MaTElDa: Multi-Table Error Detection

Fatemeh Ahmadi
BIFOLD & TU Berlin
Berlin, Germany
f.ahmadi@tu-berlin.de

Marc Speckmann
Leibniz Universität Hannover
Hannover, Germany
speckmann@dbs.uni-hannover.de

Malte Fabian Kuhlmann
Leibniz Universität Hannover
Hannover, Germany
kuhlmann@dbs.uni-hannover.de

Ziawasch Abedjan
BIFOLD & TU Berlin
Berlin, Germany
abedjan@tu-berlin.de

ABSTRACT

As data-driven applications gain popularity, ensuring high data quality is a growing concern. Yet, data cleaning techniques are limited to treating one table at a time. A table-by-table application of such methods is cumbersome, because these methods either require previous knowledge about constraints or often require labor-intensive configurations and manual labeling for each individual table. As a result, they hardly scale beyond a few tables and miss the chance for optimizing the cleaning process. To tackle these issues, we introduce a novel semi-supervised error detection approach, Matelda, that organizes a given set of tables by folding their cells with regard to domain and quality similarity to facilitate user supervision. We propose a unified feature embedding that makes cell values comparable across tables. Experimental evaluations demonstrate that Matelda outperforms various configurations of existing single-table cleaning methodologies in the multi-table scenario.

1 INTRODUCTION

With the rising importance of data augmentation and data enrichment [37], there is a new interest in connecting external data repositories or previously disconnected data silos within organizations and establishing data lakes [21, 38]. However, it is also in the nature of data lakes that the sources are not apriori curated and cleaned. As incorrect data leads to incorrect and biased behavior and outcome of data-driven applications [31, 41, 42], it is crucial to ensure that the data adheres to a minimum quality level before it can be put to use [38]. Ensuring data quality in data lakes either requires employing cleaning procedures to detect and fix errors apriori or on the fly. In both cases, we are dealing with multiple datasets at the same time. The naive approach to cleaning the whole set would be to use common data-cleaning approaches on each table individually. There are many existing data cleaning systems and algorithms that can be adapted to cope with individual tables and domains [1, 3, 5, 9–11, 14, 24, 27, 41, 49, 51]. Using traditional cleaning techniques, we would have to prepare and specify rules and/or parameters for each table in a table set, which is cumbersome for multiple tables and does not scale to sets with hundreds or thousands of tables. LLM-based techniques have been explored that rely on table-specific fine-tuning of the models to achieve acceptable results [60]. However, such approaches are generally inefficient and costly on large datasets [60].

In contrast, supervised data cleaning approaches consider data cleaning as a classification problem [23, 33, 34, 43, 58]. This way, the user does not need to know about the internals of the system and generalizable rules and parameters for a dataset. The only requirement is to have insights on some instances of a dataset to provide labels and corrections. Although the state-of-the-art semi-supervised systems, such as Raha [34], need only a few labels per table, the number of required labels will scale linearly with the number of tables, which will easily exceed current labeling budgets.

In this paper, we present our methods for detecting data errors on a set of tables, aiming at reducing the overall labeling and runtime cost below the cost of applying a state-of-the-art approach sequentially on individual tables. Following a semi-supervised approach, we require the user to label a limited number of cell values from the whole set and use these labels to detect data errors in all tables. Our assumption is that we do not have a labeling budget that covers each table individually. Even if we label only one tuple per table, we will end up requiring a budget that corresponds to the potentially large number of tables, i.e., hundreds or thousands. Therefore, one challenge is to carefully pick those informative cell values for labeling that cover multiple tables simultaneously. That brings us to the second challenge of deciding when to transfer an obtained label to a different table. The correctness of each value highly depends on the corresponding table that defines its context. To generalize the user label, we need to embed all cells from different tables with varying schemata into a unified feature space.

To address the aforementioned challenges and reduce the labeling costs, we suggest to first organize the table set in a way that similar errors can benefit from the same labels. For this purpose, we designed a *two-layer folding* approach to first separate cells based on their table domains and then fold them based on quality similarity, i.e., how similarly they are judged by a common set of error detectors [32], based on a unified feature space. Then, we distribute labels so that every fold is covered and propagate the labels within the folds. Finally, we classify the correctness of each value within its own table context.

Our extensive experiments on multiple data lakes show that our system Matelda (Multi-Table Error Detection) with the unified feature space significantly outperforms competitors at detecting errors, when there are fewer labels than tables. This improvement can be attributed to Matelda's ability to fold related data points together and to propagate labels across tables. Our encoding of column relationships effectively capture functional dependency violations, even when cells are compared across different tables. Furthermore, Matelda demonstrates a significantly better runtime compared to a sequential one-table-at-a-time execution of state-of-the-art approaches. In short, we make the following contributions:

1) We introduce and define the semi-supervised error detection

problem for a multi-table scenario.
2) We present Matelda, which uses two cell folding steps to organize all cells based on domain correspondence and latent quality similarities, facilitating effective label sharing for supervised cleaning. This way, it significantly reduces the supervision effort in a multi-table scenario over the application of traditional approaches.
3) We propose a novel approach to embed cells from tables with non-overlapping schemata into a unified, quality-based feature space, which can subsequently be used for further computations. In particular, we also show mechanisms that make column dependencies, such as FDs comparable across tables with differing schemata.
4) We present a novel table set benchmark for cleaning and compare our solution to existing state-of-the-art systems highlighting the difficulties and potentials of resource sharing when dealing with more than one table at a time. We introduce new sets of synthesized datasets containing various types of errors. These datasets can facilitate more research on data quality in data lakes.

2 FUNDAMENTALS

In this section, we lay out our assumptions on the considered types of errors, we aim to detect, and we formalize the problem of detecting data errors for multiple tables, simultaneously.

2.1 Data Errors in the Single Table Scenario

Building on prior work, we consider data errors to be all values inside a table that differ from the corresponding ground truth [34]. Let us consider $t = \{\tau_1, \tau_2, ..., \tau_{|t|}\}$ as a table or relational instance, where each τ_i represents a tuple. The schema of table t is denoted as $A = \{a_1, a_2, .., a_{|A|}\}$. In this representation, $t[i, j]$ refers to the value for attribute a_j in tuple τ_i. We denote t^* as the ground truth for table t. Consequently, any value $t[i, j]$ that differs from the corresponding value $t^*[i, j]$ is considered a data error [34].

Data errors have been loosely categorized based on different criteria in the literature [3, 9]. Generally, one distinguishes syntactic and semantic errors [34]. Syntactic errors are values that are not compatible with the structure of the containing column. Semantic errors are the ones that are syntactically correct, i.e., fit the domain, but not correct in the context of the tuple [34]. Typical heuristics for discovering syntactic errors are to detect outliers, pattern violations, or rule violations. To identify semantic errors, one has to leverage relationships and dependencies that are encoded either in rules or mappings to some sort of master data [34].

Outliers: Outliers are unlikely values in given probabilistic data distributions [34]. Outlier detection algorithms use statistical methods to find these values [35]. In our running example, depicted in Figure 1, errors that we marked in Table t1, columns "Age" and "Market Value in Millions", and Table t4, column "Population" are examples of numeric outliers.

Pattern Violations: Many data columns follow well-defined patterns, such as date and time formats. Pattern violation detection algorithms detect the values that do not conform to the specified patterns. For example, in Figure 1, columns "Release Date" and "Total Gross" in Table t5 contain pattern violations.

Rule Violations: Different rules or integrity constraints, such as denial constraints (DCs), can be defined for a single dataset [34]. Rule-violation algorithms verify the data against such rules or DCs. While one can also define statistical and pattern-based rules, we specifically refer to inter-column relationships, such as Functional dependencies (FDs), when we mention rules [46]. In the running example, Table t3 contains an FD violation. There is a functional dependency between the "Club Name" and "Country". Each club's name maps to exactly one country. *Real Madrid* is in *Spain*, and in this table, there are two records for that, and in one of them, the country is *France* which is wrong.

2.2 Problem: Multi-Table Error Detection

Given a set of tables $S = \{t_1, t_2, ..., t_{|S|}\}$, the goal of error detection in a multi-table scenario is to detect all cell values in all tables that differ from the corresponding cell in the ground truth $S^* = \{t_1^*, t_2^*, ..., t_{|S|}^*\}$. Considering $t_k[i, j]$ as a value for the attribute a_j in the tuple τ_i in table k, we define the set of all erroneous values in all tables of our set as:

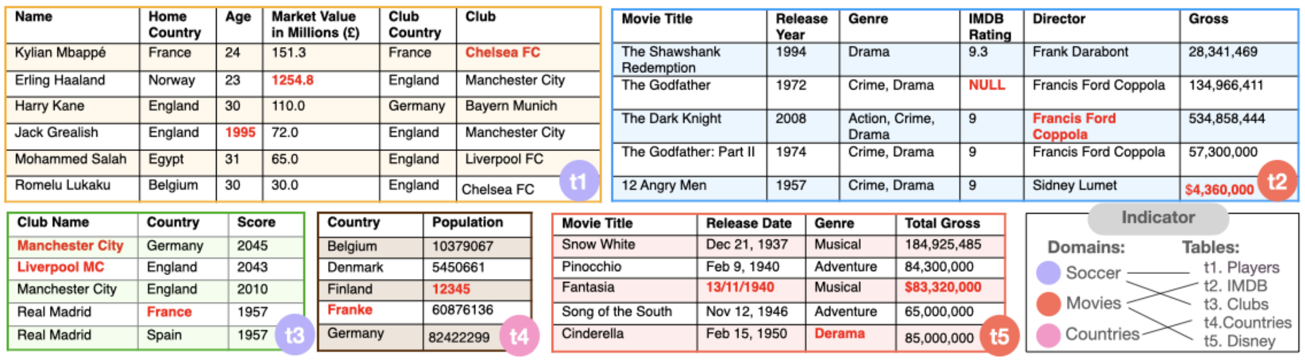
$$E = \{t_k[i, j] | \quad \forall t_k \in S. \forall t_k^* \in S^*. \exists i, j \in \mathbb{N}. t_k[i, j] \neq t_k^*[i, j]\} \quad (1)$$

The straight-forward approach to detect all errors E in the set of tables with an existing semi-supervised approach is to run the algorithm on each table individually. Each table would require its own set of user labels. The total number of required labels would then be $\Lambda = \sum_{k=1}^{|S|} l_k$, where l_k is the labeling budget for table t_k.

In this paper, we aim to develop a method that outperforms such an approach by reaching similar or better performance in terms of runtime and accuracy, while requiring a smaller labeling budget than Λ. If all datasets follow the same schema, it is generally possible to integrate them and apply a common single-dataset error detection technique on the integrated table. A more relaxed assumption is that there are some datasets that share concepts and schema information. While a full integration of such tables might not be obvious, it might be possible to connect information across such groups of tables. In particular, we want to make sure that a classifier that has been learned on labeled cells of one individual table can also predict the correctness of cells in other tables. Therefore, we need first the cells to be comparable across tables, i.e., all being represented via a unified feature vector. Therefore, we need to build a solution that covers the aforementioned three violation types without depending on the table specific metadata. Note that for each table, a different set of distributions, patterns, and rules might hold. Furthermore, we need to make sure that user labels given on individual table cells are propagated to cells with very similar characteristics. We call this *label sharing*. If the datasets are from entirely different domains, label sharing might be misleading, due to semantic heterogeneity caused by homonyms. It is thus necessary to first identify such groups of tables that have potential of label sharing through a semantically guided clustering.

2.3 Why is Raha not enough?

An obvious approach would be to simply expand the existing supervised approaches, such as Raha [34] for the multi-table error detection problem. However, such approaches have non-trivial limitations. First, Raha does not include mechanisms to learn across tables, not even across columns. Cells from different columns have different numbers of features, and the features are not comparable even across columns of the same table due to the different rules, patterns, and distributions across columns. Thus one major technical challenge is to propose a feature vector encoding that lets a single classifier as well as a single clustering algorithm treat all cells the same way. This unification does come at a cost. Getting rid of table and column specific features that



Name	Home Country	Age	Market Value in Millions (£)	Club Country	Club
Kylian Mbappé	France	24	151.3	France	Chelsea FC
Erling Haaland	Norway	23	1254.8	England	Manchester City
Harry Kane	England	30	110.0	Germany	Bayern Munich
Jack Grealish	England	1995	72.0	England	Manchester City
Mohammed Salah	Egypt	31	65.0	England	Liverpool FC
Romelu Lukaku	Belgium	30	30.0	England	Chelsea FC

t1

Movie Title	Release Year	Genre	IMDB Rating	Director	Gross
The Shawshank Redemption	1994	Drama	9.3	Frank Darabont	28,341,469
The Godfather	1972	Crime, Drama	NULL	Francis Ford Coppola	134,966,411
The Dark Knight	2008	Action, Crime, Drama	9	Francis Ford Coppola	534,858,444
The Godfather: Part II	1974	Crime, Drama	9	Francis Ford Coppola	57,300,000
12 Angry Men	1957	Crime, Drama	9	Sidney Lumet	$4,360,000

t2

Club Name	Country	Score
Manchester City	Germany	2045
Liverpool MC	England	2043
Manchester City	England	2010
Real Madrid	France	1957
Real Madrid	Spain	1957

t3

Country	Population
Belgium	10379067
Denmark	5450661
Finland	12345
Franke	60876136
Germany	82422299

t4

Movie Title	Release Date	Genre	Total Gross
Snow White	Dec 21, 1937	Musical	184,925,485
Pinocchio	Feb 9, 1940	Adventure	84,300,000
Fantasia	13/11/1940	Musical	$83,320,000
Song of the South	Nov 12, 1946	Adventure	65,000,000
Cinderella	Feb 15, 1950	Derama	85,000,000

t5

Figure 1: Running Example

might have contributed to the accuracy of the error detection model, requires us to find at least approximate replacements. Second, Raha requires the user to label a sufficient amount of cells, e.g., 20 cells per table column. The minimum amount of required labels for Raha is two labeled cells per column, which also does not scale for a setup with hundreds or more datasets. In Section 4.1.4, we propose simple variations of Raha that skip tables or columns so that the labels can be randomly assigned to table columns in a lake. These variations are able to handle situations with fewer labels than tables but perform poorly. Thus the second challenge is to identify how labels given to individual cells of a table can also be propagated to cells of other tables. For this purpose, an effective grouping strategy of tables and cells is necessary. In the next section, we will introduce Matelda and discuss the new ideas we employ to make features schema-independent and to propagate labels across tables for effective learning.

3 MATELDA

In the following, we propose Matelda, which makes such considerations to effectively find errors in a multi-table scenario. We start with an overview of Matelda's workflow and then dive deep into the different stages of folding cells for label sharing that influence the downstream error detection classification task.

3.1 General Matelda Workflow

Figure 2 illustrates the workflow of Matelda, while Alg. 1 details the process. When cleaning a set of tables with data from different domains, we have to ensure that error detection techniques do not confuse domain-dependent errors. As illustrated in the running example in Figure 1, Tables t1 and t3 are from the same domain of football, and Tables t2 and t5 are about movies. Thus, information about "year", which is prevalent in Tables t2 and t5, will follow different distributions depending on the context. Thus, Matelda first folds the cells based on their latent domain similarity (Figure 2 and Alg. 1: Step 1). We consider tables to be from the same domain if they have significant overlap regarding semantic concepts.

Once we have grouped tables that are semantically similar, the next step is to now fold cells based on their quality or dirtiness similarity (Figure 2 and Alg. 1: Step 2). This approach has already been proposed for error detection on individual tables [34]. The idea is that labels can be shared for similarly dirty values. For instance, consider the "Total Gross" column in Table t5 and the "Gross" column in Table t2, as illustrated in Figure 1. A nonessential "$" sign introduced errors in both columns. Thus, it would be

Algorithm 1: Matelda (S, Λ)

```
Input: Set of tables $S$, labeling budget $\Lambda$.
Output: Set of erroneous cell values $E$.
// Step 1: Domain-based Cell Folding
$CE \leftarrow \varnothing$ ;                         // Set of contextual embeddings
for each table $t \in S$ do
    $st \leftarrow$ serialize($t$) ;            // Concatenate rows into a string
    $ce \leftarrow$ obtain_BERT_Embedding($st$)
    $CE \leftarrow CE \cup \{ce\}$
$DFolds \leftarrow$ HDBSCAN($CE$) ;  // Each cluster is a Domain Fold $df$
// Step 2: Quality-based Cell Folding
for each $df \in DFolds$ do
    $C_{df} \leftarrow \varnothing$; // Triplets (cell value, feature vector, label)
    for each cell $c \in t$ do
        // Running Error Detectors
        $v_c \leftarrow [d_\theta(c), d_{TD}(c), d_{FD}(c), nv_{LHS}(c), nv_{RHS}(c)]$
        $C_{df} \leftarrow C_{df} \cup (c, v_c, \perp)$; // Labels are unknown still ($\perp$)
    $k \leftarrow \max(2, \Lambda \cdot \frac{|\text{columns}(df)|}{|\text{columns}(S)|})$;      // $k$ is the labeling budget
      for $df$
    $QFolds \leftarrow$ Mini-batch K-Means($C_{df}, k$)
    // Step 3: Sampling & Labeling
    for each $qf \in QFolds$ do          // Each $qf$ is a quality fold
        $(c_s, v_s, \perp) \leftarrow$ centroid($qf$) ;                  // Sampling
        $l_u \leftarrow$ user_labels ; // User labels the sampled cell value
        $c_s$.setlabel($l_u$)
        // Step 4: Label Propagation
        for each cell $c \in qf$ do
            c.setlabel($l_u$)
    // Step 5: Classification, Error Prediction per Column
    for each $cl \in columns(df)$ do
        $m \leftarrow$ GB.Train($cl$)
        $E \leftarrow E \cup$ Predict($m$, $cl$)
```

desirable to transfer the label for one of the errors to the other column. Ideally, the clean cells of both columns fall into the same quality-based cell fold while the erroneous ones fall together into a different fold. Further, consider the errors in tables t1 and t3. In both tables, there are errors in the club column that do not seem to follow a similar pattern. The erroneous team names, *Chelsea FC* and *Manchester City*, violate relationships with their adjacent columns "Club Country" and "Country", respectively. Also here, it would be ideal, if the relationship violation in both tables leads to the folding of the corresponding violating cells into the same fold. To organize values based on their data quality similarity, prior work embeds them based on signals obtained from base detectors and then clusters the cells using a similarity metric based on that embedding. Each cluster resembles a group of cells with the same latent quality [32]. We follow the same approach but take into consideration that cells are coming from different

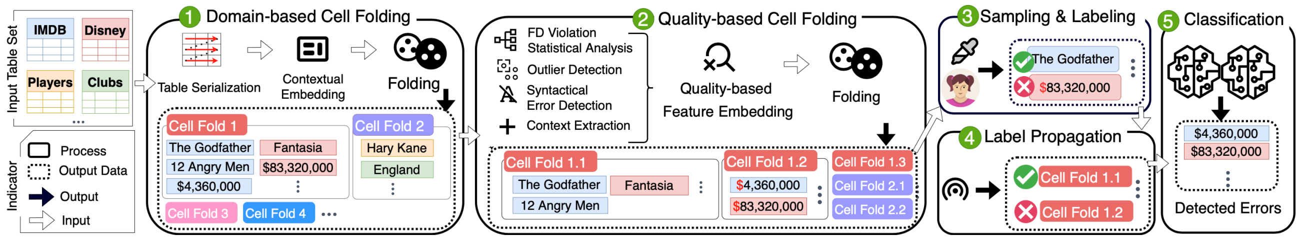


Figure 2: Matelda Workflow

tables with differing schemata. Therefore, in contrast to previous work, our base detectors cannot be column- and table-dependent. In Section 3.3, we detail how we expand on the cell featurization so that cells from different tables and columns become comparable. Once all the cells are organized in suitable folds, we draw samples from them to cover as many different types of errors as possible (Figure 2 and Alg. 1: Step 3), i.e., different typos, semantic errors, etc.. Following existing works intuition on the cluster assumption [8], we propagate those labels to the other cells of the corresponding cluster (Figure 2 and Alg. 1: Step 4). The labeled set is then used to train one classifier for each column. Finally, in Step 5, we use the trained classification models to predict the labels for all cell values of each column. In the next subsections, we describe each step in more detail.

3.2 Domain-based Cell Folding

With domain-based cell folding, we address the first concern of semantic consistency, bringing together cells from tables describing the same semantic concepts and separating those that might confuse the quality problems that are domain-dependent.

The state-of-the-art in identifying semantically similar documents and tables has advanced significantly. In fact, there are many methods from document clustering [56] and table union discovery [28, 39], that can be used for this step. As in our case, this is a coarse-granular filter and we do not believe that there is a best domain-based folding technique, we resort to a pragmatic choice here and leverage a table representation based on BERT embeddings [13, 56]. For this purpose, we serialize each table by concatenating all cell values in a row into a single string, and then concatenating all rows into a larger string (Alg. 1, Line 3). This process allows us to treat each table as a single sentence. Specifically, we use a BERT model that has been pre-trained on English documents to generate a feature vector for each table, thereby capturing its semantic characteristics (Alg. 1, Line 4).

Given that the number of domains in our table set might not be known, we require a clustering algorithm capable of automatically detecting the number of clusters. Additionally, the algorithm must be scalable to manage the entire table set. To meet these criteria, we employ HDBSCAN [7] for clustering the tables (Alg. 1, Line 6). In our implementation, a cluster must contain a minimum of two tables. Each of the outlying tables is clustered into an individual group. After the Table clustering phase, we consider each cluster a domain-fold, containing all cells from the corresponding tables.

Generally, one could replace our folding strategy with any other advanced similarity metrics and clustering algorithms. The benefit of our compositions is that the methods do not require us to apriori specify the number of clusters. In the experimental section, we compare our choice to other alternatives.

After domain-based folding, we can assume that the cells inside a cell fold are from a semantically similar domains. One could now further refine the folds based on columns. For instance, columns with identical data types are likely to exhibit similar errors. Also, columns containing numerical values are susceptible to numeric outliers, while those featuring dates may display analogous pattern violations. However, folding based on the column types introduces two disadvantages. First, it introduces an ad-hoc layer of filtering to the algorithm. Second, it impedes label sharing across potentially similarly dirty values in different columns leading to reduced effectiveness as also shown in our ablation studies. To avoid these issues, we simply encode column features into the subsequent cell folding step as we discuss in section 3.3.

3.3 Quality-based Cell Folding

After the domain-based folding, we consider each domain-fold as an independent subset of tables where labels can be shared. Next, all cells of a fold need to be embedded in a unified feature space for the subsequent training and classification steps. As previously mentioned, we use a feature embedding space based on detector signals as applied in prior research. While some of those detector signals can be directly included in our feature embedding space, there are considerations that we need to make on column- and table-dependent detector signals.

3.3.1 Cell value features. We want to embed cell values based on the signals that base detectors emit for each individual cell similar to prior work [34]. For each cell, we further encode its corresponding column and table to retain table- or column-specific relationships. The detectors are collected by automatically configuring rule-based, pattern-based, statistic-based detectors as discussed in Section 2.1 (Alg. 1, Line 10). In the following, we describe how we modify the application of each base detector to the multi-dataset scenario.

Outlier detectors: Despite folding all cells of all columns, we evaluate the outlier/inlier property of a cell only with regard to the cell values that occur in the same column. Similar to previous work [34, 35], Matelda uses Gaussian and histogram models to detect outliers. Gaussian models capture outliers with regard to the magnitude of a value, being effective in identifying numerical inconsistencies. Histogram-based models detect outliers based on their frequency. This is why histograms detect outliers in non-numeric data as well [35]. Both models require pre-defined thresholds to identify the outliers. This threshold is set based on the normalized term frequency in histogram models and the normalized distance to the mean in Gaussian models. Similar to prior work, we automatically generate multiple outlier detectors d_θ, by systematically instantiating the thresholds at different threshold levels. Formally,

$$d_{\theta_{tf}}(t[i,j]) = \begin{cases} 1 & iff \frac{TF(t[i,j])}{\sum_{i'=0}^{|t|} TF(t[i',j])} < \theta_{tf}; \\ 0 & otherwise. \end{cases} \quad (2)$$

where $\theta_{tf} \in \Theta_{tf} = \{0.1, 0.2, 0.3, ..., 0.9\}$ and $TF(t[i,j])$ is the term frequency of the specified cell value ($t[i,j]$). Also,

$$d_{\theta_{dist}}(t[i,j]) = \begin{cases} 1 & iff \frac{|t[i,j]-\mu_j|}{\sigma_j} > \theta_{dist}; \\ 0 & otherwise. \end{cases} \quad (3)$$

where $\theta_{dist} \in \Theta_{dist} = \{1, 1.3, 1.5, 1.7, 2, 2.3, 2.5, 2.7, 3\}$, μ_j is the mean, and σ_j is the standard deviation of the numerical data column $t[:,j]$.

Typo detectors: To detect pattern violations and typos, Raha uses a bag-of-characters representation to generate a character checker for each character of a data column to verify the presence of the particular character in a cell value at hand. However, if we want to apply this to our multi-table scenario, we have to consider all characters inside a domain-based cell fold individual one-hot encoded features to be able to create feature vectors with the same length for all cells. To avoid such an intractable feature space, we decided to take a more targeted approach. Among the errors on string columns, those related to patterns are detectable using the histogram-based outlier detectors, as discussed before. However, there might be spelling issues that histograms cannot cover. In a more pragmatic approach, we simply use a dictionary-based spell checker d_{TD} to detect cells with potentially arbitrary typos. In our implementation, we use "Aspell" [12] which is effective [40] and also faster than competitors, such as Norvig [1]. Aspell maintains a dictionary $Dict$ of English terms and marks any word w that is not in that dictionary as an error. Formally,

$$d_{TD}(t[i,j]) = \begin{cases} 0 & iff \ \forall w \in t[i,j]. \exists w' \in Dict. w = w' \\ 1 & otherwise. \end{cases} \quad (4)$$

For example, "Derama" in Table t5, and "Franke" in Table t4 are not present in our dictionary, and thus would be marked as errors.

Rule violation detectors: The single-dataset approach Raha considers all possible functional dependencies (FDs) with a single attribute on the left-hand side and accordingly marks violating cells of the column at hand. This approach will not work in our featurization as the cells are coming from different tables with different schemata. Thus, each cell of a fold can have a different set of FD features. To make FD violations comparable across tables, we make use of the structural similarities inspired by the similarity flooding approach [36] to identify similar FDs across tables. First, every table has a first column, which is historically the most left column and typically the key of the table. For example, considering the datasets introduced in Table 1, all tables in the Quintet dataset, 83% of the tables from GitTables, and 69% of the tables in the DGov-1K dataset, have a first column that qualifies as an ID and functionally determines all the remaining columns. Further, every column has at most two direct neighbors. According to schema suggestion approaches adjacent columns follow repetitive patterns [6]. Hence, we consider three functional dependency detectors for each column in a table. The first one is to check the functional dependency from the very first column of the table to the current under-process column. The other two possible functional dependencies are the relationship between the under-process column and its direct neighbors based on the assumption that relevant columns are positioned together in the table. Therefore, if we consider a_j as the current attribute, we will have three primary FD detectors (d_{FD}) for this column:

$$d_{a_0 \to a_j}, d_{a_{j-1} \to a_j}, d_{a_j \to a_{j+1}} \quad (5)$$

In addition to the three primary functional dependencies, we capture aggregated information about the involvement of each cell in FD violations so that the information can be expressed within a fixed set of features for all cells. In particular, we encode the relative frequency of participation of a cell in arbitrary rule-violation. We distinguish the frequencies for cases where the cell was on the right-hand side (RHS) and the left-hand side (LHS) of an FD. To capture the frequencies in a uniform way we encode relative frequencies through one-hot encoding of five uniformly picked frequency quantiles (of 20% width) per side, i.e., RHS or LHS. The normalization process for the LHS and RHS violations takes into account the specific orientation of the column within the functional dependencies, normalizing by the total number of all rules where the column is on the LHS or RHS, respectively. Formally,

$$nv_{(L/R)HS}(t[i,j]) = \frac{\#\text{(L/R)HS violations of } t[i,j]}{\#\text{Rules where col } j \text{ appears on (L/R)HS}}. \quad (6)$$

where nv_{LHS} is the relative frequency of violating rules that have column j on their LHS and nv_{RHS} is the relative frequency of violating rules that have column j on their RHS. For each $t[i,j]$, we end up with two buckets that are set to one and 8 that are set to 0, depending on where $nv_{LHS}(t[i,j])$ and $nv_{RHS}(t[i,j])$ fall.

We concatenate all these features for each cell value of each cell fold to create one feature vector. Subsequently, we use these feature vectors to perform quality-based cell folding.

3.3.2 Cell Folds and Sampling. To identify the most promising data points for labeling, we have to consider several factors. First, when considering error detection as a classification problem, we have to solve a class imbalance problem because, in most datasets, the ratio of errors is significantly smaller than the ratio of clean values. Furthermore, it is desirable to capture different types of errors during labeling. Finally, the labels should benefit multiple tables at the same time. Our quality-based cell folding gears towards facilitating the distinction of cells with different quality assessments so that labels can be drawn from cell groups with different latent quality problems. This distinction can become more and more fine-granular, the more labels we can spare, as the number of labels guides the quality-based cell folding as a clustering step. One could pick any K-Means or hierarchical clustering approach where the number of clusters can be specified. We selected Mini-batch K-Means [55], which is more efficient than hierarchical clustering that is used in prior work [32] (Alg. 1, Line 13).

Note that the number and size of domain-based folds are data-dependent and not controlled by the user (Section 3.2). Thus, we first distribute the labeling budget among the domain-based folds proportional to the number of columns they contain, enforcing that a domain-based fold receives at least two labels. Here, we consider the number of columns instead of the number of cells because the number of different columns is a more accurate proxy for estimating how many different data types and, as such, differently shaped errors we expect in a cell fold. The number of labels that were assigned to a domain-based fold will then be

[1]https://norvig.com/spell-correct.html

used to set k for the quality-based folding approach (Alg. 1, Line 12).

For each quality-based cell fold, we then pick the nearest point to the cluster center for labeling (Alg. 1, Line 15). Next, we discuss how to prepare training sets using the derived samples.

3.4 Label Propagation and Classification

As the cell clusters are established based on the similarity of the detector embeddings, the cells in each cluster are similar in terms of how they deviate from the ground truth [34]. We make use of this previously shown intuition to justify the propagation of the user-assigned label to all other cells of the same cluster. This way we obtain labeled cells for all table columns within the same domain-fold (Alg. 1, Lines 18-19). After label propagation, Matelda unfolds the different cell folds and uses one classifier for each column to predict the correctness of its cells. Similar to prior work, we use the Gradient Boosting Classifier [19] to carry out the classification, which has shown robust performance (Alg. 1, Lines 20-22).

3.5 Time Complexity

Given a table set S, with m and n being the largest possible number of columns and rows, respectively, we now describe the worst case time complexity of Matelda by iterating through the steps in Alg. 1. The initial step is domain-based cell folding. Its time complexity includes table featurization and the clustering step. The complexity of table featurization on all tables is $\mathcal{O}(|S| \cdot m \cdot n \cdot e)$ because to generate a single BERT embedding, we iterate overall rows and columns of each table once and perform the embedding operation e. After generating the embedding, we run HDBSCAN, which in the worst case has a time complexity of $\mathcal{O}(|S|^2)$ [57], as we have one embedding per table. Thus, the overall time complexity for domain-based cell folding is $\mathcal{O}(|S| \cdot m \cdot n \cdot e) + \mathcal{O}(|S|^2)$.

For quality-based cell folding, we consider the complexity of feature generation, i.e., running base detectors, and the cell clustering algorithm. Among the detectors we cover, extracting functional dependency violation statistics $nv_{(L/R)HS}$ is the most expensive ones as they require pairwise comparisons between columns. The complexity for applying this detector on each table t is $\mathcal{O}(m^2 \cdot n)$ [4]. As we have to extract features for all tables, the complexity becomes $\mathcal{O}(|S| \cdot m^2 \cdot n)$. With v detectors (features), we end up with a worst case complexity of $\mathcal{O}(v \cdot |S| \cdot m^2 \cdot n)$. The complexity of the Mini-batch K-Means itself is linear in the number of elements as it implements LLoyd's algorithm. That is why we can summarize the clustering for each domain fold as the clustering over all cells as $\mathcal{O}(i \cdot k \cdot |S| \cdot m \cdot n \cdot v)$ [57] where i is the number of iterations, k is the number of clusters.

For sampling, we extract centroids from each cell fold, which requires to average the vector of all cells of each cell fold. The complexity for this step is linear in the number of its elements. That is why we can again sum over all cells in S and obtain $\mathcal{O}(|S| \cdot m \cdot n \cdot v)$. To propagate the labels, we again iterate over all cells: $\mathcal{O}(|S| \cdot m \cdot n)$. The training and prediction complexity is $\mathcal{O}(|S| \cdot m \cdot n \cdot v \cdot (\mathcal{O}_{train} + \mathcal{O}_{pred}))$ where $\mathcal{O}_{train}$ and $\mathcal{O}_{pred}$ encode the cost for training and predicting a cell, respectively. All in all, the overall complexity is:

$$\mathcal{O}(|S| \cdot m \cdot n \cdot e) + \mathcal{O}(|S|^2) + \mathcal{O}(v \cdot |S| \cdot m^2 \cdot n) + \mathcal{O}(i \cdot k \cdot |S| \cdot m \cdot n \cdot v) + \mathcal{O}(|S| \cdot m \cdot n \cdot v) + \mathcal{O}(|S| \cdot m \cdot n) + \mathcal{O}(|S| \cdot m \cdot n \cdot v \cdot (\mathcal{O}_{train} + \mathcal{O}_{pred}))$$

In Summary our approach is quadratic in the number of tables and columns but linear in the number of table rows and feature vector size. The quadratic complexity with regard to the number of tables can be avoided if we drop domain-based folding.

Table 1: Dataset characteristics. The error types are missing value (MV), typo (T), formatting issue (FI), violated attribute dependency (VAD) and numeric outliers (NO).

Name	Number of Tables	Size (#Cells)	Error Rate (cells)	Error Types
Quintet	5	199772	9%	MV, T, FI, VAD
REIN	8	3469027	13%	MV, T, VAD, NO
DGov-NTR	143	2458289	16%	NO, FI & T, VAD
DGov-NT	159	6527740	15%	NO, FI & T
DGov-NO	96	874570	2%	NO
DGov-Typo	96	874570	9%	FI & T
DGov-RV	96	874570	8%	VAD
DGov-1K	1173	47045294	unknown	unknown
WDC	100	64286	unknown	unknown
GitTables	1000	1002488	unknown	unknown

4 EXPERIMENTS

We compare the effectiveness, efficiency, and scalability of different versions of Matelda with adaptations of state-of-the-art error detection systems. Additionally, we assess the impact of each folding step on the overall performance, conduct an ablation study on datasets with different error types to examine the robustness and limitations of Matelda in identifying diverse error types, and analyze the impact of the chosen features on the effectiveness of our approach.

4.1 Experimental Setup

4.1.1 Datasets. To evaluate our method, ideally, we require a benchmark of real-world dirty table sets with ground truth to asses all three aspects of effectiveness, efficiency, and scalability. However, to the best of our knowledge such benchmarks do not exist. As the ground truth is lacking for the open data lakes, such as Web Tables [15] and GitTables [26], we create several table set benchmarks. Table 1 contains the characteristics of these benchmarks [34, 50]. All of these datasets are publicly available [2].

Common Datasets:

Quintet Datasets: Five open-source datasets named "Flights", "Beers", "Hospital", "Movies", and "Rayyan" have been previously extensively used for evaluating data cleaning systems [34, 51]. The availability of ground truth and the diversity of the five datasets and their application in the past make them an interesting case for studying the performance of our system in a multi-dataset scenario.

WDC Web Table Corpus: We randomly picked 100 English-language relational webtables out of the massive collection of over 233 million tables derived from the July 2015 version of the CommonCrawl [30] to measure the effectiveness of our approach. We pre-filtered tables that had fewer than 21 rows or more than 100,000 rows, as well as those with fewer than three columns or more than ten columns.

GitTables: This dataset contains one million tables extracted from CSV files in GitHub repositories [26]. As the size and shape of datasets are arguably closer to datasets from an organization, we randomly selected a subset containing 1000 tables.

Datasets with synthetic errors:

DGov-X Datasets: To further understand the challenges and opportunities in cleaning larger table sets, we also created datasets with artificial errors. For this purpose, we composed five datasets using tables from data.gov, which are better curated than the aforementioned open lakes. Our benchmark creation process consists of table retrieval, functional dependency discovery, and error generation modules. In table retrieval, we retrieved random tables from data.gov. All module scripts are available in our repository [2]. As error generation scales superlinearly with the number of rows of a table, we dropped all tables from data.gov that are larger than 4MB. We also skipped tables that could not be parsed or managed by the error generation module BART [53]. Using BART, we inject random syntactic errors (typos), numeric outliers, and functional dependency violations. The latter covers semantic errors. For generating functional dependency violations, we used the HyFD algorithm [45, 47] to mine functional dependencies. BART was configured to generate errors on both sides of a functional dependency randomly.

We evenly distributed the number of errors among the three types and utilized as many functional dependencies as possible for each dataset. Details on the error rates and the size of the datasets we generated (DGov-X) are shown in Table 1.

REIN: Abdelaal et al. published this dataset consisting of eight tables, "Adult", "Breast Cancer", "Smart Factory", "Nasa", "Bikes", "Soil Moisture", "Mercedes", and "HAR" [2]. The authors introduced errors using BART [2, 53].

4.1.2 *Hardware specification and deployment.*

For our experiments, we utilized two standalone machines running Debian, equipped with 512 GB of memory and AMD processors featuring 64 cores. Specifically, for the runtime experiment, we conducted the tests on a machine running Debian 12. The code for Matelda, the baselines, and the Error-Generator are in Python and available online [2].

4.1.3 *Choice of parameters.*

If not specified otherwise, we used default parameters of underlying clustering and classification techniques. We only adjusted the minimum cluster size parameter for HDBSCAN, setting it to two for the domain folding phase. Moreover, we set the batch size for the Mini-batch K-Means algorithm to 256 × cores, thereby facilitating parallelism [48].

4.1.4 *Baselines.*

We compared our approach to two state-of-the-art supervised cleaning approaches Raha [34], and HoloDetect [23]. We further included four unsupervised approaches. Uni-Detect [59] is a pre-trained unsupervised cleaning technique. ASPELL [12] focuses on identifying and fixing spelling errors [40]. We also included two data quality testing frameworks, Deequ [54], and GX [22].

Raha. Raha is a semi-supervised error detection system [34]. It utilizes a small number of user-provided labels to drive the detection process. As Raha requires at least two labels per column to work while we also consider cases where we have fewer labels than tables, we developed several variations of Raha, some of which also handle the limited label budget situation:

(1) **Raha-Standard:** This is Raha with its standard label distribution scheme, i.e., there is at least one labeled tuple per dataset.
(2) **Raha-RandomTables (RT):** For this variation, labels are distributed randomly across tables considering entire tuples at a time. We shuffle the tables and assign one labeled tuple to each table in sequence until the labeling budget is exhausted. We skip tables that contain more columns than there are labels.
(3) **Raha-2LabelsPerCol (2LPC):** In this version, we randomly select one column at a time and label two cells from this column until the labeling budget is exhausted. This way it is ensured that each considered column receives the minimum amount of two labels. Some columns remain unlabeled.
(4) **Raha-20LabelsPerCol (20LPC):** This variation works en par with Raha-2LPC with the difference that every considered column receives at least 20 labels.

HoloDetect. HoloDetect is a semi-supervised approach that uses data augmentation. Since its original implementation is not publicly available, we used a recent third-party implementation [29].

Uni-Detect. Uni-Detect [59] is an unsupervised approach to detect errors in large corpora. Uni-Detect focuses on achieving high precision. It learns patterns for individual columns and verifies values that violate such patterns. The code was not available. Thus, we pursued a best-effort implementation after consulting with the authors. Our implementation of Uni-Detect is available [2]. The performance of Uni-Detect depends on its pre-training. As the original pre-training corpus is not available, we followed the suggestion of the authors and used the WDC Web Table Corpus [30]. We used five million random tables to pretrain Uni-Detect.

Aspell. We utilized ASPELL [12] as a dictionary-based spell checker to verify the correctness of all cells within the datasets. We employed the default English dictionary for our datasets; however, any compatible dictionaries can be used.

Deequ. This framework is designed to efficiently test datasets, against data quality constraints [54]. Deequ operates on one dataset at a time and requires predefined quality constraints. Lastly, Deequ does not pinpoint erroneous cells for each unit test. Instead, it provides a general quality profile for the dataset. We applied Deequ individually to each table and used its constraint suggestion module to obtain constraints, which we then used as input for Deequ.

Great Expectations (GX). This data quality platform is similar to Deequ [22]. It includes a data assistant module for extracting these constraints. Similar as with Deequ, we utilized this module to extract constraints for each table and used them as inputs to GX.

4.2 Effectiveness Comparisons

We assessed the effectiveness of Matelda in comparison to all baselines. The experiment is conducted on the four different sets, Quintet, REIN, DGov-NTR (DGov with Numeric outliers, Typos, and Rule violations), and DGov-NT (DGov with Numeric outliers and Typos) where we have the ground truth. The lakes differ with regard to the number of tables, error rates, and error types.

The results of the comparison are depicted in Figure 3. The results for each experiment are the average of five independent runs.

In all charts of Figure 3, the x-axis scales the labeling budget based on the number of labeled tuples per table. Thus, any $x < 1$ implies that, on average, only a fraction of a single tuple per table was labeled. The numbers in brackets on the second line indicate the total number of labeled cells for some x-values.

Matelda consistently outperforms all competitors on all datasets until a labeling budget of ten labeled tuples per dataset. Compared to the second best approach Raha and for two labeled tuples per dataset, Matelda's F1-Score is on average 9% better

[2]https://github.com/LUH-DBS/Matelda

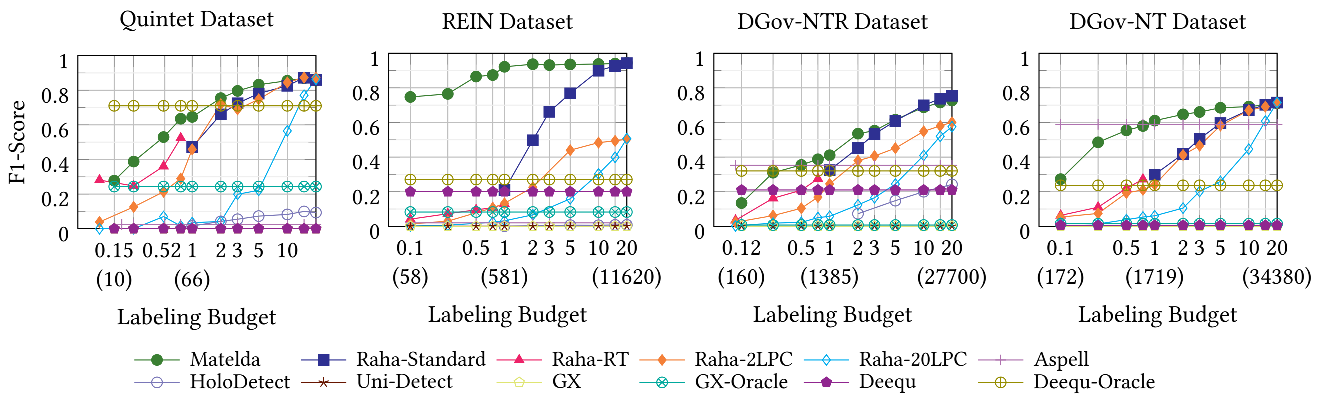


Figure 3: Effectiveness of Matelda vs. Baselines. The X-Axis depicts the average number of tuples per table that can be labeled. When the budget is below one, we have fewer labeled tuples than tables, and Raha-Standard and HoloDetect are not applicable.

on Quintet, 45% better on REIN, 9% better on DGov-NTR, and 24% better on DGov-NT. When the labeling budget exceeds ten tuples per table, Raha catches up to Matelda. When more than ten labeled tuples are available Raha catches up to Matelda and slightly outperforms it on the DGov-NTR dataset.

As discussed in Section 4.1.4, we included several variations of Raha into the mix that work with smaller labeling budgets to address the scenarios with fewer labels than datasets. Raha-2LPC and Raha-20LPC achieve generally high precision since each selected column receives enough labels. However, as many columns remain untreated, the overall recall suffers significantly. For example, Raha-20LPC achieves 57% precision on the Quintet dataset but only 2% recall when the labeling budget covers two labeled tuples per dataset. On the REIN dataset, for the same labeling budget, Raha-20LPC achieved 93% precision and only 3% recall. A similar argument can be made for Raha-RT. While a few tables receive labels, the others remain untouched, negatively affecting recall. The F1-Score of the alternatives Raha-RT, Raha-2LPC, and Raha-20LPC, varies depending on the randomly selected tables and columns to be cleaned. On the Quintet dataset, all variants show the highest average standard deviation: Raha-RT shows an average F1-Score deviation of 11%, Raha-2LPC records a deviation of 9%, and Raha-20LPC demonstrates a deviation of 7%.

Figure 3 shows a straight line for Uni-Detect and ASPELL in each graph, because they do not require labels. Uni-Detect generally performs worse than the supervised approaches. Its recall is very low as it captures only values that are globally inconsistent. Its precision is 15% on the Quintet, 91% on REIN, 23% on DGov-NTR, and 14% on the DGov-NT, respectively. Higher precision might be possible with a better training set, yet as mentioned before we followed the author's recommendations for obtaining the pre-trained dataset. Note that for the DGov datasets, we terminated each module execution after 24 hours. The pairwise comparisons employed in Uni-Detect is a bottleneck on large tables.

ASPELL is the fastest baseline and is relatively effective on synthesized datasets. Similar to Uni-Detect, ASPELL has higher precision than recall. On the REIN dataset, Aspell achieves 99% precision, and 1% recall. On the DGov-NTR dataset, ASPELL achieves a precision of 48% and a recall of 28%. Its performance is higher on the DGov-NT dataset due to a higher incidence of generated typos, reaching a precision of 67% and a recall of 50%. However, ASPELL's effectiveness drops significantly on the Quintet dataset (2% precision and 3% recall). Overall, ASPELL displays a reasonable alternative in setups where only typos are to be expected. We also experimented with employing ASPELL as a weak supervisor for label preparation, but this strategy was unsuccessful due to ASPELL's limited precision (below 70%).

Due to runtime and resource constraints, we could not run HoloDetect on all datasets. On the Quintent dataset only one iteration for all labeling budgets took one week. Thus, we did not run more iterations. On the REIN dataset, HoloDetect finished only on two tables, "breast cancer" and "Nasa". The execution on the other tables failed due to either memory usage issues or excessively long runtimes (exceeding 3 hours per table). For DGov-NTR, we limited the set of experiments to labeling budgets 2, 5, 10, and 20, which took nine days. As the performance was clearly lower than the competitors, we did not run it for smaller labeling budgets. Given the limited resources and the clear trend on the other two datasets, we did not run it on the larger DGov-NT dataset.

To run GX and Deequ, we leverage the systems' built-in functionalities to automatically extract constraints. GX provides a data assistant tool that can automatically extract four types of constraints: (1) expecting the table row count to fall within a specified range, (2) expecting the unique value count in a column to fall within a specified range, (3) expecting column values not to be null, and (4) expecting column values to be null. Deequ suggests constraints based on the data type of the column. For string columns, it suggests constraints related to the minimum and maximum value length, while for numerical columns, it considers the magnitude of values, including statistics such as average and standard deviation. Both systems apply common rules across data types, such as checking for completeness and detecting data type violations. As illustrated in Figure 3, GX has a near-zero F1-Score when its constraints are generated based on the dirty dataset only. In the same set-up, Deequ performs better, detecting data type violations and achieving F1-scores of up to 21%. To further investigate the capabilities of these systems, we also let them use the ground truth for constraint extraction, although it is unrealistic. We refer to this modified approaches as GX-Oracle and Deequ-Oracle. The GX-Oracle still yields a very low F1-score, as it only addresses issues related to missing values. Deequ-Oracle achieved a 70% F1-score on the Quintet dataset. This improvement is due to the system's ability to detect

Table 2: Effectiveness experiments on WDC

System	#TP	#FP	#FN	P	R	F1
Matelda	175	69	23	72%	**88%**	**79%**
Raha-Standard	106	49	92	68%	53%	60%
HoloDetect	85	32	113	**73%**	43%	54%
ASPELL	14	113	184	11%	7%	9%

Table 3: Recall in capturing different error types in Quintet

System	MV	REP	SEM	TYP	Total	Total
	Recall				Precision	Recall
Matelda	**95%**	**84%**	**44%**	**14%**	**75%**	**75%**
Raha-Standard	59%	74%	25%	11%	75%	61%
HoloDetect	59%	14%	23%	0%	63%	2%

representational errors and missing values when correct constraints are extracted from the clean version of the data. On all other datasets also Deequ-Oracle falls behind. Note that in a real scenario the clean dataset is not available.

4.3 Effectiveness Comparisons on a Real Lake

To further assess the effectiveness of the different approaches, we analyzed their performance on the WDC dataset. Due to the absence of ground truth in this data lake, we were limited to measuring only a sample of the results. We conducted a manual evaluation of 400 cells, comparing Matelda's effectiveness against that of Raha, HoloDetect, and ASPELL. In fact, we picked 100 random cells out of errors detected by each of these systems. For this experiment, we only tried one labeling budget variation —two labeled tuples per table. Note that, for smaller labeling budgets Raha and HoloDetect will not be applicable and for larger labeling budgets, we would have to manually label ten times more cells across 100 different tables, which is a daunting manual task.

The outcomes are outlined in Table 2. Matelda yields the highest F1-Score on the sample. While its precision is marginally below the highest recorded precision rate of 73% by HoloDetect, it achieves a significantly higher recall value of 88%. Raha follows at second place with an F1-Score of 60%.

4.4 Ablation Study on Error Types

We performed two sets of experiments to assess Matelda's effectiveness in identifying different error types. The first experiment measures Matelda's performance on datasets where only a specific type of error has been injected. The results are illustrated in Figure 4. We compare Matelda against the most effective baselines so far, Raha and its variants, along with ASPELL. On the DGov-NO dataset that only contains outliers, Matelda consistently surpasses all baselines across various labeling budgets. On the DGov-Typo dataset that only contains typos, Matelda achieves a superior F1-Score as soon as 323 cells (0.3% labeled tuples per table) are labeled. Raha catches up when the number of labeled tuples per table exceeds 15. On the DGov-RV dataset, Matelda's effectiveness is similar to Raha from one labeled tuple on. This similarity in performance shows that although our approach compares cells across tables, the designed rule-based features effectively capture the similarity of cells that participate in rule-violations. Note that Raha should have had an advantage because it clusters cells within individual columns that are subject to the same set of rules.

The second experiment measures the completeness of the system's results in detecting different error categories within the common datasets. We selected the Quintet dataset as our focus for this evaluation. We manually assign each error in the Quintet dataset to an error type. The error types include MV (Missing Values), REP (Representation and Formatting Errors), SEM (Semantic Errors), and TYP (Typos). The datasets, annotated with these types, are made publicly accessible in our repository [2]. Now we assess the recall for each error type individually. The results are drawn from the setup with two labeled tuples per table. Table 3 shows that Matelda significantly surpasses both Raha and HoloDetect on each individual error type. Notably, Matelda exhibits higher recall for semantic errors and missing values.

4.5 Matelda Variant and Component Analysis

First, we analyze variations of the folding step. Then we analyze the impact of different cell feature groups. Finally, we compare different strategies on how to train the error detection classifiers. Here, we do not include DGov-NT because it is similar to DGov-NTR but yet much larger requiring more computation.

4.5.1 *Folding Strategies Impact Analysis.*

There are several different folding scenarios to consider. One approach is to entirely avoid the folding of cells, treating each column individually. If the folding is too strict, the label sharing is also strict. This approach is basically covered by our Raha alternatives and faces the challenge of lacking labels for every distinct column. Alternatively, one can opt for excessive folding, i.e., avoiding the filtering implications of the domain-based folding strategy. As a result, we fold cells despite domain dissimilarities and let the quality-based cell folding distinguish cell folds across all domains. Another variant is to refine the domain-based cell folding, using columns features. This approach separates cells from columns vulnerable to different error types. To have a better understanding of the impact of such folding approaches, we compare the standard Matelda approach to the following two alternatives:

1) Matelda-EDF (Extreme Domain Folding): All cells belong to the same domain. Only quality-based cell folding is carried out.

2) Matelda +SF (+Syntactic Folding): In this configuration, we do a refinement of the domain-based folding based on column similarities. To decide which cells from which columns should go into the same fold, we need to capture syntactic signals that imply specific data quality problems. Similar to the work on dirtiness similarity [32], we consider features that capture data types, character distributions, and cell value lengths.

Figure 5 depicts the results of these experiments on Quintet and DGov-NTR. On the Quintet dataset, the performance differences between Matelda-Standard and its two other versions are minor. Yet, on the DGov-NTR dataset, the standard and EDF variants consistently outperform the +SF-variant, suggesting that a folding based on syntactic features could interfere with the effective sharing of labels across cells. This also shows that the features for quality-based cell folding are generally effective in considering column similarities although they describe cells from different tables.

While the performance of Matelda-EDF suggest that one could drop the domain-based folding step and put everything into one fold, it is important to note that without it, the runtime is up to 8 times higher compared to the standard approach on the DGov-NTR dataset. In Section 4.6, the runtime is discussed in more detail.

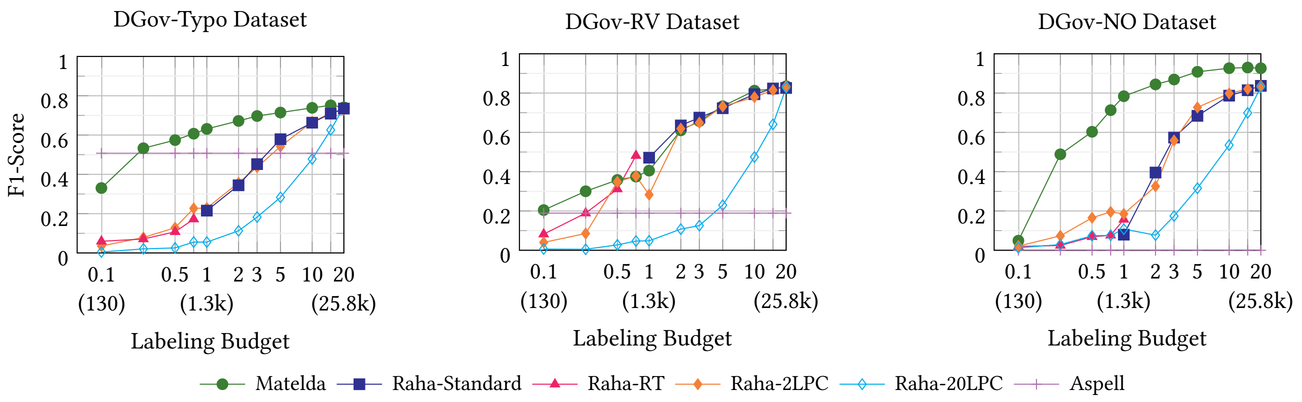


Figure 4: Effectiveness of Matelda vs. Baseline Approaches - Ablation Study on Error Types.

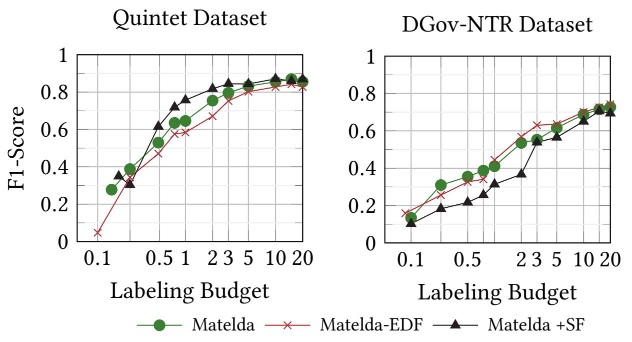


Figure 5: Folding Strategies Impact

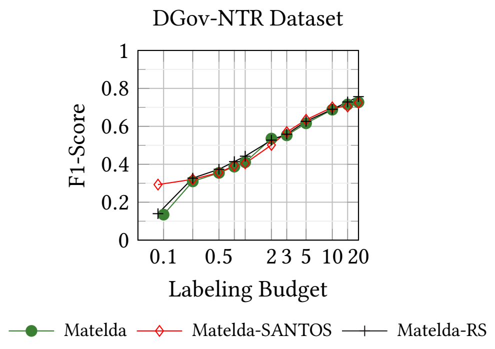


Figure 6: Domain Folding Design Impact

4.5.2 Domain-based Cell Folding Analysis. Matelda employs a BERT embedding model to represent tables during the domain-folding step. To explore alternative solutions and address potential scalability concerns, we introduce two additional variants of Matelda: Matelda-Santos, and Matelda-RS which are discussed in this section.

1) Matelda-Santos: We compared our choice to a variation that relies on unionability scores of the recent state-of-the-art system SANTOS [28]. We utilized the unionability scores as measures of similarity and performed clustering based on these scores. The workflow of the domain folding step using SANTOS is as follows. First SANTOS generates semantic representations for each table by utilizing knowledge bases. In the union discovery setting, it then computes the unionability score of a given table with those in the dataset and retrieves the unionable tables with the corresponding unionability scores. In our case, we employ this process on each individual table of our table set. For each table, we must specify an intent column to serve as the root for relationship searches in SANTOS. We default to the first column as the intent column, as it often corresponds to the subject column. In the second step, we treat the unionability scores as measures of similarity between tables and apply HDBSCAN clustering.

2) Matelda-RS: Given that using BERT embeddings may pose scalability challenges for large datasets, we developed Matelda-RS as a variant to mitigate this issue. Matelda-RS employs random sampling to reduce the complexity of the domain-folding process. Specifically, only 1% of the rows in each table are randomly selected as input for the BERT embedding and domain clustering modules.

On the Quintet dataset, SANTOS results in the same folds as our approach which is why we do not include its graph. Figure 6 shows that the effectiveness of the SANTOS-based method is similar to the standard method. However, SANTOS is significantly slower due to the computationally intensive tasks it that it performs. These include generating a synthesized knowledge base and extracting functional dependencies. On average, the runtime for Matelda-Santos stands at 4,963 seconds, in contrast to 1,130 seconds for Matelda-Standard on the DGov-NTR dataset. Matelda-RS achieves nearly the same F1-Score as Matelda, but its runtime is significantly reduced. On average, Matelda-RS achieves a runtime of 998 seconds, in contrast to 1,130 seconds for the standard Matelda when labeling two tuples per table on the DGov-NTR dataset. This demonstrates that Matelda can avoid scalability issues by relying on sampling when dealing with large datasets.

4.5.3 Quality-based Cell Folding Features Impact Analysis. To evaluate the impact of the features used for quality-based cell folding, we perform an ablation study with the following variants:

1) Matelda-NOD (No Outlier Detector): This variant includes all features with the exception of outlier detection signals.

2) Matelda-NTD (No Typo Detector): Typo detectors are excluded in this variant.

3) Matelda-NRVD (No Rule Violation Detector): Here, we leverage all features except for rule violation features.

Figure 7 shows that Matelda with all cell features outperforms the other methods in terms of F1-Score for the majority of labeling budgets on both datasets. Matelda-NOD consistently demonstrates lower performance compared to the other variants, emphasizing the essential contribution of outlier detection in assessing cell quality similarity. On the DGov-NTR dataset, the benefit

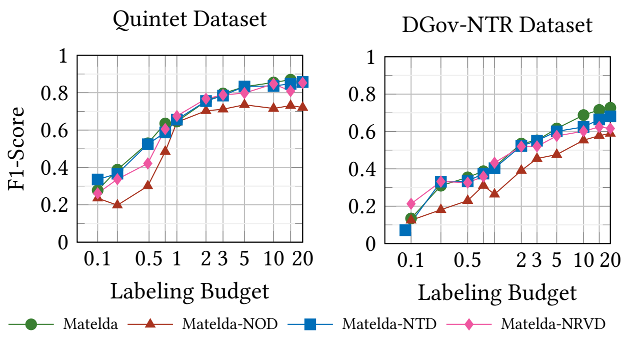


Figure 7: Quality-based Cell Folding: Feature Analysis

of the rule violation and typo detectors becomes more evident when the labeling budget increases beyond three labeled tuples per table.

4.5.4 Training Phase Design Analysis. Matelda, by default, uses a separate classifier for each column. We designed it this way because when each classifier is trained on a limited set of cells, i.e., only the cells within a single column, it will be more efficient. Moreover, as each classifier is dedicated to just one column, there is no risk of further confusing information from different columns or tables. To further validate this choice, we compare it with other possible options. We designed two other variants of Matelda:
1) Matelda-TPDF (Training Per Domain Fold): Instead of one classifier per column, we have one classifier for each domain fold.
2) Matelda-TUCF (Training using Unlabeled Cell Folds): Similar to Matelda-TPDF, this variant utilizes one classifier for each domain fold. However, we create twice as many quality-based cell folds for each domain fold, which in turn limits the label propagation to smaller but more coherent clusters.

The results of this experiment are depicted in Figure 8. The first row of charts showcases the effectiveness of each approach on Quintet and DGov-NTR. The second row demonstrates the runtime for the two datasets. Matelda and Matelda-TPDF deliver the best F1-Scores among the considered variants. This outcome aligns with our expectations, given that these variants have trained on more cell values compared to Matelda-TUCF. Yet, the standard approach is more runtime-efficient. Matelda-TUCF displays the fastest runtime, but allows some cell folds to remain unlabeled compromising its effectiveness. The experiment shows that separating the training for each column individually is best.

4.6 Efficiency and Scalability

To evaluate the efficiency and scalability of Matelda in detecting errors in large data lakes, we designed an experiment focusing on the number of tables. We used two datasets for this purpose: DGov-1K and GitTables. From GitTables, we sampled multiple sets ranging from 100 to 1000 tables, and from DGov-1K, the sets ranged from 250 to 1173 tables.

We compared the execution time of Matelda, Matelda-EDF, and Raha and report the average of three independent runs. Execution time includes all processes from data reading to error detection. User interactions were excluded from the calculation for all systems. We used two labeled tuples for each table as the labeling budget, which is the minimum requirement for Raha. Figure 9 shows that Matelda clearly scales better than Matelda-EDF on GitTables. This is because clustering a large number of

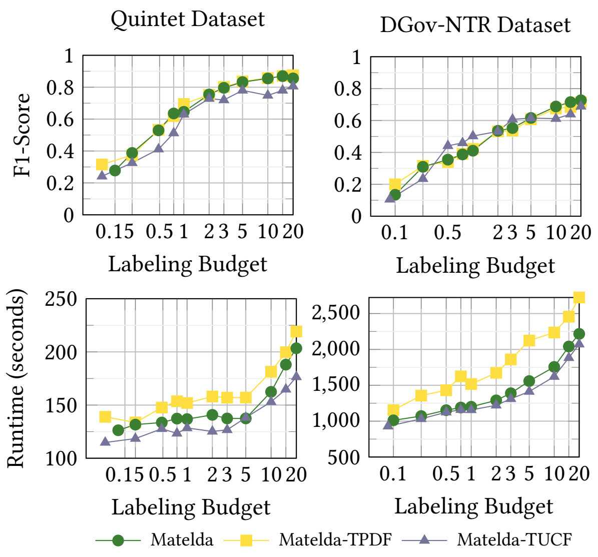


Figure 8: Comparing different training strategies

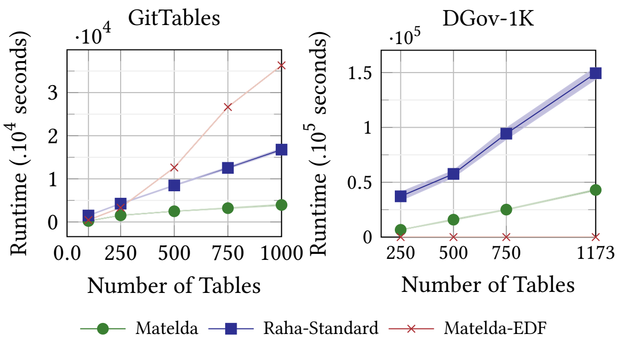


Figure 9: Scalability results- The average number of rows per table is 126 for GitTables-1000 and 3.1k for DGov-1K.

cells is memory-intensive and leads to paging, and the domain folding approach helps reduce this cost. Due to memory limitations, Matelda-EDF did not finish on DGov subsets showing the benefit of creating independent domain-based folds. Matelda is also faster than Raha. This is mainly due to the reduced feature space as schema-dependent features are absent.

4.7 Limitations

Although Matelda shows great promise in our experiments, we highlight some of its algorithmic limitations and the assumptions that must hold for it to perform well.
Assumptions on the table set: Matelda assumes a degree of semantic or syntactic similarity across the tables within the set. If such similarity is lacking, the system's ability to share and reuse labels across tables becomes ineffective.
Error types: The need for a unified feature space reduces the potential of applying schema-related detectors. Our heuristic based on similarity-flooding is a weakened form of capturing inter-column relationships. While the set of current detectors covers existing categories of errors, we cannot claim to have covered all possible signals to capture the quality of a cell.
System design: The current implementation of Matelda relies on the interplay of multiple components, such as featurization, clustering, and classification. For the sake of generalizability, we

use each of them with their default parameters, which might not be optimal.

Reliance on the clustering steps: While our experiments demonstrate that Matelda is not affected by the accuracy of the domain-based folding step, the subsequent quality-based folding is crucial for the effectiveness of label sharing. In particular, if the tables become very large and encounter many concept drifts across the rows, the small labeling budget might not be enough to effectively separate cells for label sharing.

5 RELATED WORK

Our work directly builds on existing data cleaning and lake management literature.

Data Cleaning. There is a large body of research on data quality management and data cleaning [16, 52, 54]. Traditional error detection and correction techniques such as Nadeef [14], Llunatic [20] and Horizon [52] require predefined rules and configurations. Katara [11] is another example that requires additional master data like knowledge bases. There are also commercial data quality debugging tools that identify errors based on provided constraints. These tools primarily focus on automating the application of rules and constraints to datasets [16]. Examples of such tools include Deequ [54] and Great Expectations (GX) [22]. Our approach addresses scenarios where dataset constraints are not known upfront.

Semi-supervised approaches, such as Raha [34], ED2 [43], HoloClean [51], HoloDetect [23], and Baran [33], consider data cleaning as a classification problem. Raha and Baran ensemble detectors and correctors and leverage a few labeled tuples to train the corresponding classifiers [33, 34]. ED2 applies active learning to get labels for uncertain tuples [41, 43]. HoloDetect leverages data augmentation techniques to tackle the data imbalance problem in error detection. HoloClean integrates integrity constraints, external data, and statistical profiles into a factor graph model, which it then uses to predict the correct values for detected errors.

All prior work focus on individual tables and do not provide techniques and considerations for cleaning multiple tables at once. The moment we consider multiple tables simultaneously, user involvement and runtime add up per dataset. With Matelda, we bridge this gap by identifying characteristics of data cells that are comparable across multiple datasets.

Unsupervised approaches, such as Auto-Detect [25] and Uni-Detect [59] leverage statistical consistency for identifying outliers and other inconsistencies in individual tables. Auto-Detect is primarily designed to detect pattern violations within single columns, which makes it less practical at identifying issues like semantic errors or numeric outliers. Uni-Detect requires pre-training on a clean lake. As we show in our experiments it fails to identify semantic errors and errors that are not obvious outliers. With Matelda, we cover a more comprehensive set of error types, as we generally build upon the holistic nature of the error detection strategies that are included in the state-of-the-art system Raha.

Another line of research focuses on crowdsourced based web table cleaning [61] and optimizes the cleaning task order for crowd workers. This approach is orthogonal to our semi-supervised approach.

Recently, LLMs have been considered for automated cleaning [60]. The LLM-based system Sudowoodo still requires annotations for each table to fine-tune the models and work effectively. According to its code-base, further data-specific rules are also necessary to fine-tune the model.

Data Lake Management. Matelda aims to detect errors in data lakes by reorganizing the lake's structure. Numerous works have focused on addressing challenges at the scale of data lakes [38]. This field covers a wide range of topics, including the discovery of unionable tables [28, 39], and joinable tables [17, 62], data lake organization for easier navigation [44], and enhancing data quality using constraints extracted from the corpus [18]. Our work builds on this line of research as discussed in Section 4.5.2.

6 CONCLUSION

In this paper, we tackled the problem of semi-supervised error detection in a multi-table scenario. We introduced Matelda, which pre-organizes a given set of tables so that user labels can be shared across tables and uses table-agnostic features to make cells comparable across tables. While our system has demonstrated superior effectiveness on our test set benchmarks, there are still opportunities for further improvement. Enhancements such as including additional signals for more effective detection of functional dependency violations and minimizing user labeling efforts could further optimize the method for application on large data lakes. As we focused on the error detection task in this paper, the exploration of strategies for data repair within data lakes represents a promising and largely unexplored direction for future research.

ACKNOWLEDGMENTS

We would like to extend our gratitude to Arne Meier, Paolo Papotti, Yeye He, and Sebastian Schelter, for their valuable feedback. We also thank Mohamed Abdelaal for providing us with the REIN dataset. This project has been supported by the German Research Foundation (DFG) under grant agreement 387872445.

REFERENCES

[1] Accessed: 2024. OpenRefine. https://openrefine.org/

[2] Mohamed Abdelaal, Christian Hammacher, and Harald Schöning. 2023. REIN: A Comprehensive Benchmark Framework for Data Cleaning Methods in ML Pipelines. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*.

[3] 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? *Proceedings of the VLDB Endowment (PVLDB)* 9, 12 (2016), 993–1004.

[4] Ziawasch Abedjan, Lukasz Golab, Felix Naumann, and Thorsten Papenbrock. 2018. *Data Profiling.* Morgan & Claypool Publishers.

[5] Felix Bießmann, Tammo Rukat, Philipp Schmidt, Prathik Naidu, Sebastian Schelter, Andrey Taptunov, Dustin Lange, and David Salinas. 2019. DataWig: Missing Value Imputation for Tables. *J. Mach. Learn. Res.* 20 (2019), 175:1–175:6.

[6] Michael J. Cafarella, Alon Y. Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. 2008. WebTables: exploring the power of tables on the web. *Proceedings of the VLDB Endowment (PVLDB)* 1, 1 (2008), 538–549.

[7] Ricardo J. G. B. Campello, Davoud Moulavi, Arthur Zimek, and Jörg Sander. 2015. Hierarchical Density Estimates for Data Clustering, Visualization, and Outlier Detection. *ACM Transactions on Knowledge Discovery from Data* 10, 1 (2015), 5:1–5:51.

[8] Olivier Chapelle, Jason Weston, and Bernhard Schölkopf. 2002. Cluster Kernels for Semi-Supervised Learning. In *Neural Information Processing Systems (NIPS)*.

[9] Xu Chu, Ihab F. Ilyas, Sanjay Krishnan, and Jiannan Wang. 2016. Data Cleaning: Overview and Emerging Challenges. In *Proceedings of the International Conference on Management of Data (SIGMOD)*.

[10] Xu Chu, Ihab F. Ilyas, and Paolo Papotti. 2013. Holistic data cleaning: Putting violations into context. In *Proceedings of the International Conference on Data Engineering (ICDE)*.

[11] Xu Chu, John Morcos, Ihab F. Ilyas, Mourad Ouzzani, Paolo Papotti, Nan Tang, and Yin Ye. 2015. KATARA: A Data Cleaning System Powered by Knowledge Bases and Crowdsourcing. In *Proceedings of the International Conference on Management of Data (SIGMOD)*.

[12] GNU Aspell developers. Accessed: 2024. GNU Aspell. http://aspell.net. Accessed: 31.03.2024.

[13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*.

[14] Amr Ebaid, Ahmed K. Elmagarmid, Ihab F. Ilyas, Mourad Ouzzani, Jorge-Arnulfo Quiané-Ruiz, Nan Tang, and Si Yin. 2013. NADEEF: A Generalized Data Cleaning System. *Proceedings of the VLDB Endowment (PVLDB)* 6, 12 (2013), 1218–1221.

[15] Julian Eberius, Maik Thiele, Katrin Braunschweig, and Wolfgang Lehner. 2015. Top-k Entity Augmentation Using Consistent Set Covering. In *Proceedings of the International Conference on Scientific and Statistical Database Management (SSDBM)*.

[16] Lisa Ehrlinger and Wolfram Wöß. 2022. A Survey of Data Quality Measurement and Monitoring Tools. *Frontiers Big Data* 5 (2022), 850611.

[17] Mahdi Esmailoghli, Jorge-Arnulfo Quiané-Ruiz, and Ziawasch Abedjan. 2022. MATE: Multi-Attribute Table Extraction. *Proceedings of the VLDB Endowment (PVLDB)* 15, 8 (2022), 1684–1696.

[18] Mina H. Farid, Alexandra Roatis, Ihab F. Ilyas, Hella-Franziska Hoffmann, and Xu Chu. 2016. CLAMS: Bringing Quality to Data Lakes. In *Proceedings of the International Conference on Management of Data (SIGMOD)*.

[19] Jerome H Friedman. 2001. Greedy function approximation: a gradient boosting machine. *Annals of statistics* (2001), 1189–1232.

[20] Floris Geerts, Giansalvatore Mecca, Paolo Papotti, and Donatello Santoro. 2020. Cleaning data with Llunatic. *VLDB Journal* 29, 4 (2020), 867–892.

[21] Boris Glavic, Giansalvatore Mecca, Renée J. Miller, Paolo Papotti, Donatello Santoro, and Enzo Veltri. 2024. Similarity Measures For Incomplete Database Instances. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*.

[22] Abe Gong, James Campbell, and Great Expectations. Accessed: 2024. *Great Expectations.* https://doi.org/10.5281/zenodo.5683574

[23] Alireza Heidari, Joshua McGrath, Ihab F. Ilyas, and Theodoros Rekatsinas. 2019. HoloDetect: Few-Shot Learning for Error Detection. In *Proceedings of the International Conference on Management of Data (SIGMOD)*.

[24] Victoria J. Hodge and Jim Austin. 2004. A Survey of Outlier Detection Methodologies. *Artif. Intell. Rev.* 22, 2 (2004), 85–126.

[25] Zhipeng Huang and Yeye He. 2018. Auto-Detect: Data-Driven Error Detection in Tables. In *Proceedings of the International Conference on Management of Data (SIGMOD)*.

[26] Madelon Hulsebos, Çagatay Demiralp, and Paul Groth. 2023. Gittables: A large-scale corpus of relational tables. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–17.

[27] Sean Kandel, Andreas Paepcke, Joseph M. Hellerstein, and Jeffrey Heer. 2011. Wrangler: interactive visual specification of data transformation scripts. In *Proceedings of the International Conference on Human Factors in Computing Systems (CHI)*.

[28] Aamod Khatiwada, Grace Fan, Roee Shraga, Zixuan Chen, Wolfgang Gatterbauer, Renée J. Miller, and Mirek Riedewald. 2023. SANTOS: Relationship-based Semantic Table Union Search. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 9:1–9:25.

[29] Abolfazl Mohajeri Khorasani, Sahar Ghassabi, Behshid Behkamal, and Mostafa Milani. 2023. Explainable Error Detection Method for Structured Data using HoloDetect framework. In *International Conference on Computer and Knowledge Engineering*.

[30] Oliver Lehmberg, Dominique Ritze, Robert Meusel, and Christian Bizer. 2016. A Large Public Corpus of Web Tables containing Time and Context Metadata. In *Proceedings of the International World Wide Web Conference (WWW)*.

[31] Peng Li, Xi Rao, Jennifer Blase, Yue Zhang, Xu Chu, and Ce Zhang. 2021. CleanML: A Study for Evaluating the Impact of Data Cleaning on ML Classification Tasks. In *Proceedings of the International Conference on Data Engineering (ICDE)*.

[32] Mohammad Mahdavi and Ziawasch Abedjan. 2019. REDS: Estimating the Performance of Error Detection Strategies Based on Dirtiness Profiles. In *Proceedings of the International Conference on Scientific and Statistical Database Management (SSDBM)*.

[33] Mohammad Mahdavi and Ziawasch Abedjan. 2020. Baran: Effective Error Correction via a Unified Context Representation and Transfer Learning. *Proceedings of the VLDB Endowment (PVLDB)* 13, 11 (2020), 1948–1961.

[34] 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 International Conference on Management of Data (SIGMOD)*.

[35] Zelda Mariet, Rachael Harding, Sam Madden, et al. 2016. Outlier detection in heterogeneous datasets using automatic tuple expansion. (2016).

[36] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. 2002. Similarity Flooding: A Versatile Graph Matching Algorithm and Its Application to Schema Matching. In *Proceedings of the International Conference on Data Engineering (ICDE)*.

[37] Fatemeh Nargesian, Abolfazl Asudeh, and H. V. Jagadish. 2022. Responsible Data Integration: Next-generation Challenges. In *Proceedings of the International Conference on Management of Data (SIGMOD)*.

[38] Fatemeh Nargesian, Erkang Zhu, Renée J. Miller, Ken Q. Pu, and Patricia C. Arocena. 2019. Data Lake Management: Challenges and Opportunities. *Proceedings of the VLDB Endowment (PVLDB)* 12, 12 (2019), 1986–1989.

[39] Fatemeh Nargesian, Erkang Zhu, Ken Q. Pu, and Renée J. Miller. 2018. Table Union Search on Open Data. *Proceedings of the VLDB Endowment (PVLDB)* 11, 7 (2018), 813–825.

[40] Markus Näther. 2020. An In-Depth Comparison of 14 Spelling Correction Tools on a Common Benchmark. In *Proceedings of The Language Resources and Evaluation Conference (LREC)*.

[41] Felix Neutatz, Binger Chen, Ziawasch Abedjan, and Eugene Wu. 2021. From Cleaning before ML to Cleaning for ML. *IEEE Data Engineering Bulletin* 44, 1 (2021), 24–41.

[42] Felix Neutatz, Binger Chen, Yazan Alkhatib, Jingwen Ye, and Ziawasch Abedjan. 2022. Data Cleaning and AutoML: Would an Optimizer Choose to Clean? *Datenbank Spektrum* 22, 2 (2022), 121–130.

[43] Felix Neutatz, Mohammad Mahdavi, and Ziawasch Abedjan. 2019. ED2: A Case for Active Learning in Error Detection. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*.

[44] Paul Ouellette, Aidan Sciortino, Fatemeh Nargesian, Bahar Ghadiri Bashardoost, Erkang Zhu, Ken Pu, and Renée J. Miller. 2021. RONIN: Data Lake Exploration. *Proceedings of the VLDB Endowment (PVLDB)* 14, 12 (2021), 2863–2866.

[45] Thorsten Papenbrock, Tanja Bergmann, Moritz Finke, Jakob Zwiener, and Felix Naumann. 2015. Data Profiling with Metanome. *Proceedings of the VLDB Endowment (PVLDB)* 8, 12 (2015), 1860–1863.

[46] Thorsten Papenbrock, Jens Ehrlich, Jannik Marten, Tommy Neubert, Jan-Peer Rudolph, Martin Schönberg, Jakob Zwiener, and Felix Naumann. 2015. Functional Dependency Discovery: An Experimental Evaluation of Seven Algorithms. *Proceedings of the VLDB Endowment (PVLDB)* 8, 10 (2015), 1082–1093.

[47] Thorsten Papenbrock and Felix Naumann. 2016. A Hybrid Approach to Functional Dependency Discovery. In *Proceedings of the International Conference on Management of Data (SIGMOD)*.

[48] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research (JMLR)* 12 (2011), 2825–2830.

[49] Clement Pit-Claudel, Zelda Mariet, Rachael Harding, and Sam Madden. 2016. Outlier detection in heterogeneous datasets using automatic tuple expansion. (2016).

[50] Erhard Rahm and Hong Hai Do. 2000. Data Cleaning: Problems and Current Approaches. *IEEE Data Engineering Bulletin* 23, 4 (2000), 3–13.

[51] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. 2017. HoloClean: Holistic Data Repairs with Probabilistic Inference. *Proceedings of the VLDB Endowment (PVLDB)* 10, 11 (2017), 1190–1201.

[52] El Kindi Rezig, Mourad Ouzzani, Walid G. Aref, Ahmed K. Elmagarmid, Ahmed R. Mahmood, and Michael Stonebraker. 2021. Horizon: Scalable Dependency-driven Data Cleaning. *Proceedings of the VLDB Endowment (PVLDB)* 14, 11 (2021), 2546–2554.

[53] Donatello Santoro, Patricia C. Arocena, Boris Glavic, Giansalvatore Mecca, Renée J. Miller, and Paolo Papotti. 2016. BART in Action: Error Generation and Empirical Evaluations of Data-Cleaning Systems. In *Proceedings of the International Conference on Management of Data (SIGMOD)*.

[54] Sebastian Schelter, Dustin Lange, Philipp Schmidt, Meltem Celikel, Felix Bießmann, and Andreas Grafberger. 2018. Automating Large-Scale Data Quality Verification. *Proceedings of the VLDB Endowment (PVLDB)* 11, 12 (2018), 1781–1794.

[55] D. Sculley. 2010. Web-scale k-means clustering. In *Proceedings of the International World Wide Web Conference (WWW)*.

[56] Haoxiang Shi and Tetsuya Sakai. 2023. Self-Supervised and Few-Shot Contrastive Learning Frameworks for Text Clustering. *IEEE Access* 11 (2023), 84134–84143.

[57] Pang-Ning Tan, Michael S. Steinbach, Anuj Karpatne, and Vipin Kumar. 2019. *Introduction to Data Mining (Second Edition)*. Pearson.

[58] Larysa Visengeriyeva and Ziawasch Abedjan. 2018. Metadata-driven error detection. In *Proceedings of the International Conference on Scientific and Statistical Database Management (SSDBM)*.

[59] Pei Wang and Yeye He. 2019. Uni-Detect: A Unified Approach to Automated Error Detection in Tables. In *Proceedings of the International Conference on Management of Data (SIGMOD)*.

[60] Runhui Wang, Yuliang Li, and Jin Wang. 2023. Sudowoodo: Contrastive Self-supervised Learning for Multi-purpose Data Integration and Preparation. In *Proceedings of the International Conference on Data Engineering (ICDE)*.

[61] Yihai Xi, Ning Wang, Yiyi Zhang, and Xinyu Chen. 2024. CrowdDA: Difficulty-aware crowdsourcing task optimization for cleaning web tables. *Expert Syst. Appl.* 238, Part E (2024), 122139.

[62] Erkang Zhu, Dong Deng, Fatemeh Nargesian, and Renée J. Miller. 2019. JOSIE: Overlap Set Similarity Search for Finding Joinable Tables in Data Lakes. In *Proceedings of the International Conference on Management of Data (SIGMOD)*.