Data Ingestion Validation through Stable Conditional Metrics with Ranking and Filtering

Niels Bylois
UHasselt
Belgium

Frank Neven
UHasselt
Belgium

Stijn Vansummeren
UHasselt
Belgium

Abstract

Validating the quality of continuously collected data is crucial to ensure trustworthiness of analytics insights. A widely used approach for validating data quality is to specify, either manually or automatically, so-called data unit tests that check whether data quality metrics lie within expected bounds. Unfortunately, these existing approaches suffer from two limitations. First, as we show in this paper, the data unit tests that they support are based on *global* metrics that can only provide *coarse-grained* signals of data quality and are unable to detect *fine-grained* errors. Second, even when a data unit test signals a data quality problem, it does not provide a principled method to identify the data part that is responsible for a test's failure. We present an approach for data quality validation that is not only capable of detecting fine-grained errors, but also helps in identifying responsible erroneous tuples. Our approach is based on a novel form of metrics, called conditional metrics, which allow to compute data quality signals over specific parts of the ingestion data and therefore allow for a more fine-grained analysis compared to standard global metrics that operate on the entire ingestion data. Our approach consists of two phases: a unit test discovery phase and a monitoring and error identification phase. In the discovery phase, we automatically derive conditional metric-based unit tests from historical ingestion sequences, where we employ a notion of stability over the historical ingestions as a selection criterion for using conditional metrics as data unit tests. In the subsequent phase, we use the derived unit tests to validate the quality of new ingestion batches. When an ingestion batch fails one or more unit tests, we show how conditional metrics can be used to identify potential errors. We study different ways of implementing both phases, and compare their effectiveness. We evaluate our approach on two datasets and seven synthetic error scenarios. The improvement that we measure over global metrics as well as the error-identification F1-scores that we obtain indicate that conditional metrics provide a promising approach towards fine-grained error detection for data ingestion validation.

Keywords

data cleaning, data profiling, dynamic data

1 Introduction

Data-driven decision making permeates all levels of modern enterprises and organisations. At the basis of this decision process lies the continuous collection and ingestion of relevant data from diverse sources. Validating the quality of collected data at ingestion time is crucial for numerous reasons. First, and foremost, the quality of the derived insights, and the decisions driven by them, depend directly on the quality of the collected data [25]. Furthermore, as more and more of the data analysis process is automated, small errors in source data risk propagating to later data consumers (such as machine learning models), which may themselves act as data sources in other processing pipelines—thereby potentially magnifying the error [7]. Finally, data errors may even cause data processing pipelines to crash (e.g., because of null pointer exceptions due to missing data). Machine learning platforms are therefore including explicit data validation components into their pipelines [5, 6, 9].

In recognition of the importance of data quality validation in modern data processing pipelines, several tools have been proposed to aid in automatic validation [7, 20, 25]. Broadly speaking, these tools allow specification, either manually or automatically, of so-called *data unit tests*. When a new batch of data is to be ingested, the registered tests are executed to gauge the batch's quality where failing tests highlight data quality problems. The tests themselves entail computing certain metrics on the data batch (e.g., the minimum or maximum value appearing in a numerical column or the number of distinct elements appearing in a column, see also Table 1) and checking that these fall within an expected range.

Unfortunately, these tools suffer from two limitations. First, the data unit tests that they support are based on *global* metrics: metrics that are computed on the entire data batch, or on an entire column in the batch. As the following examples show, they hence only provide *coarse-grained* signals of data quality and are unable to detect *fine-grained* errors, i.e., errors that occur only in a specific (potentially small) part of the batch. Batches with fine-grained errors hence go unnoticed. Second, even when a data unit test signals that a batch has a data quality problem, it does not provide a principled method to identify the part of the batch that is responsible for a test's failure. As such, either the entire batch must be discarded or a human expert must manually identify and subsequently remove or correct the erroneous tuples—which is time-consuming and labor-intensive.

Example 1.1. A public railway company has equipped all of its trains with measurement sensors that record the train's arrival and departure time at each train station. By comparing these times with the time schedule, the train's software computes the corresponding delays (if any). At the end of each day, the measurements of all trains are collected, and ingested in the railway company's data lake. The delays are used to identify hotspot routes, as well as computation of service quality indicators to the government. Train 5437 runs daily from Hasselt (Belgium) to Blankenberge (Belgium). This route is notorious for the delays that it incurs when it passes through the busy Brussels railway stations. As such, train 5437 normally reports non-zero delay. Due to a hardware malfunction on September 15, however, it consistently reports zero delay for this train.

The metrics used by state-of-the-art tools are unable to detect this error. Indeed: because zero delay is *not* an uncommon value when considering the entire ingestion batch (some trains run on time), metrics such as min(delay), max(delay), and avg(delay) will

not consider zero delay as an anomaly. Also other commonly used global metrics do not detect this error, as we later confirm experimentally in Section 6.6.

It is important to observe that, even if one of the global metrics signals a data quality problem, it is not clear which train or set of trains in the batch cause the problem. For instance, if the unit tests based on min(delay) or max(delay) signal an anomaly then of course we can easily identify the erroneous trains: simply compute the trains whose delay value is below (or above) the expected minimum (resp. maximum) value. However, if a unit test based on avg(delay) signals a problem, then it is unclear how to identify the trains that caused the delay to deviate. □

Example 1.2. A retail intelligence company scrapes the websites of online retailers on a daily basis to collect up-to-date pricing information of various products. The company monetizes the gathered information by selling it to interested retailers to aid in price comparisons. As such, ensuring the quality of the scraped information is paramount. On September 30, one of the to-be-scraped retailers modifies its website slightly, which confuses the scraper software and causes it to assign the erroneous product category "Pet supplies" to a specific brand of TVs instead of the correct category "TVs". Since this does not modify the set of possible values of the category column in the scraped dataset (there are still other TVs), and does not change other characteristics (total number of items scraped, minimum or maximum prices, etc) also this error cannot be detected using global metrics. □

In this paper, we present an approach for data quality validation that is not only capable of detecting the *fine-grained errors* illustrated above, but also helps in *identifying* responsible erroneous tuples. We focus on the setting where a data pipeline regularly ingests batches of external data. As the main technical vehicle underlying our approach, we introduce *conditional metrics* (CM for short). In contrast to a global metric, a conditional metric only computes its value on a specified subset of tuples in the ingestion batch. In the railway example above, for instance, the conditional metric $\textsc{avg}(\mathit{Delay} \mid \mathit{Train} = 5437)$ computes the average delay of train 5437 in the batch, as opposed to the global metric $\textsc{avg}(\mathit{Delay})$ which computes the average delay of all trains. Likewise, in the retail example, the conditional metric $\textsc{count}(\mathit{Item} \mid \mathit{Cat} = \text{TV})$ computes the number of items in the TV category, as opposed to global metric $\textsc{count}(\mathit{Item})$ that computes the total number of items. Because of their ability to calculate values for specific entities (e.g., $\mathit{Train} = 5437$), data unit tests based on conditional metrics may hence detect data quality issues at a finer level of granularity than data unit tests based on global metrics.

Our approach consists of two phases: (i) a unit test discovery phase and (ii) a monitoring and error identification phase, as schematically illustrated in Figure 1. In the *unit test discovery phase*, we are given a sequence $\overline{R}$ of previously ingested batches and our objective is to automatically derive a set Θ of CM-based data unit tests from $\overline{R}$ such that a yet-to-be-ingested batch B can be considered to be of acceptable quality if it passes all tests in Θ. Specifically, like [20], we assume that the batches in $\overline{R}$ are themselves of acceptable data quality—which is reasonable because they have already been ingested successfully. Under this assumption, we may derive CM-based data unit tests from $\overline{R}$ by considering the set of all

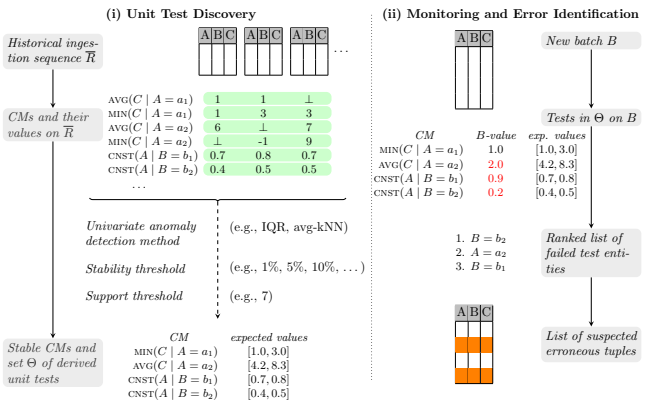


Figure 1: Overview of approach: (i) ***discovery phase*** **to derive data unit tests based on stable conditional metrics; (ii)** ***monitoring and error identification phase*** **where each new batch is validated and erroneous tuples are reported.**

possible CMs, and deriving, for each such CM m, a classifier that allows to distinguish expected values of m from anomalies. If m produces an anomaly on a yet-to-be-ingested batch B, then B will be flagged as having a quality issue. To derive the classifier based on the values of m observed in $\overline{R}$, in principle any anomaly detection method [10] may be used. Whatever method is chosen, we require that the derived classifier is consistent with our assumption that $\overline{R}$ is of reasonable quality, which implies that the values of m observed in $\overline{R}$ are without significant anomalies. We therefore propose a simple notion of *stability* as a means to decide on the set of CMs to promote to unit tests: a CM is *stable* for $\overline{R}$ w.r.t. anomaly detection method A when the number of anomalies detected by the classifier produced by A in $\overline{R}$ does not exceed a predetermined threshold. Only stable CMs are promoted to unit tests.

In the *monitoring and error identification phase*, we take the set Θ of data unit tests derived in the discovery phase, and use this to validate each new ingestion batch B. If all tests in Θ succeed on B then B is deemed to be of acceptable quality. When at least one test in Θ rejects B then our objective is to identify the tuples in B with suspected errors. Every conditional unit test specifies a set of erroneous tuples in a natural way: the subrelation of B that its CM refers to. Simply flagging *all* the tuples of violated unit tests selects too much, however (i.e., it results in high recall but very low precision). The reason is that violating tests are often correlated: a single error in B may cause multiple tests (each selecting different subrelations) to fail. The key challenge in the monitoring phase, therefore, lies in *ranking* the violated unit tests according to relevance, and from this ranked list of tests *filter* a list of suspected erroneous tuples for further inspection.

Our contributions are as follows. (1) We propose *conditional metrics* as a means for providing fine-grained data quality signals (Section 3), and the two-phase framework summarized in Figure 1 as a general framework for data ingestion validation as well as identification of erroneous tuples. We refer to this framework as SCMRF (*Stable Conditional Metrics with Ranking and Filtering*).

(2) We study the design space for discovering data unit tests in the discovery phase (Section 4). Specifically, because each CM m

produces a real value on a historical batch in $\overline{R}$, we can restrict our attention to univariate anomaly detection methods—which are significantly simpler than multivariate anomaly detection methods [10]. We consider representatives of two main classes of univariate anomaly detection methods: global methods (represented by the IQR method) and density-based methods (represented by avg-kNN) and compare their performance in defining correct sets of expected values on various error scenarios. We find that the simple IQR method is capable of finding a significant part of the errors while outperforming the avg-kNN method in the majority of the considered scenarios.

(3) We study the design space for identifying erroneous tuples in the monitoring phase. Specifically, we formulate the problem of ranking unit tests according to relevance as a problem of ranking nodes in a special kind of bipartite graph. We establish that likely candidates for ranking nodes, based on popular methods for identifying authorative or central nodes, work poorly in this respect. In response, we develop two new, tailored, ranking measures (*average tuple degree (atd)* and *correlation density (cd)*) together with an associated treshold-determination algorithm that can successfully identify erroneous tuples. (Section 5)

(4) We present an extensive experimental validation of our approach on two real world datasets and seven error scenarios (Section 6). Let SCMRF(A,f) refer to our method when using anomaly detection method A and entity ranking measure f. Our experiments show that SCMRF(IQR,*cd*) is the best method among the different choices of A and f as it finds a significant part of the errors in the majority of the scenario's (F1 $\geq$ 0.79). We evaluate the impact of the stability threshold selection, anomaly detection mechanisms, and ranking measures. We find that IQR is favorable over avg-kNN with an optimal stability threshold of 0.05, and that *cd* is superior over *atd* and well-known importance measures for nodes in a graph like PageRank, HITS and those based on centrality notions. We compare with the approach of [20] that is based on the use of global metrics, and find that our approach can detect fine-grained errors that are not detected by global metrics. In addition, SCMRF offers a direct way to identify suspected erroneous tuples which approaches based on global metrics do not. In conclusion, the obtained F1-scores and improvement over global metrics indicate that conditional metrics provide a promising approach towards fine-grained error detection for data ingestion validation.

We discuss related work in Section 2, and conclude in Section 7.

2 Related Work

Data quality. How does one measure the quality of data? It has long been accepted that data quality is a subjective matter, since what one really wishes to assess is data's "fitness of use" in a particular context [28]. The data quality research literature has identified a large number of *dimensions* along which fitness for use in a particular context can be assessed [26]. Examples of such dimension include *completeness* (the degree to which the data contains all the information required to describe a real-word entity), *consistency* (the degree to which a set of semantic rules are satisfied), *timeliness* (how up-to-date is the data), and *accuracy* (the degree to which the data correctly represents the real-world). For each dimension, numerous ways to measure quality have been proposed. In this paper, we follow the state of the art and compute quantitative statistics (in the forms of what we will call *metrics* and *conditional metrics*) on ingested data as a means to identify data quality issues related to the completeness and consistency dimensions. Table 1 gives an overview of the metrics that we consider, and how they relate to these dimensions.

Data quality validation. As already stated in the Introduction, multiple tools validate the quality of data based on the abovementioned quantitative statistics [7, 20, 25]. Specifically, Schelter et al. [24, 25] propose Deequ, a system where data unit tests can be specified in a declarative manner, and are automatically computed incrementally for scalable execution. The system offers a simple heuristic-driven constraint suggestion functionality based on single-column profiling where a user is expected to select and validate recommendations. Tensorflow Data Validation [7] aims at validating the input data of ML pipelines through a user-defined data schema that constrains attribute names, data domains, and various quantitative statistics (e.g., data distribution, uniqueness, sparsity, etc.). An initial data schema can be inferred automatically by analyzing reference data. Conceptually, the work [20] aims to automatically derive data unit test by means of machine learning. Specifically, the data validation problem is cast as a novelty detection machine learning problem utilizing global metrics as features.

All of these proposals are coarse-grained in that they compute the quantitative data statistics on the entire input data batch, or on an entire column thereof. In our work, in contrast, and as exemplified in Example 1.1 and 1.2, we adopt a more fine-grained approach based on conditional metrics that can detect problems when a smaller number of records is affected, and that in addition facilitates identification of the part of the batch that is responsible for the test's failure.

ML Model validation and debugging Most recent work on validating and debugging machine learning models employ slice finding to identify subsets of the data where a trained model performs poorly [18, 19, 23]. Targeted applications include model validation, error analysis, fairness and model understanding. This line of research is cast in a supervised setting and differs from the unsupervised data ingestion validation problem that we discuss in this paper. While conditions (predicates) are used to define slices, the slices itself are not used as conditional metric to gauge quality of the source data. In fact, the 'metrics' used by slice finding are fixed: effect size of a slice and statistical significance of loss. Furthermore, the cited work considers a single batch, whereas the presence of historical batches is essential in our work given the unsupervised setting. TensorFlow Model Analysis[1] also targets model validation combined with slicing. The same differences with our work apply.

Picket [16] represents a different approach to guarding against corrupted data during learning. In particular, it adopts an unsupervised approach based on multi-head self-attention that, given a tabular dataset D with noise identifies a clean subset $C \subseteq D$ to be used during training. In contrast to our work, Picket works a single dataset instead of on ingestion sequences. Furthermore, Picket decides whether a tuple is clean or erroneous purely on the basis of that tuple's content alone. Our work also takes into account the context in which that tuple occurs in the ingestion batch.

[1] https://github.com/tensorflow/model-analysis

Error detection for data cleaning. There is a large body of data management research related to data cleaning aimed at capturing errors at different granularities (see, e.g., [1, 13, 14, 17]). Abedjan et al. [2] delineate four categories of error detection solutions: (a) rule violation, (b) pattern violation, (c) outlier detection, and (d) duplicate conflict resolution based systems. One line of work focuses on the consistency dimension of data quality, and has proposed various kinds of *dependencies* as a means of formalizing data consistency. Such dependencies include functional dependencies, conditional functional dependencies, inclusion dependencies, denial constraints, and approximate versions thereof (see, e.g., [3, 14]). In contrast, we use quantitative statistics, not dependencies, as a means to assess data quality at both the completeness and consistency dimension. In fact, we adopt the approach of Redyuk et al. [20] utilizing feature vectors based on data quality metrics computed over ingestion batches rather than working with the raw data itself. Quantitative statistics are used as one of the input features in recent systems adopting a probabilistic view of data cleaning, as for instance [13, 21, 22]. In those works, as in [20], the quantitative statistics are assumed to be given and need not be derived as is the case for conditional metrics in the present paper. We compare with [20] in Section 6.6.

Contextual outlier detection. Contextual outlier detection techniques (e.g., [27, 29]) select a subset of attributes as contextual attributes and a subset as indicator attributes. The objective is to identify outliers within the values for indicator attributes over a subrelation (a context) selected via the the contextual attributes. Conditional metrics are computed over contexts as well but differ from the above contextual outlier detection in the following aspect: outliers are not identified based on the frequency of values occurring in the indicator attributes of the same relation but are assessed based on data quality metrics over an indicator attribute relative to previous ingestion batches.

3 Conditional Metrics

We assume that the reader is familiar with the relational data model and the standard notation of relational algebra (cf., e.g., [4]). Because metrics may yield different results when computing on sets as opposed to bags, we will work with the bag-based version of relational algebra. In particular, our relations may contain duplicates, and projection does not remove duplicates.

Metrics. A *global metric (GM) on attribute A* is a function that maps non-empty relations over the single attribute A to a real number. If μ is a GM, we will use the notation $\mu(A)$ to stress that μ is a GM on attribute A. Global metrics are extended to apply to non-empty relations with arbitrary schema (containing attribute A) by first projecting on A, i.e., $\mu(R) := \mu(\pi_A(R))$.

We assume given a fixed finite set $\mathcal{M}$ of GMs on a fixed set of attributes. Concrete GMs that we will consider in our experiments are, for example, $\textsc{count}(A)$ which counts the number of rows in a relation R, $\textsc{count-dist}(A)$ which counts the number of distinct A-values, $\textsc{avg}(A)$ that computes the average, and so on. Note that the availability of a global metric on A crucially depends on the type of A (i.e., $\textsc{avg}(\cdot)$ only makes sense on numerical attributes, and $\textsc{min}(\cdot)$ only on attributes that are equipped with an order). Table 1 lists all the GMs that we consider in our experimental validation.

Metric	Type	Definition
Dimension completeness		
COUNT	any	number of rows
COUNT-DIST	any	number of distinct values
DISTINCTNESS	any	number of distinct values divided by number of rows
COUNT-NULL	any	number of NULL values
NULLNESS	any	number of NULL values divided by number of rows
Dimension consistency		
CONSTANCY	any	frequency of most frequent value divided by number of rows
MIN	numeric	min value
MAX	numeric	max value
AVG	numeric	average value
SUM	numeric	sum of the values
STDDEV	numeric	standard deviation
MAXDIGITS	numeric	max number of digits
MAXDEC	numeric	max number of decimals
MINLEN	text	min text length
MAXLEN	text	max text length
AVGLEN	text	average text length
MEDIANLEN	text	median text length

Table 1: Metrics considered in this paper.

Conditional metrics. A *conditional metric (CM) m* is an expression of the form $\mu(Y \mid X = x)$ where X, Y are attributes, μ is a GM on Y, and x is a value in the domain of X. If R is a relation whose schema includes at least the attributes X and Y, then the result of evaluating m on R is defined as

$$m(R) := \begin{cases} \mu(\pi_Y \sigma_{X=x}(R)) & \text{if } x \in \pi_X(R), \\ \bot & \text{otherwise.} \end{cases}$$

That is, $m(R)$ applies μ on the part of R where $X = x$ if x occurs in R, and is $\bot$ otherwise.

We call X the *conditioning attribute* (also known as the *group by* attribute), and Y the *metric attribute* (also known as the *aggregation* attribute) of m. We also refer to x as the *key* of m.

Example 3.1. We further develop the scenario mentioned in Example 1.1. Consider the conditional metrics

$$m_1 = \textsc{avg}(Delay \mid Train = 5437)$$
$$m_2 = \textsc{avg}(Delay \mid Train = 2891)$$
$$m_3 = \textsc{avg}(Delay \mid Train = 6061)$$

that compute the average delay for three different trains. Here, *Train* is the conditioning attribute, *Delay* is the metric attribute, AVG is the global metric and the keys are 5437, 2891, and 6061, respectively. Figure 2 shows the result of evaluating these CMs on a toy dataset consisting of relations R_1–R_7 that comprise historical data, as well as a to-be-ingested batch B. □

We note that CMs naturally generalize to the setting where the conditioning is done over multiple attributes instead of a single one. For simplicity of exposition and investigation, however, we focus on CMs with a single conditioning attribute in this paper.

Data unit tests. A *CM-based data unit test* (hence-forth simply called unit test) ϕ is a pair (m, c) where m is a CM and $c : \mathbb{R} \to$

{true, false} is a function that classifies the range of values produced by m into expected values ($c(m(R)) = \text{true}$) and anomalies ($c(m(R) = \text{false})$. A relation R *satisfies* ϕ, denoted $R \models \phi$, if $c(m(R)) = \text{true}$ or $m(R) = \bot$. Otherwise, R *fails* ϕ; we also say that ϕ *rejects* R.

Note in particular that R vacuously satisfies ϕ if the key x of m is not present in the conditioning attribute in R (since $m(R) = \bot$ in that case). This makes sense because m (and, hence ϕ) is meant to only investigate the part of R where $X = x$. If such part does not occur, there is nothing to test.

Example 3.2. For an interval $[i, j]$, denote by $c_{[i,j]}$ the classifier that classifies all values in $[i, j]$ as true and all other values as false. Continuing example 3.1, let $\phi_1 = (m_1, c_{[2,10]})$ and $\phi_2 = (m_2, c_{[-4.5,7.5]})$, $\phi_3 = (m_3, c_{[2,10]})$. Let B be the to-be-ingested batch shown in Figure 2. Then $B \not\models \phi_1$, $B \models \phi_2$ and $B \models \phi_3$. □

It is important to stress that current data ingestion validation tools [7, 20, 24, 25] do not use CM-based data unit tests, but *GM-based* unit tests: these are based on global metrics. Formally, they are pairs (μ, c) with μ a global metric and $c\colon \mathbb{R} \to \{\text{true}, \text{false}\}$ a function that classifies the range of μ into expected values and anomalies.

Entities. The amount of possible CMs (and therefore associated data unit tests) can be huge as it depends on the number of values occurring in the input data that can be used as keys in conditional metrics. To limit the number of possible CMs, we will only consider CMs in which the conditioning attributes refer to real-world entities. For instance, in Table 2 *Train* and *Station* are entity attributes while *Delay* is not. This restriction to entities helps improve the relevance of data unit tests: it makes little sense for instance to monitor CMs like COUNT-DIST(*Train* | *Delay* = 8) that measure deflections of a specific delay value. Furthermore, such nonsensical CMs can correlate in unexpected ways with entity CMs, thereby unnecessarily complicating the identification of erroneous tuples.

Technically we assume that, before running our framework, each attribute in the ingestion relation schema has been flagged to be an entity attribute or not, and in the discovery phase we only derive CMs whose conditioning attribute are entity attributes. Because flagging is necessary only at the level of the ingestion relation schema, and not on the level of the actual ingestion batches, it is reasonable to expect that a domain expert can quickly flag entity attributes. If no domain expert is available, an automatic classification could be done based on attribute types (for instance, timestamps and reals are never entities). We do not further consider the problem of identifying entity attributes in this paper. From here on, we hence assume that the attributes that correspond to entities have been identified.

4 Discovering Conditional Metrics

Recall from the Introduction that our approach to data ingestion validation consists of two phases. A *discovery phase* where the objective is to automatically derive a set of CM-based data unit tests from a historical ingestion sequence, and a *monitoring and error identification phase* where these tests are used to determine whether a newly arrived ingestion batch is acceptable as well as identify erroneous tuples in the batch when that is not the case. We focus on the discovery phase in this section, and describe the monitoring phase in Section 5.

Ingestion sequences. We are interested in validating batches of data that are ingested over time into a single relation.[2] Formally, an *ingestion sequence* is a sequence $\overline{R} = R_1, \ldots, R_n$ of relations, all over the same schema. Every R_i is a batch of tuples to be ingested. In what follows, we write $\overline{R}[i]$ for R_i, $\overline{R}_{<i}$ for the prefix $R_1, \ldots, R_{i-1}$, and $\overline{R}_{\leq i}$ for $\overline{R}_{<i+1}$.

CMs on ingestion sequences. Given an ingestion sequence $\overline{R}$ and a conditional metric $m = \mu(Y \mid X = x)$, we write $m(\overline{R})$ for m applied point-wise to those elements of $\overline{R}$ that contain key x, i.e.,

$$m(\overline{R}) := m(\overline{R}[i_1]), \ldots, m(\overline{R}[i_k])$$

where $i_1 < \ldots < i_k$ and $\{i_1, \ldots, i_k\} = \{i \in [1, |\overline{R}|] \mid x \in \pi_X(\overline{R}[i])\}$. This is hence the sequence of real numbers that records how the values computed by m evolve over the batches containing $X = x$.

Example 4.1. For m_1 and m_2 as in Example 3.1 and $\overline{R}$ as in Table 2, $m_1(\overline{R}) = 12, 7, 5, 7, 7, 5, 5$, $m_2(\overline{R}) = 1.5, 1.5, 0.5, 0.5, 1, 0, 0$, and $m_3(\overline{R}) = 5, 5$. □

Support. The *support* of a CM m on $\overline{R}$ is defined as the number of batches in $\overline{R}$ containing the key of m. Formally, $\text{supp}(m, \overline{R}) := |m(\overline{R})|$. To avoid edge cases where keys occur too infrequently, we are only interested in deriving classifiers for CMs whose support is higher than some predefined *support threshold* t_{supp}.

Anomaly detection and stability. *Anomaly detection* refers to "the problem of finding patterns in data that do not conform to expected behavior" [10]. Such unexpected patterns are also referred to as anomalies, exceptions, or outliers. In our setting, the data is a sequence $m(\overline{R})$ of real values, and we ask that an anomaly detection method A uses $m(\overline{R})$ to deliver a classifier $c = A(m(\overline{R}))$ that can distinguish between expected values of m and unexpected ones (anomalies). Because each point in the data $m(\overline{R})$ is a single real value, we are hence interested in *univariate* anomaly detection methods, which are significantly less complex than multivariate methods [10].

In principle, any univariate anomaly detection method A can be used. Because we assume, however, that $\overline{R}$ (and therefore, $m(\overline{R})$) is mostly error-free, we require that the classifier c produced by A is *consistent* with this assumption: when we classify the elements of $m(\overline{R})$ according to c, we expect that most (if not all) of them are expected values. When c does not satisfy this condition, it makes little sense to use $\phi = (m, c)$ as a unit test, a such a test would already mark most batches in $\overline{R}$ itself as invalid.

Formally, we say that CM m is *stable* for $\overline{R}$ w.r.t. anomaly detection method A when the percentage of anomalies detected by the classifier $c = A(m(\overline{R}))$ in $m(\overline{R})$ is below a predefined *stability threshold* t_{stable}, i.e., when:

$$\frac{|\{i \in [1, |\overline{R}|] \mid m(\overline{R}[i]) \neq \bot, \overline{R}[i] \not\models (m, c)\}|}{|m(\overline{R})|} \leq t_{\text{stable}}.$$

For a fixed anomaly detection method A, only CMs that are stable for $\overline{R}$ w.r.t. A will be promoted to unit tests.

[2] The techniques in this paper can easily be generalized to multiple relations. We focus on one relation, however, to keep the presentation simple.

R_1			R_2			R_3			R_4			R_5			R_6			R_7			B		
Train	*Station*	*Del*	*Train*	*Station*	*Del*	*Train*	*Station*	*Del*	*Train*	*Station*	*Del*	*Train*	*Station*	*Del*	*Train*	*Station*	*Del*	*Train*	*Station*	*Del*	*Train*	*Station*	*Del*
5437	Genk	0	5437	Genk	0	5437	Genk	0	5437	Genk	1	5437	Genk	1	5437	Genk	0	5437	Genk	0	5437	Genk	0
5437	Hasselt	8	5437	Hasselt	0	5437	Hasselt	2	5437	Hasselt	3	5437	Hasselt	2	5437	Hasselt	0	5437	Hasselt	1	5437	Hasselt	0
5437	Brs N	15	5437	Brs N	10	5437	Brs N	6	5437	Brs N	7	5437	Brs N	8	5437	Brs N	7	5437	Brs N	4	5437	Brs N	0
5437	Brs C	15	5437	Brs C	11	5437	Brs C	7	5437	Brs C	10	5437	Brs C	10	5437	Brs C	8	5437	Brs C	7	5437	Brs C	0
5437	Brs Z	15	5437	Brs Z	12	5437	Brs Z	10	5437	Brs Z	12	5437	Brs Z	11	5437	Brs Z	10	5437	Brs Z	8	5437	Brs Z	0
5437	Gent	19	5437	Gent	9	5437	Gent	5	5437	Gent	9	5437	Gent	10	5437	Gent	5	5437	Gent	10	5437	Gent	0
2891	Aalst	2	2891	Aalst	0	2891	Aalst	1	2891	Aalst	1	2891	Aalst	2	2891	Aalst	0	2891	Aalst	0	2891	Aalst	3
2891	Gent	1	2891	Gent	3	2891	Gent	0	2891	Gent	0	2891	Gent	0	2891	Gent	0	2891	Gent	0	2891	Gent	2
						6061	Alken	5							6061	Alken	5						

		R_1	R_2	R_3	R_4	R_5	R_6	R_7	B
$m_1 =$	$\text{AVG}(Delay \mid Train = 5437)$	12	7	5	7	7	5	5	0
$m_2 =$	$\text{AVG}(Delay \mid Train = 2891)$	1.5	1.5	0.5	0.5	1	0	0	2.5
$m_3 =$	$\text{AVG}(Delay \mid Train = 6061)$	$\perp$	$\perp$	5	$\perp$	$\perp$	5	$\perp$	$\perp$

Figure 2: (Top) Ingestion sequence $\overline{R} = R_1, R_2, R_3, R_4, R_5, R_6, R_7$ and to-be-ingested batch B. (Bottom) Three conditional metrics m_1, m_2, and m_3, and their values on $\overline{R}$ and B.

In this paper, we consider representatives of two main classes of univariate anomaly detection methods: density-based methods, represented by *average k-nearest neighbour method* (avg-kNN); and global methods, *represented by inter quartile range method* (IQR).[3] In particular, the avg-kNN method is a contextual outlier methods and we consider it here because it has been used in prior work on data ingestion validation using global metrics [20]. It assumes that expected values occur in dense neighborhoods of each other, while anomalies occur far from their closest neighbors. The IQR method, also known as the box-plot method, is a statistical detection technique. We consider it here because it is arguably the simplest possible anomaly detection technique. It assumes that expected values are those that occur in the high probability region of the emperical distribution given by $m(\overline{R})$, while anomalies are those that occur in the low probability regions. While in general we may expect avg-kNN to outperform IQR, since it is more expressive, we will show in Section 6 that IQR nearly always performs equally well. Then, because it is both simpler to implement and more efficient to execute than avg-KNN, we find that IQR seems preferable in an actual implementation. We next formally define both methods.

avg-kNN The avg-kNN method is parameterized by two parameters: a natural number k and a percentage ρ known as the *contamination parameter*. For a sequence $\vec{v} = v_1, \ldots, v_n$ of elements of $\mathbb{R}$ the classifier avg-kNN($\vec{v}$) is determined as follows. For a real value u, let $\text{dist}(u, \vec{v})$ be the average (one-dimensional) distance between u and the k nearest neighbors in $\vec{v}$. Let t be the value at percentile $1 - \rho$ in the set of distances $\{\text{dist}(v_1, \vec{v}), \ldots, \text{dist}(v_n, \vec{v})\}$. The classifier avg-kNN($\vec{v}$) returned by avg-kNN is the function

$$\text{avg-kNN}(\vec{v}) : \mathbb{R} \to bools : x \mapsto \begin{cases} \text{true} & \text{if } \text{dist}(x, \vec{v}) \leq t \\ \text{false} & \text{otherwise.} \end{cases}$$

That is: a value x is expected if the average distance to the k nearest neighbors in $\vec{v}$ is at most t, and is an anomaly otherwise.

[3] Our experiments in Section 6.4, show that even these simpler strategies are effective, despite the existence of many more complex ones.

IQR For a sequence $\vec{v} = v_1, \ldots, v_n$ of elements of $\mathbb{R}$, define the first quartile q_1 (resp., the third quartile q_3), as the median of the smallest (resp., largest) $\lfloor n/2 \rfloor$ elements in $\vec{v}$. The *inter quartile range* of $\vec{v}$ is the difference between the two, $r := q_3 - q_1$, and the *Tukey fence* is the range $\mathit{fence}(\vec{v}) := [q_1 - 1.5r, q_3 + 1.5r]$. The classifier IQR($\vec{v}$) returned by IQR on $\vec{v}$ is the function

$$\text{IQR}(\vec{v}) : \mathbb{R} \to \{\text{true}, \text{false}\} : x \mapsto x \in \mathit{fence}(\vec{v}).$$

That is: a value x is expected if it lies in the Tukey fence, and is an anomaly otherwise.

Summary of the discovery phase. The discovery phase takes as input:

- an ingestion sequence $\overline{R}$;
- a support threshold t_{supp};
- a stability threshold t_{stable}; and
- an anomaly detection method A

and produces as output the set of all of unit tests that are t_{stable}-stable w.r.t. A and have support at least t_{supp}. This set, which we denote by Θ in what follows, serves as an input for the subsequent monitoring phase.

5 Monitoring and Error Identification

In the monitoring phase, we assume given a fixed set Θ of unit tests. Whenever a new ingestion batch B arrives, we apply all tests in Θ to it to check if B is of acceptable quality. If B satisfies all tests then it is judged of good quality, and ingestion can proceed. By contrast, as soon as B fails at least one test in Θ there is a potential data quality problem that needs to be resolved. In that case, we want to use the failed unit tests in Θ to help identify the tuples of B with potential errors.

Every failed unit test naturally identifies an error region of B, namely the subrelation of B that it conditions on. Unfortunately, the same erroneous tuples can be picked up by several unit tests, each selecting a distinct region of possibly widely varying size. Therefore, returning the regions of all failed unit tests is too coarse in the sense that it has high recall, but low precision. We illustrate this claim by means of the following example.

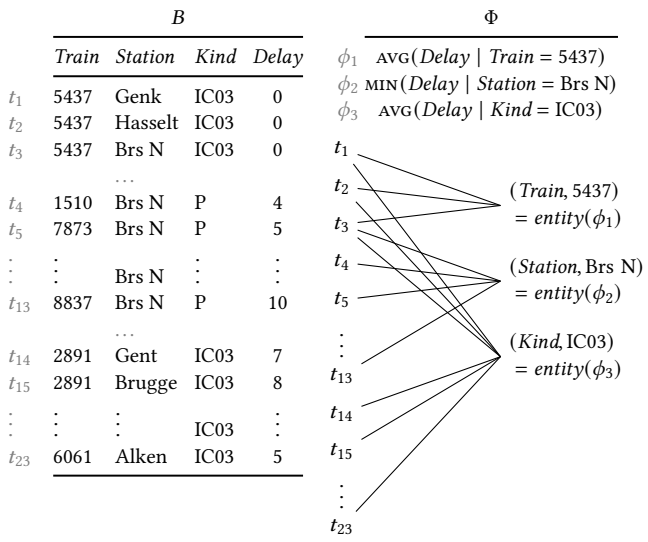


Figure 3: Illustration of a to-be-ingested batch B, failed unit tests Φ, and the corresponding entity-tuple graph.

Example 5.1. Recall the scenario outlined in Example 1.1, where due to a hardware malfunction train 5437 consistently reports zero delay. Consider the to-be-ingested batch B shown in Figure 3 as well as the failed unit tests ϕ_1, ϕ_2, and ϕ_3 listed there. We note that for the sake of this example, we have added an extra attribute *Kind* that lists the kind of service that the train offers. Unit test ϕ_1 fails because the average delay of train 5437 is now zero, which is unexpected given the historical sequence. However, in this example, the zero delay of train 5437 also causes the minimum delay in station Brussels North (Brs N) to become zero, which is also unexpected, causing ϕ_2 to fail. Similarly, it also causes the average delay of route kind IC03 to become unexpected, failing ϕ_3. The root cause here is the zero delay of train 5437, and ideally we would hence like to return only the tuples t_1–t_3 of train 5437 as suspected erroneous tuples. By contrast, the region of all failed unit tests comprises the larger set of tuples t_1–t_{23}. □

As the previous example shows, the problem is that failing unit tests may be correlated: they jointly reject B due to the same underlying set of erroneous tuples. To help identify this set of tuples, we hence desire a method to *filter* the error regions down to a subset of "representative" error regions. The tuples to report to the user for inspection are then the tuples in this subset of representative regions.

We next develop such a method, based on *ranking* the regions in failing tests such that higher-ranked regions are expected to provide a better explanation of the possible erroneous tuples than lower-ranked ones. Based on this ranking, we subsequently select the subset of representative regions.

We formalize and approach this as follows. Throughout the section, let Θ be the set of all unit tests to monitor, let B be the batch to be ingested, and let $\Phi \subseteq \Theta$ be the set of all failed unit tests.

Entities. An *entity* is a pair (X, x) with X an entity attribute and x a particular value in the domain of x. If ϕ is a unit test with CM m then we write $entity(\phi)$ for the entity (X, x) where X is the conditioning attribute of m and x is the key. Let E be the set of all entities occurring in failed unit tests, $E := \{entity(\phi) \mid \phi \in \Phi\}$. We will refer to these entities as the *failed entities* of B.

Every failed entity $e = (X, x)$ selects a non-empty set of tuples in B, which we denote by $tuples(e)$ where

$$tuples(e) := \{t \in B \mid t[X] = x\}.$$

Conversely, for a tuple t, let $entities(t)$ be the (possibly empty) set of all failed entities in which t occurs,

$$entities(t) := \{(X, x) \in E \mid t[X] = x\}.$$

For a set $C \subseteq E$, we denote by $tuples(C) := \bigcup_{e \in C} tuples(e)$. In what follows, we let $T = tuples(E)$ be the set of all tuples in B that are selected by a failed entity; we will also refer to these tuples as the *failed tuples* of B.

Entity-tuple graph. The failed entities define error regions in B. Rather than simply returning the set of all failed tuples as being potentially erroneous, we wish to return a more fine-grained set of tuples based on the structure of correlations between the failed entities. To model these correlations, we introduce the following graph structure. The *entity-tuple* graph is the bipartite graph G that has the failed tuples and failed entities as nodes, and where there is an undirected edge between tuple t and entity e if $t \in tuples(e)$ (or, equivalently, $e \in entities(t)$). See Figure 3 for an illustration.

Two entities e_1 and e_2 are *directly correlated* if there is a tuple t that is selected by both, causing the existence of the path $e_1 - t - e_2$ in the entity-tuple graph. Entities that are not directly correlated may still be *indirectly* correlated because of paths of longer length. For example, in a path of the form $e_1 - t - e_2 - u - e_3$, the entities e_1 and e_3 are indirectly correlated through e_2. In this sense, every connected component of the entity-tuple graph hence represents a subset of failed entities that are (directly or indirectly) correlated. Entities in different connected components are uncorrelated.

Ranking. Intuitively, we want to exploit the correlation topology exhibited within a connected component C of G to rank C's entities according to their "relevance" such that highly-ranked entities provide a better representation of why the unit tests associated to C failed on B.

But, how can we measure this "relevance"? Intuitively, this seems similar to that of existing measures for quantifying the authority of a node in a graph, such as, e.g., node degree, closeness centrality [12], betweenness centrality [11], PageRank [8], and HITS [15]. As we will see experimentally in Section 6, however, these existing measures fail to satisfactorily quantify "relevance". The reason is that they are generic methods that do not take into account the specific semantics of entity-tuple graphs, where there are two kinds of nodes (entities and tuples), and direct correlation between entities is encoded by paths of length 2.

We therefore next propose two tailored ranking measures on entity-tuple graphs. The first such ranking measure is inspired by the idea of measuring node importance in normal (non-bipartite) graphs by means of node degree. Specifically, in our setting, note that the degree of a tuple t in G measures how many entities become directly correlated (through t). For an entity e, we can hence use the average degree of the tuples it connects to as a proxy for measuring how correlated e is to other entities, where higher values indicate

more correlation. Formally, we define the *average tuple degree of an entity e* as

$$atd(e) := \frac{\sum_{t \in tuples(e)} |\, entities(t)|}{|\, tuples(e)|}.$$

While the average tuple degree of e hence measures how many entities (on average) become directly correlated through a tuple $t \in tuples(e)$, it does not take into account whether different tuples cause different sets of entities to become directly correlated. Yet, intuitively, when all tuples $t \in tuples(e)$ have distinct entity sets $entities(t)$ then this indicates a richer nucleus of correlations than when each tuple has the same set $entities(t)$. Our second measure takes this distinction into account. Formally, for a given entity e, we define its *correlation density* by:

$$cd(e) := \frac{|\{entities(t) \mid t \in tuples(e)\}|}{|\, tuples(e)|}$$

This hence measures how many different entity sets $entities(t)$ are exhibited by e's neighbors in G, normalized by $|\, tuples(e)|$ which is the maximum number of such sets. High values of $cd(e)$ indicate that a larger fraction of the neighboring tuples t of e have distinct sets $entities(t)$—which we take as a proxy for richer correlation structure and therefore more relevance—while lower values of $cd(e)$ indicate that more tuples share the same sets $entities(t)$, and hence poorer correlation structure.

Example 5.2. To illustrate *cd*, reconsider Example 5.1 and Figure 3. Denote e_T = (Train, 5437), e_S = (Station, Brs N) e_K = (Kind, IC03). Then, we have

$$cd(e_T) = \frac{2}{3} > cd(e_S) = \frac{2}{11} > cd(e_K) = \frac{3}{13}$$

The correlation structure can be interpreted as follows: $tuples(e_T) = \{t_1, t_2, t_3\}$ with $entities(t_1) = entities(t_2) = \{e_T, e_S\}$ and $entities(t_3) = \{e_T, e_K\}$. This means that $tuples(e_T)$ is partitioned in two sets $\{t_1, t_2\}$ and $\{t_3\}$, each 'explained' by $\{e_T, e_S\}$ and $\{e_T, e_K\}$, respectively. Similarly, $tuples(e_S) = \{t_3, \ldots, t_{13}\}$ is partitioned into the sets $\{t_3\}$ and $\{t_4, \ldots, t_{13}\}$, and $tuples(e_K) = \{t_{14}, \ldots, t_{23}\}$ is partitioned into the sets $\{t_1, t_2\}$, $\{t_3\}$ and $\{t_{14}, \ldots, t_{23}\}$. The measure *cd* favors partitions into a larger number of subsets (relative to the total number of tuples selected by the entity). The underlying expectation is that when an entity e is the root cause (like e_T in this example), more entities will interfere in $tuples(e)$. That is, $tuples(e)$ will be 'explained' by a larger number of (unique) entity sets, relative to $|\, tuples(e)|$.

Filtering. The measures introduced above allow us to rank entities on relevance, where higher values indicate more relevance. In practice, it does not suffice to only rank entities: we must also determine, based on this ranking, which entities we consider to select potentially erroneous tuples: it is these tuples that should be further inspected by a human expert.

We use the following simple method for this purpose. Fix a ranking measure f. Let C be a connected component of the entity-tuple graph G. Rank the entities of C according to the chosen measure from high to low. Let $e_1, e_2, \ldots, e_n$ be this ranking (where entities with high measure values appear first). For every $1 \leq i < n$, let $gap(i) = f(e_i) - f(e_{i+1})$. Then return only the entities $e_1, \ldots, e_i$ where i is the smallest index that maximizes $gap(i)$. We denote the latter set by $gap(f, C)$.

This procedure is repeated for every connected component of G. Note that we hence select at least one entity for every connected component. The set $\text{SET}(B)$ of *suspected erroneous tuples* of batch B is then defined as

$$\text{SET}(B) := \bigcup \{tuples(e) \mid e \in gap(f, C), \; C \text{ a connected component of } G\}.$$

Summary of the monitoring phase. The monitoring phase requires the following input:

- an ingestion batch B;
- a set of unit tests Θ; and,
- a ranking measure f.

and produces as output the set

- the set $\Phi(B) \subseteq \Theta$ of all failed unit tests over B; and,
- the set of suspected erroneous tuples $\text{SET}(B)$.

	NMBS	RETAIL
Records	6 087 982	79622
Attributes	18	22
Batches	105	100
Avg batch size	70 000	796
Scenario's	4	3
Experiments	165	27

Figure 4: Datasets

	IQR	avg-kNN	% increase
N1	5.44	15.76	190
N2	6.00	19.23	220
N3	29.82	47.67	60
N4	28.80	43.91	52
R1	12.67	21.89	72
R2	241.33	299.67	24
R3	578.44	723.00	25

Figure 5: Failed CMs per anomaly detection method.

6 Evaluation

In this section we perform an extensive evaluation of the overall effectiveness of our proposed methodology. Because our methodology supports both fine-grained error signals and error identification, which are not considered in the state of the art (cf. Sections 1 and 2), there is no prior work that we can compare to directly. As such, letting SCMRF(A,f) refer to our method when using anomaly detection method A and entity ranking measure f, we focus on assessing the different choices for A and f. In addition, we compare conditional metrics to the state of the art based on global metrics.

Concretely, we first describe our testing methodology in Section 6.1 and detail the datasets and testing scenarios used in Section 6.2. We report the overall effectiveness of SCMRF(IQR,*cd*) on these datasets and scenarios in Section 6.3. We focus here on SCMRF(IQR,*cd*) because our analysis in later sections will show that this is the best method for a stability threshold of 5%. We then proceed to assess the different choices for A (in Section 6.4) and f (Section 6.5). Finally, we compare to global metrics in Section 6.6.

6.1 Evaluation Methodology

We employ the following experimental setup. We evaluate our framework on the two datasets and seven synthetic error scenarios that are detailed in Section 6.2. For each dataset, we consider the prefix of the first ℓ ingestion batches to comprise the historical ingestion sequence $\overline{R}_{\leq \ell}$, from which data unit tests are derived using the method of Section 4. The batch $\overline{R}[\ell + 1]$ is then used as the *test* batch, i.e., the batch that needs to be validated and where

potential errors need to be identified. Because none of the datasets is labeled with a ground-truth of errors, we create a *modified version* of the test batch, referred to as B_{mod}. That is, we manipulate $\overline{R}[\ell+1]$ by synthetically generating errors based on the different error scenarios, which capture various kinds of errors that may occur. For each scenario, we also specify the set of modified tuples, which we denote by $\mathrm{MT}(B_{mod})$.

Recall from Section 5 that $\mathrm{SET}(B_{mod})$ denotes the set of suspected erroneous tuples. We employ precision, recall and F1-score as performance metrics to gauge how well $\mathrm{SET}(B_{mod})$ corresponds to $\mathrm{MT}(B_{mod})$. These are defined as follows (S refers to the selected set, R to the relevant set):

$$\text{precision} = \frac{|R \cap S|}{|S|}, \quad \text{recall} = \frac{|R \cap S|}{|R|}, \quad \text{F1} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

with $S = \mathrm{SET}(B_{mod})$ and $R = \mathrm{MT}(B_{mod})$.

Golden standard. It is well-known that the F1-score always lies in the interval [0,1] with values closer to 1 being better and F1= 1.0 serving as the golden to-achieve standard. In our context, however, such a perfect F1-score cannot always be obtained. Indeed, by design and definition, $\mathrm{SET}(B_{mod})$ is always a union of the form $\mathit{tuples}(e_1) \cup \cdots \cup \mathit{tuples}(e_n)$ for different entities e_i, whereas $\mathrm{MT}(B_{mod})$ does not need to be such a union of entities. For instance, in scenario N1 (see Section 6.2.1), p% of the tuples of a *single* entity of B are modified to obtain B_{mod}. As such, when $p < 100\%$ there is no set of entities (and hence, no set of conditional metrics or unit tests) that can select *exactly* those modified tuples. Towards a fair interpretation of F1-scores, the golden standard to be used here is therefore not F1=1.0, but the maximal F1-score that can be achieved by any union of entities. Formally, we define $\mathrm{F1}^{\max}$ as the maximal F1-score over all subsets $C \subseteq E$ of failed entities where the selected set is $S := \mathit{tuples}(C)$. When the F1-score of $\mathrm{SET}(B_{mod})$ is equal or close to $\mathrm{F1}^{\max}$ it means that no other choice of ranking measure (or filtering) can improve SCMRF. Unfortunately, it is intractable to compute $\mathrm{F1}^{\max}$. Therefore, we approximate $\mathrm{F1}^{\max}$ for each experiment via simulating annealing over the search space of subsets of failed entities. In particular, we generated for each experiment 5000 random entity sets as starting points on which we performed 10000 iterations that modify entity sets to improve the F1-score.

6.2 Datasets and scenarios

Basic statistics of the two datasets are given in Figure 4; their schemas are given in Table 2.

6.2.1 NMBS dataset. The NMBS dataset is a real-world dataset containing event data from the Belgian railway network[4], and corresponds to the train monitoring use case explained in Example 1.1. Each tuple registers the stop of a train at a particular station (attribute `name_of_stop`), together with information concerning its delay (if any), the track in the station where the train stopped, and an identification of the endpoints of the journey that the train is making (attribute `relationship`). The dataset contains only records of normal weekdays (i.e., weekends and holidays are omitted). In all related experiments, we set $\ell = 70$ and $t_{\text{supp}} = 7$. In particular, $t_{\text{supp}} = 10\%\ell$, so a key needs to occur in 10% of the historical ingestion batches to be considered as a candidate for unit testing.

[4]http://www.nmbs.be

In the scenarios described next, we use the following parameters: $K \in \{1, 5, 10\}$ and $p \in \{0.5, 0.75, 1\}$, except for $N2$ where we use $p \in \{0.5, 0.75\}$. For every choice of parameter p and K we perform 5 experiments. Table 4 details the overall number.

Scenario N1: setting delay to zero. This error scenario corresponds to that of Example 1.1. Specifically, we pick K train numbers at random from those occurring in the test batch, and set p% of their stops to report zero delay for the *delay_at_arrival* attribute.

Scenario N2: deleting trains. We wish to gauge how well conditional metrics can detect that certain (expected) tuples are missing. Specifically, we select K trains at random from those occurring in the test batch, and delete p% of their records (i.e. stops). We do not consider the case $p = 100\%$ where we remove all the records.

Scenario N3: changing stops. We pick K train numbers at random from those occurring in the test batch, and for each such train n, set p% of their stops to occur in train station s_n, where s_n is drawn randomly from the stations occurring in the test batch.

Scenario N4: swapping trains. In this scenario, we pick K train numbers and swap a train number s at random from those occurring in the test batch. For each picked train, we replace in p% of their stops the train number with that of s.

In scenarios N1 and N3 and N4, we define $\mathrm{MT}(B_{mod})$ to consist of the tuples corresponding to the selected stops. The scenario N2 is different in that all modified tuples are removed. Because those tuples are no longer present in B_{mod} and because $\mathrm{SET}(B_{mod})$ is always a subset of B_{mod}, we necessarily always have an F1-score and $\mathrm{F1}^{\max}$ of zero. To evaluate this scenario, we therefore report precision, recall and F1 on the level of entities rather than tuples. That is, we compute these metrics with respect to the *suspected erroneous entities* of batch B_{mod}, $\mathrm{SEE}(B_{mod}) := \{e \mid e \in \mathit{gap}(f, C), C$ a connected component of $G\}$, and the set of modified entities in N2 consisting of the selected trains.

6.2.2 RETAIL dataset. This dataset corresponds to the use case of Example 1.2. The Retail dataset (RETAIL) contains price data from an online store "scraped" from its website. [5] Each tuple corresponds to a retail item and registers information about its price, manufacturer, category, and brand, among others. Distinct tuples represent distinct items (i.e., each item = 1 tuple). The original dataset contains a snapshot of the website on a single date. We use this as a starting point to generate a historical sequence consisting of 100 batches. Each new batch in the sequence is generated from the previous one through the following probabilistic process. For each item there is a 5% probability that the price is altered by a random amount between $[-30\%, 30\%]$ of the original. This range is also used for the *num_images* and *weight* attributes, with a respective 5% and 0.1% probability of being modified. When the price for a specific item is changed by at least 20%, then the *is_sale* attribute for that item is set accordingly. There is a 1% probability that a row is deleted (indicating that the item is no longer available). The deleted rows have a 50% probability of being re-added in future batches. Finally, to simulate small mistakes, there is a 1% probability for both the *brand* and *category* values to be swapped with some (randomly selected) row. In all related experiments, we set $\ell = 100$ and use

[5]The original dataset is available at https://www.kaggle.com/datasets/datafiniti/electronic-products-prices.

NMBS

Attr	*Type*	*Entity*
dept_date	ts	no
train_nb	int	yes
delay_at_arrival	real	no
delay_at_dept	real	no
date_planned_arrival	ts	no
date_planned_dept	ts	no
date_real_arrival	ts	no
date_real_dept	ts	no
hour_real_arrival	ts	no
hour_real_dept	ts	no
hour_planned_arrival	ts	no
hour_planned_dept	ts	no
relationship	text	yes
operator	text	yes
track_of_dept	text	yes
track_of_arrival	text	yes
direction_of_relationship	text	yes
name_of_stop	text	yes

RETAIL

Attr	*Type*	*Entity*
id	text	yes
price_availability	bool	no
prices_condition	text	no
prices_currency	text	no
prices_issale	bool	no
prices_merchant	text	yes
prices_shipping	text	no
price	int	no
prices_dateseen	int	no
asins	text	yes
brand	text	yes
categories	text	yes
ean	int	yes
manufacturer	text	yes
manufacturernumber	text	yes
name	text	yes
primarycategories	text	yes
upc	int	yes
weight	text	no
num_images	int	no
num_urls	int	no
num_keys	int	no

Table 2: Schema of NMBS and RETAIL datasets.

the same values for $t_{\text{supp}} = 7$. In the scenario's below, MT(B_{mod}) consists of the tuples corresponding to the selected items.

Scenario R1: setting price of an item. We pick 5 items at random from those occurring in the test batch, and set their price to a value of 10, 25, or 100. These are values that occur in the dataset but not for the selected items. We run each experiment once for each price value for three different sets of items resulting in a total of 9 experiments. Note that, in the terminology of Section 6.2.1, $p = 100\%$ as there is only 1 tuple for each item.

Scenario R2: setting price of a brand. We pick 6 brands at random and change the price of $p\%$ of their items as in R1, for $p \in \{0.5, 0.75, 1.0\}$. We made sure to test brands of multiple sizes and randomly picked 2 small brands (< 10 items), 2 medium brands ($10 - 100$ items) and 2 large brands (> 100 items). In total 9 experiments were run for this scenario.

Scenario R3: setting price of a category. This scenario is similar to *R2*, where brands are exchanged for categories.

6.3 SCMRF(IQR,*cd*)

We discuss the effectiveness of the best method SCMRF(IQR,*cd*) when $t_{\text{stable}} = 0.05$. Figure 6 displays the per-scenario precision, recall and F1-score, reported as an average over all experiments in the considered scenario. Specifically, SCMRF(IQR,*cd*) attains an average F1 ≥ 0.79 in scenarios N1, N2, R2 and R3, and attains F1 ≥ 0.64 in N3 and R1. The ratio between the F1-score attained by SCMRF(IQR,*cd*) and $\text{F1}^{\max}$ is 95% (for N1), respectively 99% (N2), 82% (N3), 85% (R1), 90% (R2) and 90% (R3).

The raw F1-scores demonstrate that SCMRF(IQR,*cd*) is capable of reliably detecting a significant amount of errors in the studied scenario's. The ratio w.r.t. $\text{F1}^{\max}$ shows that within the proposed method SCMRF, the choice of IQR and *cd* is almost optimal.

SCMRF(IQR,*cd*) is less effective on scenario N4, with a reported F1-score of 0.40 and a $\text{F1}^{\max}$ of 0.82. Recall that in N4, we pick K train numbers $t_1, \ldots, t_K$, and modify $p\%$ of their stops by changing attribute train_nb with a randomly drawn train number s. In this scenario, MT(B_{mod}) contains only the $p\%$ modified stops after

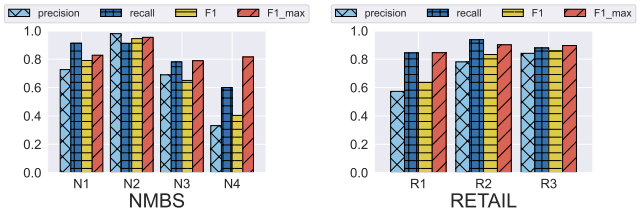


Figure 6: Performance of SCMRF(IQR,*cd*).

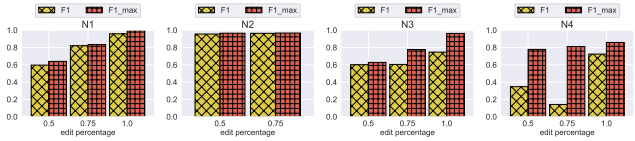


Figure 7: SCMRF(IQR,*cd*) relative to edit percentage.

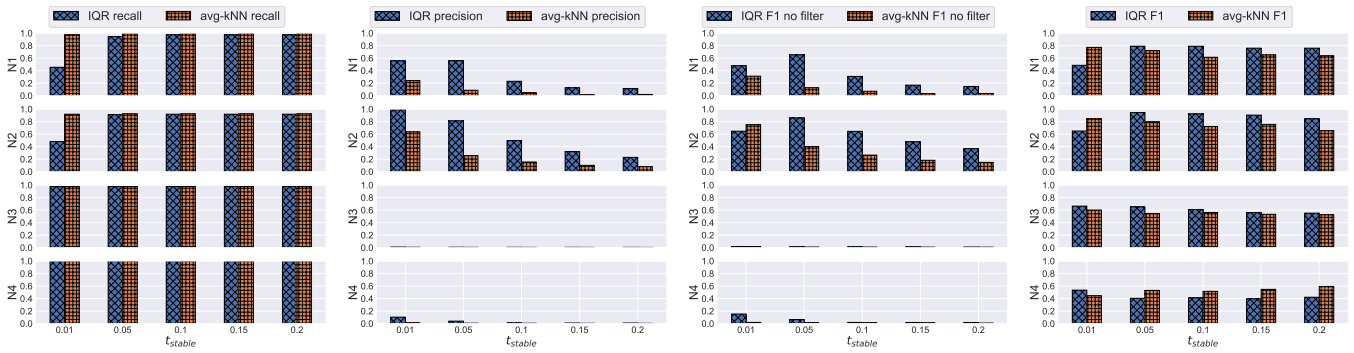
modification, which all have train number s. To successfully identify MT(B_{mod}), SCMRF(IQR,*cd*) will hence have to rank high the entity (train_nb, s). By contrast, through a detailed investigation of the N4 experiments, we find that *cd* consistently ranked the entities (train_nb, t_1), ..., (train_nb, t_k) higher than (train_nb, s). Note that the former select the tuples of the trains that were modified; in particular, on B_{mod} they select the $1-p\%$ stops that were left unmodified. At an intuitive level, by returning the tuples of these entities, SCMRF(IQR,*cd*) hence correctly indicates that there is something wrong with train numbers $t_1, \ldots, t_k$. Since these tuples do not belong to MT(B_{mod}), however, this explains the low precision that we obtain in this scenario. This analysis is confirmed by the graphs in Figure 7, which show the F1-score on the NMBS scenarios relative to the edit percentage p: for N4 we see that when $p = 100\%$, and there are hence no unmodified stops left, SCMRF(IQR,*cd*) attains an F1-score that is close to optimal. For lower values of p, the F1-score is significantly lower. Because of the mismatch between selected and modified tuples, we conclude that performance measures on N4 are overly pessimistic and do not provide the complete story. In the sequel, we therefore also separately discuss the case for N4 where $p = 100\%$ as a more reliable proxy for N4. By contrast, for N1, N2, and N3, F1 and $\text{F1}^{\max}$ are very close to relatively close over all edit percentages, implying that SCMRF(IQR,*cd*) can consistently find errors.

Conclusion: SCMRF(IQR,*cd*) finds a significant part of the errors in the majority of the scenario's (F1 ≥ 0.79). Comparison with $\text{F1}^{\max}$ reveals that within the proposed method SCMRF, the choice of IQR as anomaly detection method and *cd* as ranking measure is close to optimal.

6.4 Discovery phase

We evaluate the impact of the stability threshold and the anomaly detection mechanism. As stated before, we see IQR and avg-kNN as exemplars of global and density-based techniques, respectively. Our experiments demonstrate that even these simpler strategies are effective, despite the existence of many more complex ones.

Figure 8a–8c display recall, precision and F1 for NMBS at the various stability threshold levels when no filtering is applied to the set of failed entities E. That is, when the set of suspected erroneous



(a) Recall without filtering **(b) Precision without filtering** **(c) F1-score without filtering** **(d) SCMRF(IQR,*cd*) and SCMRF(avg-kNN,*cd*)**

Figure 8: Comparison of stability threshold selection and anomaly detection methods for NMBS.

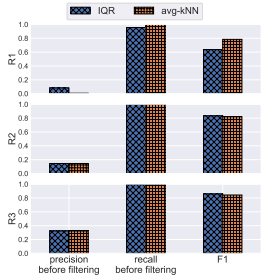


Figure 9: SCMRF(IQR,*cd*) and SCMRF(avg-kNN,*cd*) on RETAIL for $t_{\text{stable}} = 0.05$.

tuples is $tuples(E)$ and recall, precision and F1-score are hence computed with the selected set $S = tuples(E)$. The graphs for RETAIL are similar and therefore omitted.

Figure 8a shows that when $t_{\text{stable}} > 0.01$, both anomaly detection methods have (almost) perfect recall. This is crucial since the error identification stage only takes into account entities arising from failed unit tests. In other words, if an error is not discovered by the anomaly detection stage, it cannot be selected by the filter step. Figure 8b shows that the anomaly detection methods by themselves lack precision. Indeed, the overall precision for N3 and N4 is negligible at all stability thresholds, while it is much higher for N1 and N2 but decreases when the stability threshold increases.[6] Figure 8c, which displays the F1-score, hence emphasizes the need of a filtering step. In addition, we note that without filtering, the global strategy (IQR) performs better than the density-based method (avg-kNN).

Figure 8d and Figure 9, show the performance for SCMRF(IQR,*cd*) and SCMRF(avg-kNN,*cd*) in the presence of filtering by means of *cd*. Contrasted with the no-filtering situation, we see that for both anomaly detection methods, filtering results in a (significantly) higher F1-score for all scenario's when $t_{\text{stable}} > 0.01$, while the F1-score is the same or better when $t_{\text{stable}} = 0.01$.

Let us next contrast IQR and avg-kNN with respect to their end-to-end performance. From Figure 8d, we can see that IQR outperforms avg-kNN for N1, N2, N3, R2, and R3 while the situation is reversed for N4 and R1. At its core, the superior performance of avg-kNN in the latter two scenarios is explained by the fact that avg-kNN returns significantly more failed entities than IQR, as reported in Figure 5. By detailed investigation of the N4 experiments we found that this higher number of failed entities generates a more interconnected entity-tuple graph when compared to the entity-tuple graph generated by IQR. This causes entities to have different *cd* values which, in this specific scenarios, allows to rank $(\text{train\_nb}, s)$ which captures the modified tuples higher than the entities $(\text{train\_nb}, t_1), \ldots, (\text{train\_nb}, t_k)$ which capture the non-modified tuples of partially modified trains. (See also the discussion of the N4 experiment at the end of Section 6.3.)

For scenario R1, the difference in F1-score between SCMRF(IQR,*cd*) and SCMRF(avg-kNN,*cd*) is partly explained by the higher number of failed entities returned by avg-kNN. Indeed, observe from Figure 9 that the recall for IQR without filtering is 95% while that for avg-kNN is 100%.

Conclusion: While the larger number of failed entities that are returned by avg-kNN is beneficial for N4 and R1, we are of the opinion that this is undesirable in general: returning more entities overall does not increase recall, but only makes the filtering step more challenging (as witnessed by N1, N2, N3, R2, and R3). In addition, it is computationally much cheaper to compute stable CMs for IQR than for avg-kNN. As such, overall we prefer IQR as an anomaly detection method. For IQR the optimal stability threshold is 0.05.

6.5 Monitoring and error identification phase

We evaluate the effectiveness of several ranking metrics and take into account measurements that are often used to assess the importance of nodes in a graph in addition to the measures *cd* and *atd* as defined in Section 5. Specific measures we consider are PageRank [8], HITS [15], betweenness centrality [11], closeness centrality [12], (bipartite) degree centrality (the fraction of nodes a node is connected to), and the degree of a node. For HITS we display the authority score, which in our tests we found to be marginally superior to the hub score. Figure 10 and Figure 11 display the F1-score on the various scenario's for SCMRF(IQR,f) under different ranking measures f with $t_{\text{stable}} = 0.05$. We see that for N1 and N2, all ranking measures behave rather similarly while *cd* and *atd* perform slightly better. For N3, R1, R2, and R3 *cd* is clearly superior. For N4, the average F1-score is either low or has a high variability (e.g., closeness centrality, betweenness centrality, *cd*). As discussed in

[6] This decrease is expected: more conditional metrics are stable at higher stability thresholds, meaning that the set of erroneous entities potentially grows when t_{stable} becomes larger. Since $\text{MT}(B_{mod})$ remains constant, the precision decreases for larger t_{stable} values.

Figure 10: F1-score on NMBS per ranking measure

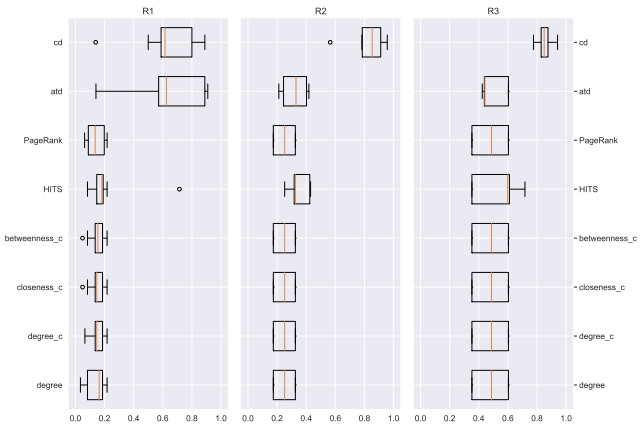


Figure 11: F1-score on RETAIL per ranking measure

Section 6.3, N4 with $p = 100\%$ provides a more reliable view. For this scenario, as can be seen in Figure 10, *cd* is clearly superior.

Conclusion: The experiments confirm that *cd* is the superior ranking measure.

6.6 Comparison with global metrics

Recall from Section 3, that global metrics differ from conditional ones in that they do not take a subset but a whole column into account. In this section, we evaluate the ability of global metrics to identify errors in the different scenarios and consider the metrics in Table 1. Each relation R thus induces a d-dimensional feature vector $v(R)$ from the d global metrics (in this paper, $d = 17$). The data quality problem then reduces to deciding whether for an ingestion batch B, $v(B)$ is an outlier w.r.t. $v(\overline{R})$ for a given historical sequence $\overline{R}$. If so, then B is said to be erroneous and therefore rejected. It should be noted that this approach operates on the granularity of the batch which is accepted or rejected as a whole and does not supply a direct method for identifying suspected erroneous tuples within a rejected batch.

Redyuk et al. [20] propose to use avg-kNN with the Euclidean distance metric, $k = 5$, and a contamination parameter of $\rho = 0.01$, as an outlier detection method. We implemented this method and tested it on NMBS and RETAIL. We found that for NMBS, only 3 of the 165 experiments, the modified batch was rejected. This is not very surprising as global metrics are not well-suited for identifying fine-grained errors, i.e., errors that occur only in a specific (potentially small) part of the batch. For RETAIL, in contrast, all R2 and R3 experiments were flagged as erroneous. Again, this is not very surprising as in these scenarios a much larger amount of rows are modified (12-18%). For R1, 4/9 experiments were flagged as outliers. Even though, many batches in RETAIL where discovered to be erroneous, we reiterate that techniques based on global metrics offer no direct way to identify suspected erroneous tuples and only offer a decision on the level of the whole batch.

Conclusion: Our approach can detect fine-grained errors that can not be detected by global metrics and in addition offers a direct way to identify suspected erroneous tuples.

7 Conclusion

We presented the SCMRF framework based on stable conditional metrics as an approach for fine-grained data quality validation that in addition aids in identifying suspected erroneous tuples. We showed that correlation density, a novel ranking measure, in combination with IQR as an anomaly detection method results in a high F1-score for erroneous tuple detection in the various scenario's. Furthermore, we argued that this combination is close to optimal within the proposed framework: no other ranking measure can improve much over *cd*. Finally, we showed that errors over ingestion sequences can be detected that are missed by uniform metrics. The latter underscores that conditional metrics can be a valuable addition to the more traditional but coarser global metrics.

We view this work as an initial exploration of the potential of conditional metrics as a tool for error detection in ingestion sequences. While the initial results are most promising, various challenges remain that can be addressed in follow-up work: (1) How to improve the filtering step? More specifically, what methods improve over the employed 'gap method' to choose an optimal cut off point in the ranking of entities? (2) Rather than naive enumeration, what are principled ways to more efficiently discover stable conditional metrics? (3) Minimize the number of data unit tests for validation while minimally sacrificing accuracy.

Acknowledgments

We thank Kris Luyten for many helpful discussions on the material presented in this paper and Brecht Vandevoort for comments on a previous version of this paper. S. Vansummeren was supported by the Bijzonder Onderzoeksfonds (BOF) of Hasselt University under Grant No. BOF20ZAP02. This work is partially funded by the Research Foundation – Flanders (FWO-grant G055219N). The resources and services used in this work were provided by the VSC (Flemish Supercomputer Center), funded by the Research Foundation – Flanders (FWO) and the Flemish Government.

References

[1] Ziawasch Abedjan, Xu Chu, Dong Deng, Raul Castro Fernandez, Ihab F. Ilyas, Mourad Ouzzani, Paolo Papotti, Michael Stonebraker, and Nan Tang. 2016. Detecting Data Errors: Where are we and what needs to be done? *Proc. VLDB Endow.* 9, 12 (2016), 993–1004. https://doi.org/10.14778/2994509.2994518

[2] Ziawasch Abedjan, Lukasz Golab, and Felix Naumann. 2015. Profiling relational data: a survey. *VLDB J.* 24, 4 (2015), 557–581. https://doi.org/10.1007/s00778-015-0389-y

[3] Ziawasch Abedjan, Lukasz Golab, Felix Naumann, and Thorsten Papenbrock. 2018. *Data Profiling.* Morgan & Claypool Publishers. https://doi.org/10.2200/S00878ED1V01Y201810DTM052

[4] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of databases.* Vol. 8. Addison-Wesley Reading.

[5] Denis Baylor, Eric Breck, Heng-Tze Cheng, Noah Fiedel, Chuan Yu Foo, Zakaria Haque, Salem Haykal, Mustafa Ispir, Vihan Jain, Levent Koc, Chiu Yuen Koo, Lukasz Lew, Clemens Mewald, Akshay Naresh Modi, Neoklis Polyzotis, Sukriti Ramesh, Sudip Roy, Steven Euijong Whang, Martin Wicke, Jarek Wilkiewicz, Xin Zhang, and Martin Zinkevich. 2017. TFX: A TensorFlow-Based Production-Scale Machine Learning Platform. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Halifax, NS, Canada) *(KDD '17)*. Association for Computing Machinery, New York, NY, USA, 1387–1395. https://doi.org/10.1145/3097983.3098021

[6] Joos-Hendrik Boese, Valentin Flunkert, Jan Gasthaus, Tim Januschowski, Dustin Lange, David Salinas, Sebastian Schelter, Matthias W. Seeger, and Bernie Wang. 2017. Probabilistic Demand Forecasting at Scale. *Proc. VLDB Endow.* 10, 12 (2017), 1694–1705. https://doi.org/10.14778/3137765.3137775

[7] Eric Breck, Neoklis Polyzotis, Sudip Roy, Steven Whang, and Martin Zinkevich. 2019. Data Validation for Machine Learning. In *Proceedings of Machine Learning and Systems 2019, MLSys 2019, Stanford, CA, USA, March 31 - April 2, 2019*, Ameet Talwalkar, Virginia Smith, and Matei Zaharia (Eds.). mlsys.org. https://proceedings.mlsys.org/book/267.pdf

[8] Sergey Brin and Lawrence Page. 1998. The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Comput. Networks* 30, 1-7 (1998), 107–117. https://doi.org/10.1016/S0169-7552(98)00110-X

[9] Emily Caveness, Paul Suganthan G. C., Zhuo Peng, Neoklis Polyzotis, Sudip Roy, and Martin Zinkevich. 2020. TensorFlow Data Validation: Data Analysis and Validation in Continuous ML Pipelines. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo (Eds.). ACM, 2793–2796. https://doi.org/10.1145/3318464.3384707

[10] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly Detection: A Survey. *ACM Comput. Surv.* 41, 3, Article 15 (jul 2009), 58 pages. https://doi.org/10.1145/1541880.1541882

[11] Linton C. Freeman. 1977. A set of measures of centrality based on betweenness. *Sociometry* 40, 1 (1977), 35–41.

[12] Linton C. Freeman. 1979. Centrality in networks Conceptual clarification. *Social Networks* 1, 3 (1979), 215–239.

[13] Alireza Heidari, Joshua McGrath, Ihab F. Ilyas, and Theodoros Rekatsinas. 2019. HoloDetect: Few-Shot Learning for Error Detection. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska (Eds.). ACM, 829–846. https://doi.org/10.1145/3299869.3319888

[14] Ihab F. Ilyas and Xu Chu. 2019. *Data Cleaning.* ACM. https://doi.org/10.1145/3310205

[15] Jon M. Kleinberg. 1999. Authoritative Sources in a Hyperlinked Environment. *J. ACM* 46, 5 (1999), 604–632. https://doi.org/10.1145/324133.324140

[16] Zifan Liu, Zhechun Zhou, and Theodoros Rekatsinas. 2022. Picket: guarding against corrupted data in tabular data during learning and inference. *VLDB J.* 31, 5 (2022), 927–955. https://doi.org/10.1007/s00778-021-00699-w

[17] Mohammad Mahdavi, Ziawasch Abedjan, Raul Castro Fernandez, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. 2019. Raha: A Configuration-Free Error Detection System. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska (Eds.). ACM, 865–882. https://doi.org/10.1145/3299869.3324956

[18] Eliana Pastor, Luca de Alfaro, and Elena Baralis. 2021. Looking for Trouble: Analyzing Classifier Behavior via Pattern Divergence. In *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, Guoliang Li, Zhanhuai Li, Stratos Idreos, and Divesh Srivastava (Eds.). ACM, 1400–1412. https://doi.org/10.1145/3448016.3457284

[19] Neoklis Polyzotis, Steven Whang, Tim Klas Kraska, and Yeounoh Chung. 2019. Slice Finder: Automated Data Slicing for Model Validation. In *Proceedings of the IEEE Int' Conf. on Data Engineering (ICDE), 2019.* https://arxiv.org/pdf/1807.06068.pdf

[20] Sergey Redyuk, Zoi Kaoudi, Volker Markl, and Sebastian Schelter. 2021. Automating Data Quality Validation for Dynamic Data Ingestion. In *Proceedings of the 24th International Conference on Extending Database Technology, EDBT 2021, Nicosia, Cyprus, March 23 - 26, 2021*, Yannis Velegrakis, Demetris Zeinalipour-Yazti, Panos K. Chrysanthis, and Francesco Guerra (Eds.). OpenProceedings.org, 61–72. https://doi.org/10.5441/002/edbt.2021.07

[21] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. 2017. HoloClean: Holistic Data Repairs with Probabilistic Inference. *Proc. VLDB Endow.* 10, 11 (2017), 1190–1201. https://doi.org/10.14778/3137628.3137631

[22] Christopher De Sa, Ihab F. Ilyas, Benny Kimelfeld, Christopher Ré, and Theodoros Rekatsinas. 2019. A Formal Framework for Probabilistic Unclean Databases. In *22nd International Conference on Database Theory, ICDT 2019, March 26-28, 2019, Lisbon, Portugal (LIPIcs)*, Pablo Barceló and Marco Calautti (Eds.), Vol. 127. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 6:1–6:18. https://doi.org/10.4230/LIPIcs.ICDT.2019.6

[23] Svetlana Sagadeeva and Matthias Boehm. 2021. SliceLine: Fast, Linear-Algebra-based Slice Finding for ML Model Debugging. In *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, Guoliang Li, Zhanhuai Li, Stratos Idreos, and Divesh Srivastava (Eds.). ACM, 2290–2299. https://doi.org/10.1145/3448016.3457323

[24] Sebastian Schelter, Felix Bießmann, Dustin Lange, Tammo Rukat, Philipp Schmidt, Stephan Seufert, Pierre Brunelle, and Andrey Taptunov. 2019. Unit Testing Data with Deequ. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska (Eds.). ACM, 1993–1996. https://doi.org/10.1145/3299869.3320210

[25] Sebastian Schelter, Dustin Lange, Philipp Schmidt, Meltem Celikel, Felix Bießmann, and Andreas Grafberger. 2018. Automating Large-Scale Data Quality Verification. *Proc. VLDB Endow.* 11, 12 (2018), 1781–1794. https://doi.org/10.14778/3229863.3229867

[26] Laura Sebastian-Coleman. 2012. *Measuring Data Quality for Ongoing Improvement: A Data Quality Assessment Framework* (1st ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[27] Guanting Tang, Jian Pei, James Bailey, and Guozhu Dong. 2015. Mining multidimensional contextual outliers from categorical relational data. *Intell. Data Anal.* 19, 5 (2015), 1171–1192. https://doi.org/10.3233/IDA-150764

[28] Richard Y. Wang and Diane M. Strong. 1996. Beyond Accuracy: What Data Quality Means to Data Consumers. *J. Manage. Inf. Syst.* 12, 4 (March 1996), 5–33. https://doi.org/10.1080/07421222.1996.11518099

[29] Li Wei, Weining Qian, Aoying Zhou, Wen Jin, and Jeffrey X. Yu. 2003. HOT: Hypergraph-Based Outlier Test for Categorical Data. In *Advances in Knowledge Discovery and Data Mining*, Kyu-Young Whang, Jongwoo Jeon, Kyuseok Shim, and Jaideep Srivastava (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 399–410.