Repairing Raw Data Files with TASHEEH

Mazhar Hameed*
Gisma University of Applied Sciences
Potsdam, Germany
mazhar.hameed@gisma.com

Gerardo Vitagliano*
MIT CSAIL
Cambridge, MA
gerarvit@mit.edu

Fabian Panse*
University of Augsburg
Augsburg, Germany
fabian.panse@uni-a.de

Felix Naumann
Hasso Plattner Institute
University of Potsdam, Germany
felix.naumann@hpi.de

ABSTRACT

Data files serve as a vital resource for all data-driven applications. Among these, comma-separated value (CSV) files are particularly popular with users and businesses due to their flexible standard. However, also due to this loose standard, the data in these files are often indeed "raw", fraught with many types of structural inconsistencies that hinder seamless ingestion into a data system. We say that rows in CSV files with such structural inconsistencies are *ill-formed*.

Traditionally, data practitioners write custom code to repair the structure and format of ill-formed rows, even before they can leverage data cleaning tools and libraries, which typically assume that data are already properly loaded. Writing such code and configuring loading scripts is tedious, time-consuming, and requires both expertise and frequent human intervention.

To address these challenges, we present TASHEEH – a system that automatically detects ill-formed rows containing data and then standardizes their structure into a uniform format based on the structure of well-formed rows. By automating these essential steps, our system frees up valuable time and resources, enabling practitioners to focus on downstream stages of the data processing pipeline.

1. REPAIRING ILL-FORMED DATA ROWS

Preparing or wrangling data, including raw data in files, is a decades-old problem. For raw data, comma-separated value (CSV) files, due to their flexible standard, are particularly popular among business users, data storage companies, and researchers to collect and share data [3, 4, 29, 31, 32, 39, 41]. Our recent data loading benchmark, POLLOCK [47], surveyed 17 governmental data portals across six continents and found that CSV files are the second most popular file format, making up over 31% of their datasets, emphasizing their critical role in data processing pipelines. However, their popularity, driven by their flexible format, demands much effort for data practitioners during pre-processing: due to their loose format, these files appear in various dialects [9, 12, 46] deviating from the RFC standard [20]. In addition, they often contain various structural inconsistencies [15, 24].

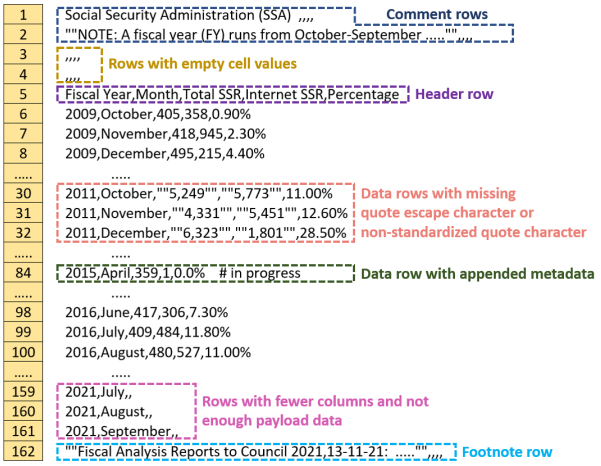

Figure 1: A sample of a raw CSV file with *ill-formed* rows due to structural inconsistencies.

According to the POLLOCK survey, an analysis of 3 712 real-world CSV files from the Mendeley data-sharing platform[1] and the UK Government data portal[2] revealed widespread deviations from the RFC standard. Specifically, 1 040 files exhibited inconsistent row lengths, caused by schema drift, preamble lines, or mismatched column counts. Regarding delimiters, 943 files did not use the standard comma delimiter, including 834 that used semicolons, 101 that relied on combinations of commas with whitespace or tabs, and 8 that exclusively used tabs or whitespace, introducing inconsistencies across rows. Furthermore, 476 files contained multiple header lines, of which 94 had multirow table headers spanning two or three lines, and 282 included preamble rows separating comments or metadata from the table. Newline sequences also varied: instead of following the standard combination of carriage return and line feed (CRLF), 1 691 used only line feed (LF), and 7 relied solely on carriage return (CR). These results underscore the prevalence and diversity of structural challenges encountered in real-world CSV data.

*This research was conducted while the authors were at the Hasso Plattner Institute, University of Potsdam.

[1] https://www.mendeley.com/
[2] https://www.data.gov.uk/

Consequently, it is challenging to load these files correctly into data-driven systems without prior data preparation steps [14, 47]. Developers and scientists spend much of their development time cleaning and organizing data in these files, leaving less time for analytical tasks [19, 34, 45].

Recently, our community has begun recognizing such shortcomings as research opportunities and has developed solutions for preparing and cleaning data [5–7, 15, 24, 28, 35, 38, 40, 44]. However, we are still far from creating a fully automated data processing pipeline, in part due to the open challenges of handling raw CSV files.

Among the many challenges, detecting and cleaning "ill-formed" rows (see Section 3.3 for a formal definition) in CSV files are difficult problems [15]. Such ill-formed rows occur in raw data due to loosely defined schemata, incorrect formatting of values, discrepancies in row structures, etc. They can lead to aborted loading processes, incorrectly parsed data, and can interfere with the training process of machine learning algorithms. Figure 1 shows an example of a raw CSV file taken from a government data portal. It highlights groups of ill-formed rows with different inconsistencies. Our goal is to automatically detect those rows that contain data, which we call *wanted rows* (see Section 3.1 for a formal definition), and automatically repair their structural inconsistencies.

These detection and repair tasks are challenging, because some rows are ill-formed and contain no data, e.g., table titles, footnotes, or empty rows. We call these rows *ill-formed unwanted*. Other rows may contain data yet be ill-formed, e.g., because they contain additional structural or formatting information and possibly additional columns. We refer to these rows as *ill-formed wanted*. To *detect* ill-formed rows, we make use of our pattern-based system, Suragh [15], which abstracts row structures into structural patterns based on a syntactic pattern grammar. Here, we extend the use of our syntactic pattern grammar with our new system Tasheeh[3]. The goal of Tasheeh is to improve the classification of ill-formed rows by recognizing wanted and unwanted ill-formed rows, but in particular to automatically *clean* wanted rows.

In the following, we discuss structural challenges using the example file shown in Figure 1.

Example 1. *The real-world file in Figure 1 contains, among other inconsistencies, cell values with either a non-standard quote character or a missing quote escape character, e.g.,* `""5,249""` *(row 30). The RFC standard for CSV files [20] states: 1) Each field must be enclosed in double-quotes if its value contains a character used as a field delimiter; and 2) a double-quote appearing inside a field must be escaped by preceding it with another double quote. With those rules, the standardized versions of the value should either appear as* `"5,249"` *as per the Rule 1 if the cell value is a number* `5,249`*, or as* `"""5,249"""`*, as per Rule 2 if the cell value is a string value* `"5,249"` *(with quotations included). Loading the file as-is in a downstream application might lead to a shift of values across columns.*

Example 2. *Another example of structural inconsistency in the file of Figure 1 is data and metadata appearing in the same row (row 84). We also observed this inconsistency appear in the opposite order as data next-to metadata. In both combinations, metadata appear mainly in the form of comments, where users leave notes for reference or try to explain data in that row. Another cause is manual data entry or automatic data integration from multiple sources, where users or automated scripts miss the* newline separator*, resulting in a different number of columns across rows.*

Tasheeh aims to help streamline a data processing pipeline by automating preparation tasks at the structural level and minimizing the burden of manual data preparation. Moreover, a comprehensive demo integrating our error detection system, Suragh, and the correction system, Tasheeh, was presented at CIKM 2023 [16]. The demo features a minimalist, user-friendly graphical interface designed to detect and correct errors in CSV files with just a few clicks.

Our work makes the following main contributions:

1. A formalization to describe ill- and well-formedness of rows, wanted and unwanted rows, and row structure standardization.
2. A set of files from four open data sources, with each row annotated for ill-formedness or well-formedness, and for wantedness or unwantedness, for a total of 200 351 rows. The files, together with the classification annotations, manually cleaned wanted rows, and code, are publicly available[4].
3. A system, Tasheeh, that automatically recognizes ill-formed wanted rows and cleans their structure using a novel pattern transformation algebra.
4. A wide range of experiments conducted to validate Tasheeh for both classification and transformation of ill-formed rows.

The rest of the paper is organized as follows: Section 2 presents related research efforts and tools addressing data preparation challenges. Section 3 defines the relevant concepts related to pattern extraction and ill- and well-formedness of rows, and provides formal definitions. Section 4 illustrates the workflow of Tasheeh and presents the processes of classifying and transforming ill-formed rows. Section 5 presents the experimental evaluation of Tasheeh. Section 6 provides a summary of this study and discusses data preparation opportunities beyond Tasheeh.

2. STATE-OF-THE-ART IN DATA PREPARATION

While structural inconsistencies in CSV files have been underexplored, there have been some notable attempts in related work to comprehend the structure of tabular data. We provide a succinct overview of these approaches and briefly describe the pertinent research directions that can be complemented by our research.

Table extraction.

Extracting relational tables from diverse sources has led to the development of several tools [4, 5, 10, 11, 25, 27]. Notable examples include Tegra [5] (for web lists), TableSense [10] (for spreadsheets), and Pytheas [4] (for CSV files). Tegra optimizes row splits to ensure column alignment, TableSense uses deep learning to segment tables in

[3] Tasheeh (tas-heeh) is an Urdu word that means correction or rectification.

[4] `https://github.com/HMazharHameed/TASHEEH`

spreadsheets, and PYTHEAS employs machine-learned rules to detect tables in CSV files by identifying the position of data rows. While these systems do not explicitly focus on cleaning structural issues, which is the primary focus of our research, we experimentally compare TASHEEH with PYTHEAS, as it is designed for CSV files and effectively identifies data rows, enabling a direct comparison with our classification component (Section 5.2). In contrast, TEGRA assumes that delimiters are consistent across all rows, while TABLESENSE relies on the structured format of spreadsheets with clearly defined cell boundaries. These assumptions limit their applicability to the less structured and more variable nature of CSV files, rendering a direct comparison unfair.

Row and cell type detection.

In plain text files like CSV, not every row may contain data [29]. Therefore, accurately identifying the cell and row boundaries and comprehending their underlying semantics are essential for efficient data processing. Researchers have devised various tools employing both supervised and unsupervised approaches to classify cells and rows within tabular data [2, 14, 24, 26, 33]. Among these approaches, Jiang et al. proposed the state-of-the-art STRUDEL approach [24]: STRUDEL is a multi-class random forest classifier that leverages three types of features: content, context, and computational features to classify rows in CSV files. Although its primary focus is not on structure-cleaning, we include it in our comparative analysis by evaluating its performance against TASHEEH's classification component (Section 5.2).

File structure preparation.

In the context of non-standardized CSV files, various structural inconsistencies can arise when processing data using different tools or parsers [47]. Among these, one notable issue is the occurrence of shifted column values. Sun et al. introduced the SRFN system [42] to address this specific problem. The approach focuses on repairing shifted values by leveraging the likelihood of neighboring attribute values and determining the correct position for swapping values among columns. It is the sole solution that attempts to address one structural problem in CSV files. In our evaluation (Section 5.3), we compare the performance of the TASHEEH transformation component with the SRFN system.

Commercial Tools for Data Preparation.

The widespread adoption of commercial data preparation tools underscores the market's recognition of their vital role in modern data pipelines [14]. Tools like OpenRefine [22], Tableau [43], and Trifacta [23] cater to diverse use cases, streamlining error detection, data transformation, and integration tasks. However, as highlighted in our survey of commercial tools [14], these solutions often rely on pre-processed and well-structured input data, struggling to handle raw, non-standardized CSV files. This gap emphasizes the need for more adaptive solutions to address challenges like structural inconsistencies in raw data, providing a strong impetus for systems like TASHEEH.

3. SYNTACTIC ROW PATTERNS

In this section, we first define wanted and unwanted rows and present our problem definition. Following that, we provide a brief overview of our previous work, SURAGH, which we use as an initial step in TASHEEH.

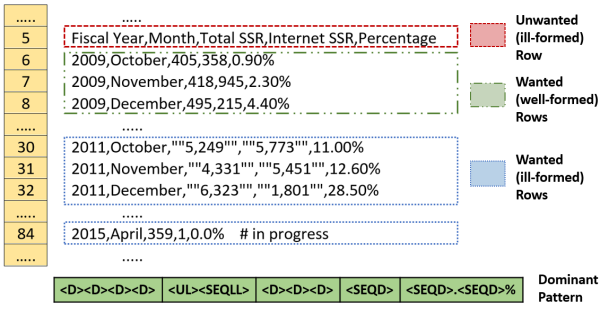

Figure 2: Selected rows of the CSV file of Figure 1 with well-formed rows (green), ill-formed wanted rows (blue), and an ill-formed unwanted row (red). The dominant pattern at the bottom corresponds to the file structure as automatically detected by SURAGH.

3.1 Problem Definition

The input to our approach is a file that is composed of a number of rows. A row is a sequence of characters terminated by a newline separator. Further, let T be a relational table serialized in a CSV file F and let R be the set of rows of F. Every tuple $t \in T$ contains data from one or possibly multiple rows (e.g., due to a misplaced line separator). Moreover, due to missing or misplaced line separators, a single row may contain data for two tuples. Formalizing these concepts, we define wanted and unwanted rows as follows:

DEFINITION 1. *Let T be a relational table serialized in a CSV file F, and let $\Phi\colon T \to 2^R$ be a function that maps every tuple $t \in T$ to a non-empty set of rows in F from which it can be parsed. A row $r \in R$ is called* wanted, *if it serializes data from any tuple of T, i.e., if $\exists t \in T\colon r \in \Phi(t)$, and* unwanted *otherwise.*

Since at parsing time we do not know the relational table serialized in a file F (nor do we know Φ), classifying rows as wanted or unwanted is often not trivial and leads to a trade-off between the two data quality dimensions completeness and soundness. If we mistakenly label a 'wanted' row as 'unwanted', it leads to information loss, causing the loaded table to miss some data and thus becoming incomplete. Vice versa, if an 'unwanted' row is erroneously classified as 'wanted', it introduces incorrect information into the table.We now define the problem we address as follows:

> *Given as input a raw data file F with a set of rows, identify the structure of the table T serialized in F and transform all wanted rows to follow that structure, while retaining all of their data values.*

To solve this problem, we need to perform three steps: (1) *structure detection* to identify the table T, (2) *row classification* to separate wanted and unwanted rows, and (3) *row transformation* to standardize the structure of wanted rows into a uniform format.

We have addressed the first step of the problem in our previous work, SURAGH, using a pattern-based approach [15]. SURAGH takes a CSV file as input and effectively classifies its rows as ill-formed or well-formed based on the dominant

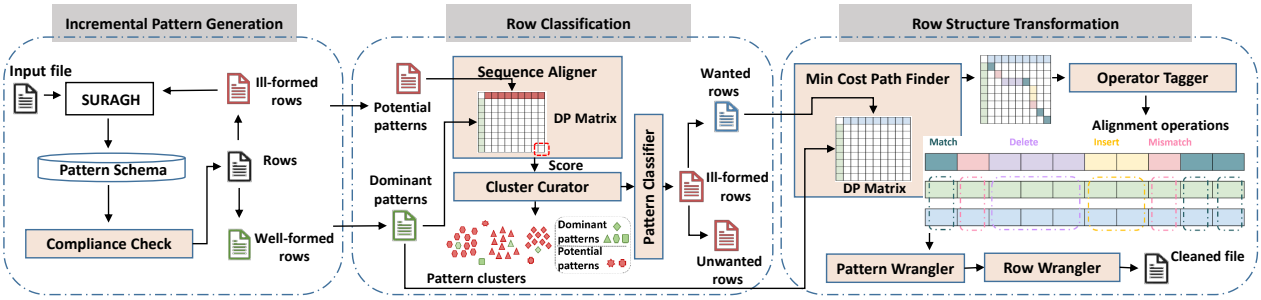

Figure 3: The workflow of TASHEEH

row pattern(s) (see Figure 2). For the next steps, we developed TASHEEH that utilizes the pattern language introduced in SURAGH and further enhances the process by classifying ill-formed rows into wanted and unwanted rows. Figure 2 shows the example results of TASHEEH's classification process on the results of SURAGH.

In the following sections, we briefly explain how SURAGH extracts dominant row patterns from CSV rows, a basic step for row classification and transformation in TASHEEH. Note that we assume the transformations should only clean the structure of the rows and should neither lose data nor invent new information that was not present in the input file.

3.2 Pattern Modeling

The goal of SURAGH is to understand the structure of rows in an input file, abstracting it with patterns. To generate patterns, SURAGH defines a grammar to map cell values into abstract representations. We refer to the production rules of this grammar as *abstractions*, which are of two types: (1) encoder and (2) aggregator. The encoder abstractions convert single characters into a more general representation, e.g., the character "A" is represented with $\langle UL \rangle$, for "Upper Letter". The aggregator abstractions combine representations resulting from other encoder and aggregator abstractions based on a given rule, e.g., the character sequence "ABC" is first encoded as $\langle UL \rangle\langle UL \rangle\langle UL \rangle$, and then can be combined into the single abstraction $\langle SEQUL \rangle$, for "Sequence of Upper Letters". Using the given pattern grammar, including 21 abstractions, SURAGH generates *syntactic cell patterns* for each cell value [15]. Abstractions are weighted based on their specificity to later prune overly general patterns.

3.3 Pattern Extraction

After generating patterns for each cell value, SURAGH aggregates them for each column, creating *syntactic column patterns*. Among all possible cell patterns within a column, it retains only those with a sufficiently high specificity and with enough coverage of actual cells in the column. For the example input file in Figure 1, the three selected column patterns for the column "`Percentage`" are $\langle SEQD \rangle.\langle SEQD \rangle\%$, $\langle D \rangle.\langle D \rangle\langle D \rangle\%$, and $\langle D \rangle\langle D \rangle.\langle D \rangle\langle D \rangle\%$.

A *syntactic row pattern* is obtained by combining one column pattern for each of the input file columns. To identify good row patterns that represent one or more rows, SURAGH inspects all combinations of columns patterns, again selecting those with high specificity and coverage. For the input file in Figure 1, two of the syntactic row patterns are shown in the following table where cell separators indicate the "Delimiter" $\langle DEL \rangle$ abstraction, which we omit to save space:

#	Syntactic Row Patterns				
$\mathcal{P}_1$	$\langle D \rangle\langle D \rangle\langle D \rangle\langle D \rangle$	$\langle UL \rangle\langle SEQLL \rangle$	$\langle D \rangle\langle D \rangle\langle D \rangle$	$\langle D \rangle\langle D \rangle\langle D \rangle$	$\langle D \rangle.\langle D \rangle\langle D \rangle$ %
$\mathcal{P}_2$	$\langle D \rangle\langle D \rangle\langle D \rangle\langle D \rangle$	$\langle UL \rangle\langle SEQLL \rangle$	$\langle D \rangle\langle D \rangle\langle D \rangle$	$\langle SEQD \rangle$	$\langle SEQD \rangle.\langle SEQD \rangle$ %

Row patterns that are not a proper subset of another pattern are called *dominant* patterns. To avoid pattern redundancy, SURAGH detects and removes all non-dominant patterns. For example, for the input file in Figure 1, the row pattern $\mathcal{P}_2$ is a dominant row pattern (see Figure 2), as it is not a subset of any other row pattern.

Finally, the constructed set of dominant row pattern(s) is used to classify individual rows as ill-formed or well-formed: A row *conforms to* a dominant row pattern if it has the same number of columns as the dominant pattern, and all column values of the row conform to the corresponding column patterns of the dominant pattern. We call such a row *well-formed*, and *ill-formed* otherwise [15]. TASHEEH uses dominant patterns as a filter in the classification phase and as the target of the transformation phase: all wanted rows should conform to a dominant pattern (see Section 4).

4. THE TASHEEH SYSTEM

TASHEEH operates in three phases (see Figure 3). In the first phase, it uses SURAGH to classify rows as ill-formed or well-formed using *dominant* row patterns ($\mathcal{P}_d$). For ill-formed rows, it incrementally generates patterns, referred to as *potential* row patterns ($\mathcal{P}_p$), until all such rows have corresponding patterns (see Section 4.1).

The second phase classifies ill-formed rows as wanted or unwanted using incrementally generated patterns and a pattern level distance measure derived from sequence alignment methods [13]. Section 4.2 explains this step in detail.

In its third and final phase, TASHEEH collects wanted rows, well-formed rows, and their patterns from the previous phases. It then uses a pattern transformation algebra to transform the wanted rows into well-formed ones – Section 4.3 explains the details.

4.1 Incremental Pattern Generation

SURAGH generates dominant row patterns to classify rows as ill-formed or well-formed, retaining only dominant patterns and originally discarding others. However, TASHEEH requires these additional patterns because (1) their abstraction facilitates effective comparison with dominant patterns, and (2) transforming general patterns covering multiple rows is more efficient than transforming individual rows.

To generate such further patterns, TASHEEH incrementally

Table 1: Example set of dominant and potential row patterns (aligned by delimiters); $\mathcal{P}_d$ corresponds to well-formed rows (Figure 2 → rows 6-8), $\mathcal{P}_{p1}$ corresponds to an unwanted row (Figure 2 → row 5), $\mathcal{P}_{p2}$ corresponds to wanted rows (Figure 2 → rows 30-32), and $\mathcal{P}_{p3}$ corresponds to a wanted row (Figure 2 → row 84).

Dominant Row Pattern:

$\mathcal{P}_d$	⟨D⟩⟨D⟩⟨D⟩⟨D⟩	⟨DEL⟩ ⟨UL⟩⟨SEQLL⟩	⟨DEL⟩ ⟨D⟩⟨D⟩⟨D⟩	⟨DEL⟩ ⟨SEQD⟩	⟨DEL⟩ ⟨SEQD⟩.⟨SEQD⟩%	

Potential Row Patterns:

$\mathcal{P}_{p1}$	`Fiscal Year`	⟨DEL⟩ `Month`	⟨DEL⟩ `Total SSR`	⟨DEL⟩ `Internet SSR`	⟨DEL⟩ `Percentage`	
$\mathcal{P}_{p2}$	`2011`	⟨DEL⟩ ⟨UL⟩⟨SEQLL⟩	⟨DEL⟩ `""`⟨D⟩	⟨DEL⟩ ⟨D⟩⟨D⟩⟨D⟩ `""`	⟨DEL⟩ `""`⟨D⟩	⟨DEL⟩ ⟨D⟩⟨D⟩⟨D⟩ `""` ⟨DEL⟩ ⟨SEQD⟩.⟨SEQD⟩%
$\mathcal{P}_{p3}$	`2015`	⟨DEL⟩ ⟨UL⟩⟨SEQLL⟩	⟨DEL⟩ `359`	⟨DEL⟩ `1`	⟨DEL⟩ `0.0% # in progress`	

executes SURAGH, updating the classification criteria in each iteration. Each iteration excludes previously identified well-formed rows and re-assesses ill-formed ones, dynamically refining well-formedness definitions. For example, for the input file in Figure 1, three of the potential row patterns along with the dominant row pattern are shown in Table 1.

After obtaining dominant and potential patterns for well- and ill-formed rows, TASHEEH processes them further to classify ill-formed rows into *wanted* and *unwanted* – see next.

4.2 Row Classification

In this phase, TASHEEH calculates the *minimum pattern-level distance* between dominant and potential patterns. It then identifies the closest dominant pattern for each potential pattern and classifies ill-formed rows as wanted or unwanted, based on the distance score.

4.2.1 Pattern sequence aligner

The potential patterns generated by TASHEEH may or may not correspond to data rows. To measure their similarity to dominant patterns, we introduce a pattern-level distance measure derived from sequence alignment methods. Similar to edit distance frameworks for string matching [48], sequence alignment applies operations, such as "match", "mismatch", and "indel" (insertion or deletion), with user-defined operation costs. With this in mind, we introduce a distance-based alignment framework for pattern-to-pattern alignments, where the input sequences are entire row patterns [17]. The input patterns are compared column by column, splitting them on the delimiter character of a raw CSV file.

Consider the dominant pattern $\mathcal{P}_d$ and a potential pattern $\mathcal{P}_p$ from Table 1. Our alignment framework generates $\mathcal{P}'_d$ and $\mathcal{P}'_p$, ensuring the same number of column patterns by padding the shorter sequence with a gap character "-" based on the minimum cost edit path. Using a dynamic programming approach, similar to other sequence alignment methods [30], the framework finds the alignment with the lowest distance. To find an alignment between the two patterns $\mathcal{P}_d$ and $\mathcal{P}_p$, we instantiate a matrix $\mathcal{M}$ where the position at element i, j represents the minimum cost to transform $\mathcal{P}_p[0, \ldots, j]$ into $\mathcal{P}_d[0, \ldots, i]$. The matrix is initialized with $\mathcal{M}[i][0] = i$ and $\mathcal{M}[0][j] = j$, and then all other costs are filled using Equation (1):

$$\mathcal{M}(\mathcal{P}_d, \mathcal{P}_p)[i][j] = \min \begin{cases} \mathcal{M}[i-1][j] + 1, \\ \mathcal{M}[i][j-1] + 1, \\ \mathcal{M}[i-1][j-1] + \\ \quad \mathcal{D}(\mathcal{P}_d[i-1], \mathcal{P}_p[j-1]) \end{cases} \tag{1}$$

Here, the cost of the insertion and deletion is set to 1 (first and second lines of Equation (1)). To determine the cost between individual column patterns, we define a pattern distance function $\mathcal{D}$ by enumerating four possible cases, which are summarized in Equation (2):

$$\mathcal{D}(\alpha, \beta) = \begin{cases} 0, & \text{if } \alpha = \beta \\ 1, & \text{if } \alpha \in \{\text{'-'}, \langle EV \rangle\} \text{ or } \beta \in \{\text{'-'}, \langle EV \rangle\} \\ 1, & \text{if } \alpha = \langle DEL \rangle \text{ and } \beta \notin \{\text{'-'}, \langle EV \rangle \langle DEL \rangle\} \\ 1 - \dfrac{\sum\limits_{i=l,d,s} min(|\alpha_i|, |\beta_i|)}{max(|\alpha|, |\beta|)}, & \text{otherwise} \end{cases} \tag{2}$$

The pattern distance formula in Equation (2) is inspired by the string-by-string alignment approach [18], which we adapted to define the distance function $\mathcal{D}$ between column patterns (α, β) using abstractions [15]. We consider the typical three groups of abstraction classes: letters "*l*", digits "*d*", and symbols "*s*". Moreover, ⟨*EV*⟩ represents an empty value, while ⟨*DEL*⟩ denotes delimiter abstractions (see details in [15]). This choice of quantifying *distance* between string patterns is motivated by the need to capture structural similarities. For example, the values "`123 Main Street, New York, NY`" and "`789 Broadway Avenue, New York, NY`" exhibit a significant structural similarity despite high Levenshtein, Jaccard, and Hamming distances.

The aforementioned dynamic programming approach solves the following optimization:

$$\mathcal{D}(\mathcal{P}_d, \mathcal{P}_p) = \min_{\mathcal{P}'_d, \mathcal{P}'_p} \frac{1}{|\mathcal{P}'_p|} \sum_{k=1}^{|\mathcal{P}'_p|} \mathcal{D}(\mathcal{P}'_d[k], \mathcal{P}'_p[k]) \tag{3}$$

Here, $\mathcal{P}_d$ and $\mathcal{P}_p$ are the original input row patterns, while $\mathcal{P}'_d$ and $\mathcal{P}'_p$ are padded row patterns to obtain the same number of columns. Note that the special gap symbol can be padded at any position in the pattern sequence to minimize the pattern distance.

4.2.2 Pattern classification

Potential patterns that are close to a dominant pattern have a lower distance score. In cases with a single dominant pattern (e.g., Figure 2), potential patterns are aligned to it to calculate their distance. For files with multiple dominant patterns, distances are calculated for each combination of dominant and potential patterns. These distances are then passed to the pattern classifier, which uses a threshold θ to label potential patterns as wanted (distance $\leq \theta$) or unwanted (distance $> \theta$), indicating whether the corresponding rows contain data. With the experiments detailed in Section 5.2 we determined that $\theta = 0.3$ yielded the highest F-1 score.

4.3 Row Structure Transformation

In this phase, TASHEEH collects the ill-formed wanted rows, well-formed rows, their patterns, and the corresponding dynamic programming matrices $\mathcal{M}$ from the previous phase. First, it chooses the best alignment between dominant and

Table 2: Row pattern transformation operators

Operator	Description
Drop	Returns an empty column pattern.
Extract	Extracts a (wanted) part from a column pattern.
Ignore	Returns the unchanged input column pattern.
Move	Relocates a column pattern from one position to another.
Merge	Concatenates column patterns and appends the merged column pattern to the specified position.
Pad	Pads a row pattern with empty cell(s).
Permute	Rearranges a column pattern set with a given order.
Re-quote	Adds or removes quotes from a column pattern.
Re-escape	Adds or removes escapes from a column pattern.
Re-delimit	Adds or removes a field separator from a column pattern.
Re-line	Adds or removes a row separator from a column pattern.
Replace	Replaces abstractions in a column pattern.

Table 3: Pattern sequence alignment operators, where underlined transformation operators were used for both TASHEEH and BASELINE transformation strategies.

Alignment operator	Transformation direction	Corresponding transformation operator(s)
Match	*Diagonal*	Ignore
Mismatch	*Diagonal*	Drop, Extract, Ignore, Replace, Re-delimit, Re-quote, Re-escape, Re-line
Insert	*Vertical*	Pad, Permute
Delete	*Horizontal*	Merge, Drop, Extract, Move, Re-delimit, Re-quote, Re-escape, Re-line

wanted patterns, determining the necessary transformations to clean up the structure of wanted patterns. Then, the transformations identified at the pattern level are used to transform the wanted rows into well-formed ones.

Table 2 presents a set of operators to transform one pattern into another. This set is also the basis to later transform the corresponding rows from ill-formed ones to well-formed ones. The pattern transformation operators take one or more input column patterns and output up to one column pattern with a possibly transformed structure.

In the following sections, we explain how to obtain a complete pattern-level edit path and the functionalities of the transformation operators.

4.3.1 Minimum cost edit path

To align a dominant pattern with a wanted pattern, we trace back from the bottom-right of their matrix $\mathcal{M}$. A graph is constructed on $\mathcal{M}$, where each node represents a matrix cell containing a pair of column patterns, and edges indicate alignment operations ("match", "mismatch", "insert", and "delete") with weights based on transformation costs. Edge weights guide the selection of the best alignment, with the minimum cost path representing the optimal transformation. The alignment operations and their corresponding transformations are detailed in Table 3.

Next, we employ Dijkstra's shortest path algorithm [8] to compute the minimum cost edit path, defining the alignment between dominant and wanted patterns (see Table 4). The alignment process results in aligned row patterns ($\mathcal{P}_d$, $\mathcal{P}_{p2}$) and marked alignment operators, which the transformation engine later uses (see next).

4.3.2 Pattern wrangler

TASHEEH collects the aligned patterns and their minimum cost alignment from the previous step. It passes each aligned column pattern from the row patterns to the transformation engine, the *pattern wrangler*, which applies the required transformations. The engine stores the transformation results in a queue and continues processing the remaining column patterns (see Algorithm 1). After completing all transformations, the preferred results are applied to the corresponding data rows.

Algorithm 1: Pattern Wrangler

Input: Dominant pattern $\mathcal{P}_d$, Potential pattern $\mathcal{P}_p$, alignment A between $\mathcal{P}_d, \mathcal{P}_p$
Output: List of Transformations T

$T \leftarrow []$
foreach $0 \leq i \leq |\mathcal{P}_p|$ **do**
  $transformations \leftarrow$ ApplicableTransformations$(\mathcal{P}_d[i], \mathcal{P}_p[i], A[i])$
  $T_c \leftarrow$ GenerateCombinations$(transformations)$
  $T \leftarrow T \cup \arg\min_{c \in T_c} \Big(\textsc{Distance}\big(\mathcal{P}_d[i],\ c(\mathcal{P}_p[i])\big) \Big)$
end
return T

In the following, we briefly outline the alignment operators listed in Table 3 and their corresponding transformation operators from Table 2 in the context of the pattern wrangler.

Match. When aligning row patterns, identical column patterns that require no transformation are marked with the "match" alignment operator with the corresponding Ignore transformation operator. For example, in Table 4, matched column patterns indicate no transformation is needed.

Mismatch. For mismatched patterns, the engine selects operators based on the abstractions present. When dialect characters (e.g., delimiter, quotes, escapes) are absent, operators, such as Re-quote, Re-escape, and Re-delimit are excluded. Instead, it uses Drop, Replace, and Extract, prioritizing the latter two to reduce the pattern distance between the transformed and the dominant pattern, with Drop as the least preferred.

Insert. As previously stated, CSV files often have rows with varying column counts, causing misaligned patterns. The transformation engine detects missing columns using the "insert" alignment operator and resolves inconsistencies by applying Pad or Permute, typically padding extra columns at the start or end with field separators. A more complex scenario arises when columns must be added in the middle of a row, requiring precise placement. The transformation engine resolves this by iterating through possible alignment index combinations to determine the optimal position.

Delete. If a wanted row pattern has more columns than the dominant pattern, the alignment framework inserts gaps and marks the extra columns with the "delete" operator to address inconsistencies from shifted values or missing delimiters. The transformation engine starts with the Merge operator to combine patterns and update positions. It then ap-

Match	Match	Match	Match	Delete			Match	Delete			Match	Match
⟨D⟩⟨D⟩⟨D⟩⟨D⟩	⟨DEL⟩	⟨UL⟩⟨SEQLL⟩	⟨DEL⟩	⟨D⟩⟨D⟩⟨D⟩	_	_	⟨DEL⟩	⟨SEQD⟩	_	_	⟨DEL⟩	⟨SEQD⟩.⟨SEQD⟩%
2011	⟨DEL⟩	⟨UL⟩⟨SEQLL⟩	⟨DEL⟩	""⟨D⟩	⟨DEL⟩	⟨D⟩⟨D⟩⟨D⟩""	⟨DEL⟩	""⟨D⟩	⟨DEL⟩	⟨D⟩⟨D⟩⟨D⟩""	⟨DEL⟩	⟨SEQD⟩.⟨SEQD⟩%

Table 4: Minimum cost edit path alignment between row patterns $\mathcal{P}_d$ (Figure 2 → rows 6-8) and $\mathcal{P}_{p2}$ (Figure 2 → rows 30-32)

plies Re-quote, Re-escape, and Re-delimit operators to standardize quotes, escapes, and delimiters, ensuring compliance with RFC 4180.

4.3.3 *Row wrangler*

The row wrangler takes the sequence of transformations from the transformation queue and applies them to all data rows of the pattern at hand, thus cleaning the structure of the ill-formed but wanted rows. As a final result, TASHEEH usually outputs a clean and structured CSV file.

5. EXPERIMENTS

This section provides an overview of our experimental results, starting with the datasets and annotation details. We then present the performance of our pattern classifier at various distance score thresholds, followed by comparisons with a BASELINE approach and state-of-the-art row classifiers. Next, we analyze the effectiveness of TASHEEH's transformations. Finally, we provide a brief summary of a user study comparing manual wrangling to our system. Further experiments are detailed in [17].

5.1 Datasets and Annotation

We used datasets collected from four open data sources: DataGov, Mendeley, GitHub, and UKGov, leveraging files from our previous work SURAGH [15], and supplementing them with additional files. The statistics for these data sources are summarized in Table 5.

Table 5: Datasets: number of files (F), average number of rows (R), average well-formed (WF) rows per file, average ill-formed wanted (IFW) rows per file, and average ill-formed unwanted (IFU) rows per file.

Source	# F	Avg # R	Avg # WF	Avg # IFW	Avg # IFU
DataGov	62	877.7	819.9	51.5	6.2
Mendeley	34	2909.6	2841.4	51.0	17.2
GitHub	28	662.1	627.7	17.4	17.1
UKGov	24	1186.2	1153.3	30.4	2.5

Building on our previous work [15], we extended the annotated data following the same annotation strategy, labeling 200 351 rows as ill-formed or well-formed and further refining them into wanted and unwanted rows. Additionally, we created a ground truth of manually cleaned wanted rows for each file across all datasets for transformation experiments. The code artifacts together with datasets and annotations are publicly available.

5.2 Classification Performance Evaluation

This section evaluates TASHEEH's classification component, including finding the best distance threshold setting. In accordance with Definition 1, a detected ill-formed wanted row is a true positive if its ground truth label matches; otherwise, it is a false positive. We use the standard precision P and recall R metrics to assess the effectiveness of our system.

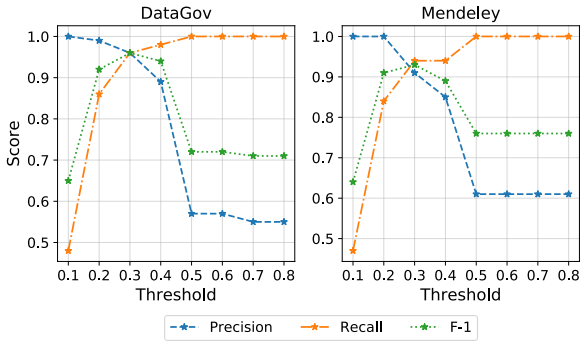

Figure 4: Precision, recall and F-1 measures at different distance score threshold values

Table 6: Row classification comparison overview (F1-scores)

Source	# rows	BASELINE	PANDAS	PYTHEAS	STRUDEL	TASHEEH
DataGov	54 416	0.54	0.66	0.84	0.89	**0.96**
Mendeley	98 927	0.51	0.61	0.79	0.86	**0.93**
GitHub	18 538	0.56	0.67	0.76	0.81	**0.95**
UKGov	28 469	0.62	0.74	0.87	0.93	**0.98**

We experimented with different threshold values to optimize the classification task. Figure 4 shows precision, recall, and F-1 scores for the DataGov and Mendeley datasets. Similar results were observed for UKGov and GitHub datasets (not shown due to space constraints), with a threshold of 0.3 yielding the highest F-1 score.

We compared TASHEEH's row classification component with four other approaches: a BASELINE classifier, inspired by TABULAR [1], which identifies rows as complete or erroneous based on whether their column count matches the header, and the related work approaches PANDAS [37], PYTHEAS [4], and STRUDEL [24]. Table 6 summarizes the row classification results, reporting F1-scores for each system. TASHEEH outperformed all other systems, achieving the highest F1-scores across all datasets. Further details about these experiments are discussed in [17].

5.3 Transformation Performance Evaluation

We evaluated the effectiveness of TASHEEH's transformations by measuring the accuracy of correctly cleaned ill-formed rows against the manually created transformation ground truth. A row is considered correctly cleaned only if the system's output matches *exactly* with the ground truth. For unwanted rows, the correct operation is deletion.

As noted in Section 2, no prior work addresses automatic cleaning of ill-formed rows in CSV files. Thus, we compared TASHEEH with a BASELINE transformation strategy that employs a simplified set of operations, underlined in Table 3.

To evaluate the transformation performance of both TASHEEH and the BASELINE, we opted to use TASHEEH as the row classifier, since it outperformed the other row classifiers. Additionally, we evaluated our approach using a PERFECT row classifier with manually annotated ground truth for comparison.

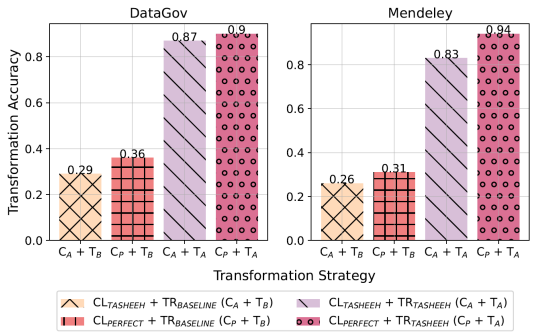

Figure 5: BASELINE and TASHEEH transformation effectiveness with TASHEEH and PERFECT row classifiers

5.3.1 *Ill-formed rows transformation evaluation*

Figure 5 presents the results for the DataGov and Mendeley datasets, showing the performance of both transformation strategies combined with the TASHEEH and the PERFECT classifiers. We observed similar results for the UKGov and GitHub datasets. The BASELINE strategy performed well in files with errors limited to unwanted rows, where only deletion was required, and in cases involving padding cells. Overall, the combination of TASHEEH transformation and a PERFECT classifier achieved the best results. However, using TASHEEH for both classification and transformation produced comparable performance, underscoring the effectiveness of its classifier and transformation strategy in handling ill-formed rows. Further details can be found in [17].

5.3.2 *SRFN - TASHEEH comparison*

As described in Section 2, SRFN is the only other system that addresses a specific type of structural inconsistency in data rows: entries shifted into incorrect columns by "swapping repair using a fixed set of neighbors" [42]. We evaluated SRFN using artifacts from its GitHub repository[5] on our dataset, as the authors' dataset was unavailable. SRFN requires users to specify fixed attributes, rows for repair, and the number of nearest neighbors (k). Following the authors' guidelines, we tested k values from 2 to 864, depending on file length. While SRFN can address issues like swapping misplaced name and passport values, it failed to resolve any inconsistencies (e.g., shifted values) in our datasets across all settings, achieving an overall transformation accuracy of 0.

5.4 Usability Case Study

We conducted a user study measuring the time and accuracy of cleaning raw data files both manually and with TASHEEH. We invited five computer scientists with data cleaning expertise, not involved in our project, to clean a random sample of ten files from our real-world datasets, using any tool they preferred. Manually cleaning required a significant amount of time, averaging 67±18 minutes across all experts.

[5]`https://github.com/SwappingRepair/SRFN`

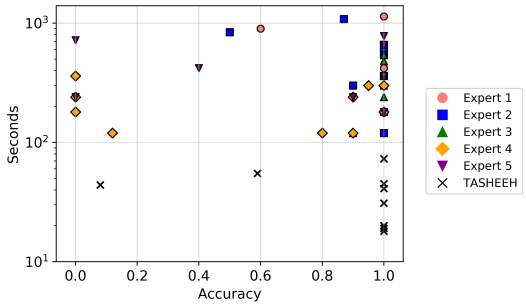

Figure 6: Results of our usability study, comparing time and accuracy of manual cleaning with TASHEEH.

Additionally, the average accuracy achieved is 83 ± 17 %: sometimes experts simply removed the inconsistencies they did not understand, e.g., misplaced delimiter. In contrast, with TASHEEH, the accuracy is significantly higher, with 8 out of 10 files achieving a perfect cleaning result, averaging 87% at a fraction of the time: averaging 6.80 ± 0.34 minutes (across three experimental runs). To summarize, the use of TASHEEH not only significantly reduces the overall cleaning time but also delivers improved accuracy compared to a fully manual approach.

6. CONCLUSION

While many data science pipelines and data cleaning methods assume data to be preloaded into some system, data practitioners know that even achieving that premise is both challenging and tedious. Our work introduces TASHEEH, a data preparation system designed to identify and clean ill-formed data rows in raw CSV files, thus pushing the boundary of automatic data cleaning to the very beginning of data processing. TASHEEH utilizes the pattern language introduced in our previous work SURAGH [15], which classifies rows as either ill-formed or well-formed, based on the dominant row patterns. TASHEEH further classifies the ill-formed rows as *wanted* (data) or *unwanted* (non-data) and ultimately repairs the structural inconsistencies in the ill-formed wanted rows using a pattern transformation algebra.

TASHEEH automatically generates accurate transformations for 86% of ill-formed rows across all files, thus automatically recovering much data that could otherwise not be ingested.

Building on its robust capabilities, TASHEEH can serve as a critical pre-processing step for integrating raw CSV data into modern data platforms. Tools like Pandas [37], which is widely used in machine learning workflows, and open-source database systems, such as DuckDB [36], are effective for parsing structured files, but skip or log faulty rows when structural issues occur. Similarly, modern data integration platforms like Airbyte [21], though adept at connecting diverse data sources, rely on well-structured inputs for seamless operation. These limitations emphasize the necessity for systems like TASHEEH, which can proactively address structural inconsistencies, effectively bridging the gap between raw data and seamless integration into data pipelines.

Acknowledgments

This research was funded by the HPI research school on Data Science and Engineering.

7. REFERENCES

[1] Abdulrazaq Hassan Abba and Mohammed Hassan. Design and implementation of a csv validation system. In *Proceedings of the International Conference on Applications in Information Technology (ICITA)*, pages 111–116, 2018.

[2] Marco D Adelfio and Hanan Samet. Schema extraction for tabular data on the web. *PVLDB*, 6(6):421–432, 2013.

[3] Sara Bonfitto, Luca Cappelletti, Fabrizio Trovato, Giorgio Valentini, and Marco Mesiti. Semi-automatic column type inference for csv table understanding. In *International Conference on Current Trends in Theory and Practice of Informatics*, pages 535–549. Springer, 2021.

[4] Christina Christodoulakis, Eric B Munson, Moshe Gabel, Angela Demke Brown, and Renée J Miller. Pytheas: pattern-based table discovery in csv files. *PVLDB*, 13(12):2075–2089, 2020.

[5] Xu Chu, Yeye He, Kaushik Chakrabarti, and Kris Ganjam. Tegra: Table extraction by global record alignment. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 1713–1728, 2015.

[6] Xu Chu, John Morcos, Ihab F Ilyas, Mourad Ouzzani, Paolo Papotti, Nan Tang, and Yin Ye. Katara: A data cleaning system powered by knowledge bases and crowdsourcing. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 1247–1261, 2015.

[7] Michele Dallachiesa, Amr Ebaid, Ahmed Eldawy, Ahmed Elmagarmid, Ihab F Ilyas, Mourad Ouzzani, and Nan Tang. NADEEF: a commodity data cleaning system. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 541–552, 2013.

[8] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[9] Till Döhmen, Hannes Mühleisen, and Peter Boncz. Multi-hypothesis csv parsing. In *Proceedings of the International Conference on Scientific and Statistical Database Management (SSDBM)*, pages 1–12, 2017.

[10] Haoyu Dong, Shijie Liu, Shi Han, Zhouyu Fu, and Dongmei Zhang. Tablesense: Spreadsheet table detection with convolutional neural networks. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, volume 33, pages 69–76, 2019.

[11] Hazem Elmeleegy, Jayant Madhavan, and Alon Halevy. Harvesting relational tables from lists on the web. *PVLDB*, 2(1):1078–1089, 2009.

[12] Chang Ge, Yinan Li, Eric Eilebrecht, Badrish Chandramouli, and Donald Kossmann. Speculative distributed csv data parsing for big data analytics. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 883–899, 2019.

[13] Martin Gollery. Bioinformatics: sequence and genome analysis. *Clinical Chemistry*, 51(11):2219–2220, 2005.

[14] Mazhar Hameed and Felix Naumann. Data preparation: A survey of commercial tools. *SIGMOD Record*, 49(3):18–29, 2020.

[15] Mazhar Hameed, Gerardo Vitagliano, Lan Jiang, and Felix Naumann. Suragh: Syntactic pattern matching to identify ill-formed records. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, pages 143–154, 2022.

[16] Mazhar Hameed, Gerardo Vitagliano, and Felix Naumann. Morpher: Structural transformation of ill-formed rows. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*, pages 5051–5055, 2023.

[17] Mazhar Hameed, Gerardo Vitagliano, Fabian Panse, and Felix Naumann. TASHEEH: Repairing row-structure in raw CSV files. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, pages 426–439, 2024.

[18] Yeye He, Xu Chu, Kris Ganjam, Yudian Zheng, Vivek Narasayya, and Surajit Chaudhuri. Transform-data-by-example (TDE) an extensible search engine for data transformations. *PVLDB*, 11(10):1165–1177, 2018.

[19] Joseph M Hellerstein, Jeffrey Heer, and Sean Kandel. Self-service data preparation: Research to practice. *IEEE Data Engineering Bulletin*, 41(2):23–34, 2018.

[20] RFC 4180. `https://tools.ietf.org/html/rfc4180`, 2005. (last accessed January 19th, 2025).

[21] Airbyte Inc. Airbyte: The open-source data integration platform. `https://airbyte.com`. (last accessed January 19, 2025).

[22] Google Inc. Openrefine. `www.openrefine.org`. (last accessed January 19th, 2025).

[23] Trifacta Inc. Trifacta data engineering cloud (now alteryx designer cloud after acquisition). `www.trifacta.com`. (last accessed January 19th, 2025).

[24] Lan Jiang, Gerardo Vitagliano, and Felix Naumann. Structure detection in verbose csv files. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, pages 193–204, 2021.

[25] Elvis Koci, Maik Thiele, Wolfgang Lehner, and Oscar Romero. Table recognition in spreadsheets via a graph representation. In *IAPR International Workshop on Document Analysis Systems (DAS)*, pages 139–144. IEEE, 2018.

[26] Elvis Koci, Maik Thiele, Oscar Romero, and Wolfgang Lehner. Cell classification for layout recognition in spreadsheets. In *International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K)*, pages 78–100. Springer, 2019.

[27] Elvis Koci, Maik Thiele, Oscar Romero, and Wolfgang Lehner. A genetic-based search for adaptive table recognition in spreadsheets. In *International Conference on Document Analysis and Recognition (ICDAR)*, pages 1274–1279. IEEE, 2019.

[28] Sanjay Krishnan, Jiannan Wang, Eugene Wu, Michael J Franklin, and Ken Goldberg. ActiveClean: Interactive data cleaning for statistical modeling. *PVLDB*, 9(12):948–959, 2016.

[29] Johann Mitlöhner, Sebastian Neumaier, Jürgen Umbrich, and Axel Polleres. Characteristics of open data csv files. In *Proceedings of the International Conference on Open and Big Data (OBD)*, pages 72–79. IEEE, 2016.

[30] Saul B Needleman and Christian D Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48(3):443–453, 1970.

[31] Sebastian Neumaier, Axel Polleres, Simon Steyskal, and Jürgen Umbrich. Data integration for open data on the web. In *Reasoning Web International Summer School*, pages 1–28. Springer, 2017.

[32] Dan Olteanu. The relational data borg is learning. *PVLDB*, 13(12):3502–3515, 2020.

[33] David Pinto, Andrew McCallum, Xing Wei, and W Bruce Croft. Table extraction using conditional random fields. In *Proceedings of the International Conference on Information retrieval (SIGIR)*, pages 235–242, 2003.

[34] Gil Press. Cleaning data: Most time-consuming, least enjoyable data science task. *Forbes*, March 2016.

[35] Abdulhakim A Qahtan, Ahmed Elmagarmid, Raul Castro Fernandez, Mourad Ouzzani, and Nan Tang. FAHES: A robust disguised missing values detector. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 2100–2109, 2018.

[36] Mark Raasveldt and Hannes Mühleisen. DuckDB: an embeddable analytical database. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 1981–1984, 2019.

[37] Jeff Reback, jbrockmendel, Wes McKinney, Joris Van den Bossche, Matthew Roeschke, Tom Augspurger, Simon Hawkins, Phillip Cloud, gfyoung, Sinhrks, Patrick Hoefler, Adam Klein, Terji Petersen, Jeff Tratner, Chang She, William Ayd, Shahar Naveh, JHM Darbyshire, Richard Shadrach, Marc Garcia, Jeremy Schendel, Andy Hayden, Daniel Saxton, Marco Edward Gorelli, Fangchen Li, Torsten Wörtwein, Matthew Zeitlin, Vytautas Jancauskas, Ali McMaster, and Thomas Li. pandas-dev/pandas: Pandas 1.4.3, June 2022.

[38] Theodoros Rekatsinas, Xu Chu, Ihab F Ilyas, and Christopher Ré. HoloClean: Holistic data repairs with probabilistic inference. *PVLDB*, 10(11):1190–1201, 2017.

[39] Yoones A Sekhavat, Francesco Di Paolo, Denilson Barbosa, and Paolo Merialdo. Knowledge base augmentation using tabular data. In *Proceedings of the Workshop on Linked Data on the Web (LDOW)*, 2014.

[40] Vraj Shah and Arun Kumar. The ML data prep zoo: Towards semi-automatic data preparation for ML. In *Proceedings of the International Workshop on Data Management for End-to-End Machine Learning (DEEM)*, pages 1–4, 2019.

[41] Elias Stehle and Hans-Arno Jacobsen. ParPaRaw: Massively parallel parsing of delimiter-separated raw data. *PVLDB*, 13(5):616–628, 2020.

[42] Yu Sun, Shaoxu Song, Chen Wang, and Jianmin Wang. Swapping repair for misplaced attribute values. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 721–732. IEEE, 2020.

[43] LLC Tableau Software. Tableau. `www.tableau.com`. (last accessed January 19th, 2025).

[44] Nan Tang, Ju Fan, Fangyi Li, Jianhong Tu, Xiaoyong Du, Guoliang Li, Sam Madden, and Mourad Ouzzani. RPT: Relational pre-trained transformer is almost all you need towards democratizing data preparation. *PVLDB*, 14(8):1254–1261, 2021.

[45] Ignacio G Terrizzano, Peter M Schwarz, Mary Roth, and John E Colino. Data wrangling: The challenging journey from the wild to the lake. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*, 2015.

[46] Gerrit JJ van den Burg, Alfredo Nazábal, and Charles Sutton. Wrangling messy csv files by detecting row and type patterns. *Data Mining and Knowledge Discovery*, 33(6):1799–1820, 2019.

[47] Gerardo Vitagliano, Mazhar Hameed, Lan Jiang, Lucas Reisener, Eugene Wu, and Felix Naumann. Pollock: A data loading benchmark. *PVLDB*, 16(8):1870–1882, 2023.

[48] Robert A Wagner and Michael J Fischer. The string-to-string correction problem. *Journal of the ACM (JACM)*, 21(1):168–173, 1974.