been proposed. MLInspect [46], [47] instruments pipelines extracted from Python scripts with user-specified checks for data distribution issues. BugDoc [48], in turn, finds root-causes for system failures in the parameters of data processing pipelines. Similarly, DataExposer [49] identifies data profiles that are causally connected to system malfunction, based on successful/failed runs. Both BugDoc and DataExposer require an oracle to declare the correctness of a pipeline result, a limitation known as the oracle problem in software testing [50].

Slice Finder [51] relies on a statistical analysis of performance measures for a pipeline to identify data slices for which a machine learning model performs poorly. Coco [52], [53] is a line of work for identifying conformance constraints as invariant relations between attributes of a dataset for data exploration, cleaning and debugging. Root-cause analysis of processing pipelines was realized based on stack traces and data dependencies in Dagger [54] and for fairness-related data-issues in Gopher [55]. Similarly, AID [56] identifies root-causes of bugs by automated code changes and causal inference. Complaint-Driven Data Debugging [57] points out elements in training data that cause undesired model predictions.

The above systems outline a wide range of strategies to explore the validity of processing pipelines. Yet, none of them caters explicitly for the impact of data quality issues on the results. Since data quality issues are omnipresent in many scientific fields, with potentially drastic implications on the ability to assess the validity of processing pipelines, they shall be incorporated in sanity checking, though.

Data Validation. The idea to formulate data validity expectations as constraints was brought forward by Deequ [13] for pipelines in Apache Spark. Since then, similar approaches have been developed for various data processing platforms, including Tensorflow data validation (TFDV) [12], Pandera [58] for the validation of Python dataframes, and GreatExpectations (GX) [11] as a platform for validation, profiling, and documentation of data in Python pipelines. Data Linter [59], in turn, focuses on the validation of training data for machine learning, e.g., class imbalances issues. These approaches are similar in that they also rely on user-defined constraints for capturing validity requirements. Unlike SOUND, they do not cater for data quality issues or non-unary constraints in context of a pipeline.

Data Provenance. Our strategy to identify a root-cause of a sanity check violation, if it is suspected to be a change in data values, exploits the pipeline provenance. Many systems are available to track and visualize such provenance information, e.g., VAMSA [60], Vizier [61], and VisTrails [62]. An integration of SOUND with these systems may enable even richer analytics, e.g., by including pipeline revisions in the identification of root-causes of validity issues.

VIII. Conclusions

In this paper, we presented SOUND to enable sanity checking of pipelines for data series in the presence of data quality issues. We proposed a rich constraint model to formalize validity expectations for data series and presented a statistical framework for their evaluation once data series include value uncertainty and intervals of data sparsity. Further, we proposed automated checks for drilling into the root-cause of constraint violations, thereby guiding users in the interpretation of a constraint violation. We demonstrated the feasibility and utility of SOUND by applying it for pipelines developed in the domains of smart grid monitoring and astrophysics. Here, we showed that SOUND enables sanity checking with a modest performance overhead, while incorporating data quality issues effectively and providing guidance on the identification of root-causes.

Acknowledgment

This work is supported by the Helmholtz Einstein International Berlin Research School in Data Science (HEIBRiDS) and the Helmholtz Information & Data Science Academy (HIDA).

References

[1] J. Hudson, S. Fielding, and C. R. Ramsay, "Methodology and reporting characteristics of studies using interrupted time series design in healthcare," *BMC medical research methodology*, vol. 19, pp. 1–7, 2019.

[2] M. F. Dixon, I. Halperin, and P. Bilokon, *Machine learning in finance*. Springer, 2020, vol. 1170.

[3] C. Wang, X. Huang, J. Qiao, T. Jiang, L. Rui, J. Zhang, R. Kang, J. Feinauer, K. A. McGrail, P. Wang *et al.*, "Apache iotdb: Time-series database for internet of things," *Proceedings of the VLDB Endowment*, vol. 13, no. 12, pp. 2901–2904, 2020.

[4] Y. Ban, P. Zhang, A. Nascetti, A. R. Bevington, and M. A. Wulder, "Near real-time wildfire progression monitoring with sentinel-1 sar time series and deep learning," *Scientific reports*, vol. 10, no. 1, p. 1322, 2020.

[5] A. Blázquez-García, A. Conde, U. Mori, and J. A. Lozano, "A review on outlier/anomaly detection in time series data," *ACM Comput. Surv.*, vol. 54, no. 3, apr 2021. [Online]. Available: https://doi.org/10.1145/3444690

[6] A. P. Ruiz, M. Flynn, J. Large, M. Middlehurst, and A. Bagnall, "The great multivariate time series classification bake off: a review and experimental evaluation of recent algorithmic advances," *Data Mining and Knowledge Discovery*, vol. 35, no. 2, pp. 401–449, 2021.

[7] A. Katsifodimos and S. Schelter, "Apache flink: Stream analytics at scale," in *2016 IEEE International Conference on Cloud Engineering Workshop (IC2EW)*, 2016, pp. 193–193.

[8] M. Zaharia *et al.*, "Apache spark: A unified engine for big data processing," *Commun. ACM*, vol. 59, no. 11, p. 56–65, oct 2016. [Online]. Available: https://doi.org/10.1145/2934664

[9] C. S. Liew, M. P. Atkinson, M. Galea, T. F. Ang, P. Martin, and J. I. V. Hemert, "Scientific workflows: Moving across paradigms," *ACM Comput. Surv.*, vol. 49, no. 4, Dec. 2016.

[10] M. A. Jamil, M. Arif, N. S. A. Abubakar, and A. Ahmad, "Software testing techniques: A literature review," in *2016 6th international conference on information and communication technology for the Muslim world (ICT4M)*. IEEE, 2016, pp. 177–182.

[11] A. Gong, J. Campbell, and Great Expectations, "Great Expectations." [Online]. Available: https://github.com/great-expectations/great_expectations

[12] E. Breck, M. Zinkevich, N. Polyzotis, S. Whang, and S. Roy, "Data validation for machine learning," in *Proceedings of SysML*, 2019. [Online]. Available: https://mlsys.org/Conferences/2019/doc/2019/167.pdf

[13] S. Schelter, D. Lange, P. Schmidt, M. Celikel, F. Biessmann, and A. Grafberger, "Automating large-scale data quality verification," *Proceedings of the VLDB Endowment*, vol. 11, no. 12, pp. 1781–1794, Aug. 2018.

[14] F. Schintke, K. Belhajjame, N. D. Mecquenem, D. Frantz, V. E. Guarino, M. Hilbrich, F. Lehmann, P. Missier, R. Sattler, J. A. Sparka, D. T. Speckhard, H. Stolte, A. D. Vu, and U. Leser, "Validity constraints for data analysis workflows," *Future Gener. Comput. Syst.*, vol. 157, pp. 82–97, 2024. [Online]. Available: https://doi.org/10.1016/j.future.2024.03.037

[15] Q. Tan, M. Ye, B. Yang, S. Liu, A. J. Ma, T. C.-F. Yip, G. L.-H. Wong, and P. Yuen, "Data-gru: Dual-attention time-aware gated recurrent unit for irregular multivariate time series," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 01, 2020, pp. 930–937.

[16] B. Goswami, N. Boers, A. Rheinwalt, N. Marwan, J. Heitzig, S. F. Breitenbach, and J. Kurths, "Abrupt transitions in time series with uncertainties," *Nature communications*, vol. 9, no. 1, p. 48, 2018.

[17] Z. Abedjan, L. Golab, and F. Naumann, "Profiling relational data: a survey," *VLDB J.*, vol. 24, no. 4, pp. 557–581, 2015. [Online]. Available: https://doi.org/10.1007/s00778-015-0389-y

[18] P. Esling and C. Agón, "Time-series data mining," *ACM Comput. Surv.*, vol. 45, no. 1, pp. 12:1–12:34, 2012. [Online]. Available: https://doi.org/10.1145/2379776.2379788

[19] N. R. Mabroukeh and C. I. Ezeife, "A taxonomy of sequential pattern mining algorithms," *ACM Comput. Surv.*, vol. 43, no. 1, pp. 3:1–3:41, 2010. [Online]. Available: https://doi.org/10.1145/1824795.1824798

[20] A. Donath, R. Terrier, Q. Remy, A. Sinha, C. Nigro, F. Pintore, B. Khélifi, L. Olivera-Nieto, J. E. Ruiz, K. Brügge *et al.*, "Gammapy: A python package for gamma-ray astronomy," *Astronomy & Astrophysics*, vol. 678, p. A157, 2023.

[21] C. J. Merchant, F. Paul, T. Popp, M. Ablain, S. Bontemps, P. Defourny, R. Hollmann, T. Lavergne, A. Laeng, G. De Leeuw *et al.*, "Uncertainty information in climate data records from earth observation," *Earth System Science Data*, vol. 9, no. 2, pp. 511–527, 2017.

[22] M. Jain, V. Chandan, M. Minou, G. A. Thanos, T. K. Wijaya, A. Lindt, and A. Gylling, "Methodologies for effective demand response messaging," in *2015 IEEE International Conference on Smart Grid Communications, SmartGridComm 2015, Miami, FL, USA, November 2-5, 2015*. IEEE, 2015, pp. 453–458. [Online]. Available: https://doi.org/10.1109/SmartGridComm.2015.7436342

[23] Z. Jerzak and H. Ziekow, "The debs 2014 grand challenge," in *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems*, ser. DEBS '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 266–269. [Online]. Available: https://doi.org/10.1145/2611286.2611333

[24] G. Raman, B. Zhao, J. C.-H. Peng, and M. Weidlich, "Adaptive incentive-based demand response with distributed non-compliance assessment," *Applied Energy*, vol. 326, p. 119998, 2022.

[25] T. C. T. A. Consortium, "Science with the cherenkov telescope array," *International Journal of Modern Physics D*, 2015.

[26] D. N. Burrows *et al.*, "The Swift X-Ray Telescope," vol. 120, no. 3, pp. 165–195. [Online]. Available: https://doi.org/10.1007/s11214-005-5097-2

[27] E. C. Bellm *et al.*, "The Zwicky Transient Facility: System Overview, Performance, and First Results," *Publications of the Astronomical Society of the Pacific*, vol. 131, no. 995, p. 018002, Jan. 2019.

[28] A. Artikis, M. Weidlich, F. Schnitzler, I. Boutsis, T. Liebig, N. Piatkowski, C. Bockermann, K. Morik, V. Kalogeraki, J. Marecek, A. Gal, S. Mannor, D. Gunopulos, and D. Kinane, "Heterogeneous stream processing and crowdsourcing for urban traffic management," in *Proceedings of the 17th International Conference on Extending Database Technology, EDBT 2014, Athens, Greece, March 24-28, 2014*, S. Amer-Yahia, V. Christophides, A. Kementsietsidis, M. N. Garofalakis, S. Idreos, and V. Leroy, Eds. OpenProceedings.org, 2014, pp. 712–723. [Online]. Available: https://doi.org/10.5441/002/edbt.2014.77

[29] F. Carcillo, Y. L. Borgne, O. Caelen, and G. Bontempi, "An assessment of streaming active learning strategies for real-life credit card fraud detection," in *2017 IEEE International Conference on Data Science and Advanced Analytics, DSAA 2017, Tokyo, Japan, October 19-21, 2017*. IEEE, 2017, pp. 631–639. [Online]. Available: https://doi.org/10.1109/DSAA.2017.10

[30] A. Artikis, N. Katzouris, I. Correia, C. Baber, N. Morar, I. Skarbovsky, F. Fournier, and G. Paliouras, "A prototype for credit card fraud management: Industry paper," in *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems, DEBS 2017, Barcelona, Spain, June 19-23, 2017*. ACM, 2017, pp. 249–260. [Online]. Available: https://doi.org/10.1145/3093742.3093912

[31] G. Casella and R. L. Berger, *Statistical inference*. Duxbury Pacific Grove, CA, 2002.

[32] C. Robert and G. Casella, *Monte Carlo statistical methods*. Springer Verlag, 2004.

[33] B. Efron, "Bootstrap Methods: Another Look at the Jackknife," *The Annals of Statistics*, vol. 7, no. 1, pp. 1 – 26, 1979. [Online]. Available: https://doi.org/10.1214/aos/1176344552

[34] S. Gonçalves and D. Politis, "Discussion: Bootstrap methods for dependent data: A review," vol. 40, no. 4, pp. 383–386. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1226319211000664

[35] D. N. Politis and J. P. Romano, "The stationary bootstrap," *Journal of the American Statistical Association*, vol. 89, no. 428, pp. 1303–1313, 1994.

[36] F. J. Massey Jr, "The kolmogorov-smirnov test for goodness of fit," *Journal of the American statistical Association*, vol. 46, no. 253, pp. 68–78, 1951.

[37] D. Palyvos-Giannas, K. Tzompanaki, M. Papatriantafilou, and V. Gulisano, "Erebus: Explaining the outputs of data streaming queries," *Proc. VLDB Endow.*, vol. 16, no. 2, p. 230–242, oct 2022. [Online]. Available: https://doi.org/10.14778/3565816.3565825

[38] H. Stolte, "Reproducibility Setup for SOUND: Sanity Checking of Pipelines for Uncertain and Sparse Data Series," Oct. 2024. [Online]. Available: https://doi.org/10.5281/zenodo.14008484

[39] A. A. Abdo, M. Ackermann, M. Ajello, W. Atwood, M. Axelsson, L. Baldini, J. Ballet, D. Band, G. Barbiellini, D. Bastieri *et al.*, "Fermi/large area telescope bright gamma-ray source list," *The Astrophysical Journal Supplement Series*, vol. 183, no. 1, p. 46, 2009.

[40] B. Glavic, A. Meliou, and S. Roy, "Trends in explanations: Understanding and debugging data-driven systems," *Foundations and Trends® in Databases*, vol. 11, no. 3, pp. 226–318, 2021.

[41] J. M. Zhang, M. Harman, L. Ma, and Y. Liu, "Machine learning testing: Survey, landscapes and horizons," *IEEE Transactions on Software Engineering*, 2020.

[42] B. Glavic, "Data provenance - origins, applications, algorithms, and models," *Foundations and Trends® in Databases*, vol. 9, no. 3-4, pp. 209–441, 2021.

[43] M. Herschel, R. Diestelkämper, and H. Ben Lahmar, "A survey on provenance: What for? what form? what from?" *The VLDB Journal*, vol. 26, no. 6, p. 881–906, dec 2017.

[44] D. Suciu, D. Olteanu, R. Christopher, and C. Koch, *Probabilistic Databases*, 1st ed. Morgan & Claypool Publishers, 2011.

[45] S. Schmidl, P. Wenig, and T. Papenbrock, "Anomaly detection in time series: A comprehensive evaluation," *Proc. VLDB Endow.*, vol. 15, pp. 1779–1797, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:250331153

[46] S. Grafberger, J. Stoyanovich, and S. Schelter, "Lightweight inspection of data preprocessing in native machine learning pipelines," in *Conference on Innovative Data Systems Research (CIDR)*, 2021.

[47] S. Grafberger, P. Groth, J. Stoyanovich, and S. Schelter, "Data distribution debugging in machine learning pipelines," *The VLDB Journal*, pp. 1–24, 2022.

[48] R. Lourenço, J. Freire, and D. Shasha, "Bugdoc: Algorithms to debug computational processes," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 463–478.

[49] S. Galhotra, A. Fariha, R. Lourenço, J. Freire, A. Meliou, and D. Srivastava, "Dataexposer: Exposing disconnect between data and systems," *CoRR*, vol. abs/2105.06058, 2021.

[50] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The oracle problem in software testing: A survey," *IEEE Transactions on Software Engineering*, vol. 41, no. 5, pp. 507–525, 2015.

[51] N. Polyzotis, S. Whang, T. K. Kraska, and Y. Chung, "Slice finder: Automated data slicing for model validation," in *Proceedings of the IEEE Int' Conf. on Data Engineering (ICDE), 2019*, 2019. [Online]. Available: https://arxiv.org/pdf/1807.06068.pdf

[52] A. Fariha, A. Tiwari, A. Meliou, A. Radhakrishna, and S. Gulwani, "Coco: Interactive exploration of conformance constraints for data understanding and data cleaning," in *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, 2021, pp. 2706–2710.

[53] A. Fariha, A. Tiwari, A. Radhakrishna, S. Gulwani, and A. Meliou, "Conformance constraint discovery: Measuring trust in data-driven systems," in *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, 2021, pp. 499–512.

[54] E. K. Rezig, L. Cao, G. Simonini, M. Schoemans, S. Madden, N. Tang, M. Ouzzani, and M. Stonebraker, "Dagger: A data (not code) debugger," in *10th Conference on Innovative Data Systems Research, CIDR 2020, Amsterdam, The Netherlands, January 12-15, 2020, Online Proceedings*. www.cidrdb.org, 2020. [Online]. Available: http://cidrdb.org/cidr2020/papers/p35-rezig-cidr20.pdf

[55] B. Salimi, R. Pradhan, J. Zhu, and B. Glavic, "Interpretable data-based explanations for fairness debugging," in *Proceedings of the 48th International Conference on Management of Data*, 2022, pp. 247–261.

[56] A. Fariha, S. Nath, and A. Meliou, "Causality-Guided Adaptive Interventional Debugging," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. Portland OR USA: ACM, Jun. 2020, pp. 431–446.

[57] L. Flokas, W. Wu, Y. Liu, J. Wang, N. Verma, and E. Wu, "Complaint-Driven Training Data Debugging at Interactive Speeds," in *Proceedings of the 2022 International Conference on Management of Data*. Philadelphia PA USA: ACM, Jun. 2022, pp. 369–383.

[58] Niels Bantilan, "pandera: Statistical Data Validation of Pandas Dataframes," in *Proceedings of the 19th Python in Science Conference*, Meghann Agarwal, Chris Calloway, Dillon Niederhut, and David Shupe, Eds., 2020, pp. 116 – 124.

[59] N. Hynes, D. Sculley, and M. Terry, "The data linter: Lightweight automated sanity checking for ml data sets," 2017. [Online]. Available: http://learningsys.org/nips17/assets/papers/paper_19.pdf

[60] M. H. Namaki, A. Floratou, F. Psallidas, S. Krishnan, A. Agrawal, and Y. Wu, "Vamsa: Tracking provenance in data science scripts," *CoRR*, vol. abs/2001.01861, 2020.

[61] M. Brachmann, C. Bautista, S. Castelo, S. Feng, J. Freire, B. Glavic, O. Kennedy, H. Müeller, R. Rampin, W. Spoth *et al.*, "Data debugging and exploration with vizier," in *Proceedings of the 2019 International Conference on Management of Data*, 2019, pp. 1877–1880.

[62] J. Freire and C. T. Silva, "Making computations and publications reproducible with vistrails," *Comput. Sci. Eng.*, vol. 14, no. 4, pp. 18–25, 2012.