TASHEEH: Repairing Row-Structure in Raw CSV Files

Mazhar Hameed
mazhar.hameed@hpi.de
Hasso Plattner Institute, University of Potsdam
Germany

Gerardo Vitagliano
gerardo.vitagliano@hpi.de
Hasso Plattner Institute, University of Potsdam
Germany

Fabian Panse
fabian.panse@hpi.de
Hasso Plattner Institute, University of Potsdam
Germany

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

ABSTRACT

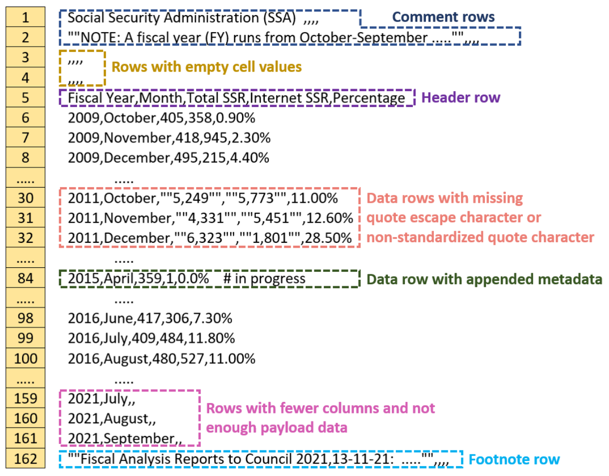
Comma-separated value (CSV) files follow a useful and widespread format for data exchange due to their flexible standard. However, due to this flexibility and plain text format, such files often have structural issues, such as unescaped quote characters within quoted fields, columns containing different value formats, rows with different numbers of cells, etc. We refer to rows that contain such structural inconsistencies as *ill-formed*. Consequently, ingesting them into a host system, such as a database or an analytics platform, often requires prior data preparation steps.

Traditionally, data scientists write custom code to clean ill-formed rows, even before they can use data cleaning tools and libraries, which assume all data to be properly loaded. These tasks are tedious and time-consuming, requiring expertise and frequent human intervention. To automate this process, we propose 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. Of 200 351 manually annotated rows from four different sources, Tasheeh was able to correctly detect 95.53% of data rows and accurately generate transformations for 87.83% of them.

1 ANOMALOUS ROW STRUCTURES

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 [6, 7, 41, 44, 45, 54, 59]. However, this flexibility comes with much responsibility and effort for data engineers and analysts during pre-processing: due to their loose format, these files appear in various dialects [13, 17, 65] deviating from the RFC standard [29]. In addition, they often contain various structural inconsistencies [21, 31]. Consequently, it is challenging to load these files correctly into data-driven systems without prior data preparation steps [20, 66]. Developers and scientists spend much of their development time cleaning and organizing data in these files and have little time for analytical tasks [26, 48, 63].

Recently, our community has begun recognizing such shortcomings as research opportunities and has developed solutions for preparing and cleaning data [8, 10, 11, 21, 31, 38, 50, 53, 55, 62]. However, we are still far from creating a fully automated data processing pipeline, in part due to open challenges of raw CSV files.



```
1    Social Security Administration (SSA) ,,,,
2    ""NOTE: A fiscal year (FY) runs from October-September .....""",,,
3    ,,,,
4    ,,,,
5    Fiscal Year,Month,Total SSR,Internet SSR,Percentage
6    2009,October,405,358,0.90%
7    2009,November,418,945,2.30%
8    2009,December,495,215,4.40%
.....      .....
30   2011,October,""5,249"",""5,773"",11.00%
31   2011,November,""4,331"",""5,451"",12.60%
32   2011,December,""6,323"",""1,801"",28.50%
.....      .....
84   2015,April,359,1,0.0%   # in progress
.....      .....
98   2016,June,417,306,7.30%
99   2016,July,409,484,11.80%
100  2016,August,480,527,11.00%
.....      .....
159  2021,July,,
160  2021,August,,
161  2021,September,,
162  ""Fiscal Analysis Reports to Council 2021,13-11-21:  .....""",,,
```

Comment rows
Rows with empty cell values
Header row
Data rows with missing quote escape character or non-standardized quote character
Data row with appended metadata
Rows with fewer columns and not enough payload data
Footnote row

Figure 1: A sample of a raw CSV file with *ill-formed* rows due to structural inconsistencies at both column- and row-levels.

As we noted in [21], among other challenges, detecting and cleaning "ill-formed" rows (see Section 3.3 for a formal definition) in CSV files are difficult problems. *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. We highlight 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.

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 [21], 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[1]. The goal of Tasheeh is to improve the classification of ill-formed rows by recognizing wanted and unwanted ill-formed rows, but in particular automatically *clean* wanted rows structure.

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

Real-world CSV files are quite "wild" because of their flexible standard [21]. For example, due to structural inconsistencies (ill-formed rows) the file in Figure 1, cannot be directly ingested correctly even into many advanced data preparation and cleaning tools, such as Trifacta Wrangler [30], Tableau [61], or OpenRefine [19]. Not to mention loading it into a relational DBMS, which would complain about almost every row. The ingestion challenges stem from the fact that existing data-driven systems for parsing CSV files usually assume that the file structure follows the RFC standard [29], without considering the variations that may exist in practice [7]. This assumption can lead to significant structural challenges during the file ingestion process, such as shifted column values, incorrect field boundaries, misinterpreted quotes and escape characters, fields spanning to multiple rows, data mixed with metadata, rows with varying lengths, etc. [66]. In the following, we discuss structural challenges using the example file shown in Figure 1.

Example 1.1. The 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 4180 standard [29] for CSV files states: 1) Each field must be enclosed in double-quotes if its value contains a character used as a field delimiter; 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 literal string `"5,249"` (with quotations included). Loading the file as-is in a downstream application might lead to a shift in column values.

Moreover, Sun et al. noted that shifts in values can also occur when extracting data from multiple sources, during manual data entry, or when sensors fail [60].

Example 1.2. Another example of structural inconsistency in the file of Figure 1 (row 84) is data and metadata appearing in the same row. 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.

Out of a random subset of 1 000 files from www.data.gov, we found that in about 5.8% data and metadata were present in the same rows. Such additional metadata are not the only reason for different numbers of columns in rows. Due to the flexible format of CSV files, users add additional columns for data and explanations to the data by simply adding delimiters to the rows. Some rows contain fewer columns due to missing values or deletion operations causing the same problem. In another random subset of 1 000 files from Mendeley's data sharing platform[2], we observed that around 7.3% contained *wanted* rows with a varying number of columns.

Preparing data with such structural inconsistencies for loading into data-driven applications is a challenging and time-consuming task that often requires significant manual effort. 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.

Our paper 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, each annotated for ill-formed or well-formed, and wanted or unwanted rows for a total of 200 351 rows. The files, together with the classification annotations, manually cleaned wanted rows, and code, are publicly available[3].
(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 work. 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. Finally, Section 6 concludes our study.

2 RELATED WORK

While a considerable amount of research has been conducted on detecting and correcting semantic errors in data, little attention has been given to addressing the structural problems in CSV files. However, 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 presents an interesting problem, leading to the development of multiple tools [7, 8, 14, 15, 35, 37]. In particular, TEGRA [8] (for web lists), TABLESENSE [14] (for spreadsheets), and PYTHEAS [7] (for CSV files) have achieved considerable success in the table extraction domain. TEGRA approaches table extraction as a global optimization problem. Its function searches for the best position to split every row to ensure column alignment and coherence of values. TABLESENSE is a deep learning architecture that leverages a combination of visual and rich-text Excel features to accurately identify and segment tables within spreadsheets. PYTHEAS is the state-of-the-art system dedicated to extracting tables from CSV files. It utilizes machine-learned rules to discover tables in CSV files by identifying the position of data rows.

Although these systems do not explicitly focus on cleaning structural issues, which is the primary objective of our research, we compare our approach with the state-of-the-art system PYTHEAS, since (1) it is designed for plain text files, and (2) the system's ability to identify data rows in CSV files allows for a direct comparison with our classification component (Section 5.2). Conversely, TEGRA operates under the assumption that the delimiter is the only potential structural issue, and it also expects the delimiter to be consistent across all rows, and TABLESENSE expects a spreadsheet format as input, where cell boundaries are typically

[2]https://www.mendeley.com/

[3]https://github.com/HMazharHameed/TASHEEH

well-defined and distinct, which contrasts with the less structured nature of plain text files like CSV. Therefore, we believe applying the latter systems would not lead to a fair comparison.

Row and cell type detection. In plain text files like CSV, not every row may contain data [41]. Therefore, accurately identifying the boundaries of cells and rows and comprehending their underlying semantics are essential features for efficient data processing. Researchers have devised various tools employing both supervised and unsupervised approaches to classify cells and rows within tabular data [4, 20, 31, 36, 47]. Among these approaches, Jiang et al. proposed the state-of-the-art Strudel approach [31]: Strudel is a multi-class random forest classifier that leverages three types of features: content, context, and computational features to accurately 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 [66]. Among these, one notable issue is the occurrence of shifted column values. Sun et al. introduced the SRFN system [60] 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. It is the sole solution available 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.

Recent advances in natural language processing, such as the development of large language models (LLM), exemplified by the GPT family [51], have sparked interest in using such models for traditional data wrangling and cleaning tasks. The underlying idea is to utilize pre-trained LLMs and employ zero-shot or few-shot inference techniques to address various data management tasks. In Section 5.3 we provide a brief overview of our experience utilizing these models to address structural inconsistencies.

Data transformation. Data transformation has been a long-standing challenge in research, with various proposals to address it. Notable among these is the "transform-by-example" method, which allows users to provide input/output examples for the system to search for consistent programs [3, 22, 23, 32, 33, 56]. However, these approaches expect the input data to already be in relational table format, which the system can then analyze and transform accordingly. In contrast, the CSV files we focus on in this research often exhibit various structural problems that make them challenging to parse by these systems, let alone apply transformations on them. Our system complements the existing research on data transformation by transforming the structurally broken CSV files into a consistent format to be then loaded into these transformation tools.

Error detection and correction. The importance of detecting and correcting data quality issues has been widely acknowledged in the research community [2, 25]. Numerous error detection techniques have been proposed [9, 24, 27, 28, 40, 50, 68], as well as error correction methods [10, 16, 34, 39, 46, 49, 53]. Again, these techniques rely on structurally sound data as input. Our system addresses this challenge by providing a solution for resolving structural inconsistencies in CSV files, thus enabling these downstream data quality techniques to parse them correctly for subsequent operations.

3 PRELIMINARIES

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 introductory step in Tasheeh.

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, two tuples can also contain distinct data from the same row. Formalizing these concepts, we define wanted and unwanted rows as follows:

Definition 3.1. 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, making it unsound. 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 [21]. Suragh takes a CSV file as input and classifies its rows as ill- or well-formed based on the dominant 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. In Figure 2, the output of Tasheeh's classification process is merged with the results of Suragh to minimize visual clutter.

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, such as filling null values, disambiguating values, or normalizing addresses. These semantic transformations can be applied later in the pre-processing pipeline, leveraging data cleaning tools and libraries [3, 19, 22, 30, 33, 34, 53, 57, 58, 61] that are specifically designed to perform such transformations.

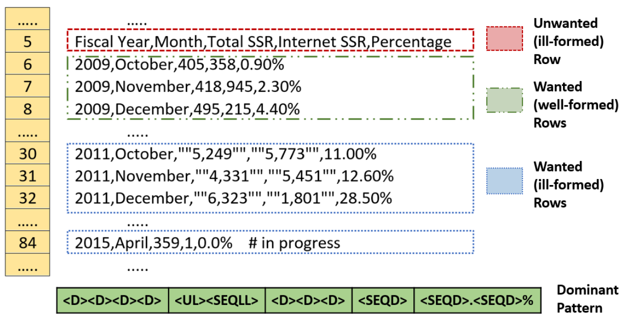

```
.....        .....
5            Fiscal Year,Month,Total SSR,Internet SSR,Percentage
6            2009,October,405,358,0.90%
7            2009,November,418,945,2.30%
8            2009,December,495,215,4.40%
.....        .....
30           2011,October,""5,249"",""5,773"",11.00%
31           2011,November,""4,331"",""5,451"",12.60%
32           2011,December,""6,323"",""1,801"",28.50%
.....        .....
84           2015,April,359,1,0.0%    # in progress
.....        .....
```

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

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 transform 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 in the single abstraction $\langle SEQUL\rangle$, for "Sequence of Upper Letters". Using the given pattern grammar, Suragh generates patterns for each cell value, referred to as *syntactic cell patterns*.

The application of the grammar rules is order-dependent, and every input value can be transformed into one or more syntactic cell patterns. For example, for the cell value "`June`" of the column "`Month`" in Figure 1, three possible cell patterns are $\langle UL\rangle\langle LL\rangle\langle LL\rangle\langle LL\rangle$, $\langle UL\rangle$ $\langle SEQLL\rangle$, and $\langle TXT\rangle$. Note that different patterns have different levels of "specificity", depending on the level of abstractions.

3.3 Pattern Extraction

After generating patterns for each cell value, Suragh collects them for an entire column, referred to as *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 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 selecting a single 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, selecting among 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 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 [21]. Tasheeh uses dominant row patterns as a filter in the pattern classification phase and as the target of the pattern transformation phase (see Section 4).

4 THE TASHEEH SYSTEM

Tasheeh performs in three phases (see Figure 3). In the first phase, it first uses Suragh to classify input file rows as ill-formed or well-formed using *dominant* row patterns ($\mathcal{P}_d$). Then, it runs Suragh incrementally for ill-formed rows to obtain row patterns specifically for those rows; we call these patterns *potential* row patterns ($\mathcal{P}_p$): these ill-formed data rows can possibly be transformed into well-formed data rows. Tasheeh repeats the incremental pattern generation process until no ill-formed rows are left without a potential pattern. Section 4.1 provides details for this step.

The second phase uses the incrementally generated patterns to classify ill-formed rows into wanted and unwanted ones. It leverages a pattern-level distance measure inspired by sequence alignment [18]. This pattern sequence alignment helps Tasheeh determine the extent to which ill-formed rows differ structurally from well-formed rows. 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 a set of dominant row patterns for a given file. Using these dominant patterns, it classifies rows as ill-formed and well-formed. During this process, to reduce the dominant pattern search space, Suragh retains only the dominant patterns and discards all other patterns, including those for ill-formed rows. However, Tasheeh requires these patterns for two reasons: (1) The level of abstraction of these patterns makes the comparison with dominant patterns more applicable than with original rows. (2) Transforming general patterns that cover multiple ill-formed rows is more efficient and scalable than transforming rows individually.

To generate such further patterns, Tasheeh incrementally executes Suragh. In each iteration, Tasheeh revises the criteria for classifying rows based on the dominant patterns generated in that cycle. This iterative process entails discarding previously identified well-formed rows and re-assessing rows previously labeled as ill-formed. As a result, the definition of well-formedness dynamically adapts with each iteration, guided by the remaining rows that have yet to be classified. 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.

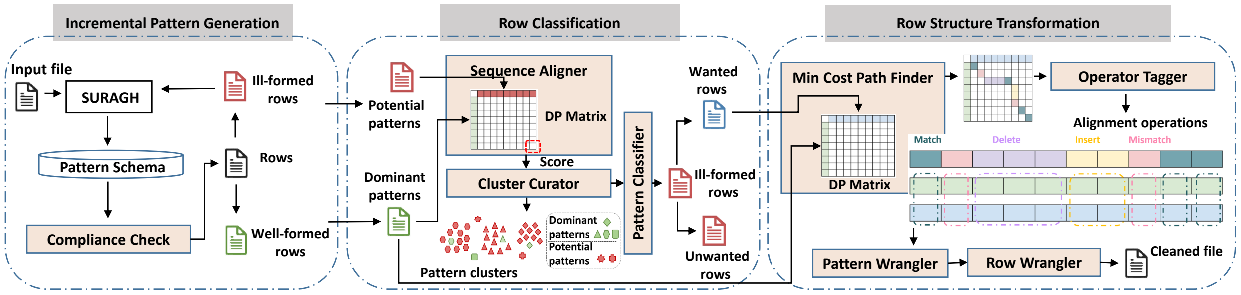


Figure 3: The workflow of Tasheeh

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`				

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

4.2 Row Classification

In this phase, Tasheeh first calculates the *minimum pattern-level distance* between dominant and potential patterns using a dynamic programming approach. Then, the distance score is used to find for each potential pattern the closest corresponding dominant pattern as a target for transformation. Finally, Tasheeh classifies the potential patterns and their associated rows as ill-formed wanted or ill-formed unwanted, based on the pattern distance.

4.2.1 Pattern sequence aligner. The potential patterns that Tasheeh generated in the previous phase may or may not cover rows that contain data. To understand how similar they are to dominant patterns, which do cover data rows, we introduce a pattern-level distance measure inspired by sequence alignment [18]. Sequence alignment frameworks arrange the characters of two sequences to maximize the number of matching characters [43], calculating the similarity between the sequences using dynamic programming [5].

Like edit distance frameworks for string matching [67], sequence alignment frameworks also expect a pair of input sequences and apply the required operation. These frameworks aim at computing the closest possible match between characters of the input sequences so that the overall character-wise distance is minimized. To align the sequences, a set of operations includes "match", "mismatch", and "indel" (insertion and deletion; represented by a gap character "-"), with a given cost for each operation. This cost may be fixed or may vary depending on the user definition.

We introduce a distance-based alignment framework for pattern-by-pattern alignments, where the input sequences are entire row patterns. The input patterns are compared column by column, splitting them on the delimiter character of the raw CSV file.

Let us consider the dominant pattern $\mathcal{P}_d$ and a potential pattern $\mathcal{P}_p$ from Table 1. Our distance-based alignment framework generates $\mathcal{P}'_d$ and $\mathcal{P}'_p$ with the same number of column patterns so that they can be aligned. To do so, the framework pads column patterns using a special gap character to the shorter of the two sequences based on the minimum cost edit path. We implemented our framework using a dynamic programming approach to find the alignment with the lowest distance, similar to other sequence alignment distances [43]. To find an alignment between the two patterns $\mathcal{P}_d, \mathcal{P}_p$, 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 are calculated as 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). Given two column patterns α, β:

(1) If α is equal to β, their distance is 0.
(2) If either α or β are the gap "-" pattern, or contain a null value pattern ⟨*EV*⟩, their distance is 1.
(3) If α contains the delimiter ⟨*DEL*⟩ and β contains data (or vice versa), their distance is also 1.
(4) If both column patterns represent data and do not contain any delimiters or gaps, we first obtain the similarity score between the column patterns as the number of common abstractions in the same class (letters, digits, special characters) divided by the maximum length of the two column patterns. We then define column pattern distance as 1 minus the similarity score [range: 0-1].

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

The pattern distance formula is inspired by the string-by-string alignment approach [22], which we adapted to define the distance function $\mathcal{D}$ between syntactic column patterns (α, β) using abstractions [21]. We consider the typical three groups of abstraction classes: letters "l", digits "d", and symbols "s" (see details in [21]). 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 edit, 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.

Example 4.1. Table 2 demonstrates the pattern-by-pattern distance scores between the dominant pattern $\mathcal{P}_d$ and potential pattern $\mathcal{P}_{p3}$ from Table 1, where the column pattern distance scores were determined through the use of Equation (2). The overall row pattern distance was determined as $(0+0+0+0+0+0+0+0+0.90)/9 = 0.10$ by applying Equation (3).

Table 2: Column pattern-wise distance scores between row patterns $\mathcal{P}_d$ (Figure 2 → rows 6-8) and $\mathcal{P}_{p3}$ (Figure 2 → row 84).

Aligned Pattern Pairs		Distance Score	Freq.
$\mathcal{P}_d$ column patterns	$\mathcal{P}_{p3}$ column patterns		
$\langle D\rangle\langle D\rangle\langle D\rangle\langle D\rangle$	`2015`	0	1
$\langle DEL\rangle$	$\langle DEL\rangle$	0	4
$\langle UL\rangle\langle SEQLL\rangle$	$\langle UL\rangle\langle SEQLL\rangle$	0	1
$\langle D\rangle\langle D\rangle\langle D\rangle$	`359`	0	1
$\langle SEQD\rangle$	`1`	0	1
$\langle SEQD\rangle.\langle SEQD\rangle\%$	`0.0% # in progress`	0.90	1

Potential patterns that are close to the dominant pattern have a lower distance score. To classify wanted and unwanted patterns, we introduce a distance score threshold: rows whose potential pattern is not too different from the dominant pattern, i.e., with a distance lower than the threshold, are considered "wanted", and all others are considered "unwanted" (see Section 4.2.3 below for more details).

4.2.2 Cluster curator. In cases with only one dominant pattern, such as in the example file shown in Figure 2, all potential patterns are aligned to the single dominant pattern to determine their distance. However, when files have multiple dominant patterns, the distances between each combination of dominant and potential patterns are calculated, and for each potential pattern, we choose the dominant pattern with the lowest distance.

4.2.3 Pattern classification. After obtaining the pattern distance for each combination of dominant and potential patterns, Tasheeh passes it to the pattern classifier, which uses a distance score threshold to determine whether the potential patterns are wanted or unwanted, resulting in whether the corresponding rows contain data or not. Given a distance score threshold θ, all potential patterns with a distance score $\leq \theta$ are labeled as wanted, while the rest are considered as unwanted. With the experiments detailed in Section 5.2 we determined that a threshold value $\theta = 0.3$ yielded the highest F-1 score.

Table 3: 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.

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 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 corresponding wanted rows into well-formed ones.

4.3.1 Pattern transformation algebra. Table 3 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.

Each operator has a specific function based on the inconsistencies that need to be resolved in column pattern(s). For some inconsistencies, multiple operators may have to be combined in a specific execution order. For example, to correct shifted column values, Tasheeh uses the functionalities of Merge, Re-delimit, Re-quote, Re-escape, and Drop. Here, the execution order is important because if the Drop operator precedes the Merge operator, data are lost. Similarly, without the Re-delimit operator before the Merge operator, shifted values are not fixed, and even worse, shifted further due to incorrect field boundaries. In the following section, we explain how we obtain a complete pattern-level edit path and the functionalities of the transformation operators.

4.3.2 Minimum cost edit path. To find the alignment between a dominant pattern and a wanted pattern, we trace back from the bottom right of their matrix $\mathcal{M}$ from the previous phase. First, we construct a graph on $\mathcal{M}$ where each node corresponds to a cell of the matrix and contains information about a pair of column patterns from the dominant and the wanted patterns. Each node has three outgoing edges indicating the three possible directions from one node to another based on the alignment operators ("match", "mismatch", "insert", and "delete"), where each edge has its weight based on the cost of the operation required.

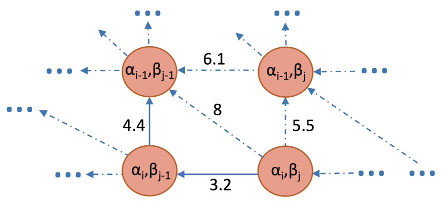


Figure 4: Weighted graph for minimum cost path finder

The use of edge weights provides information about the cost of each transformation, enabling us to choose the best overall alignment between the patterns. An example of a weighted graph with outgoing edges is illustrated in Figure 4. The solid line in the graph represents the path with the minimum cost from one node to another, where every direction provides information about the set of operations required to transform one column pattern into another. For example, the diagonal direction indicates the match and mismatch alignment operations, and for each of them, we can apply a set of transformation operators as shown in Table 4. The determination of which operation to use is based on the column patterns and the distance score. If two column patterns are equal, the distance score is zero, and the diagonal direction indicates a match operation. On the other hand, if the column patterns are different, the direction indicates a mismatch operation.

Similarly, an outgoing vertical direction represents the insertion operation, while an outgoing horizontal direction represents the deletion operation. Both directions are accompanied by a set of transformation operations (see Table 4). In essence, the process involves determining the edge weights to represent the cost (distance) associated with each transformation. Following that, we employed Dijkstra's shortest path algorithm [12] to compute the most efficient path, allowing for the determination of the minimum cost edit path from the initial node to the final node. This path represents the alignment between a dominant pattern and a wanted pattern, as shown in Figure 5. The figure provides a visual representation of the aligned row patterns ($\mathcal{P}_d$, $\mathcal{P}_{p2}$) and the marked alignment operators. This information later will be used by the transformation engine for pattern transformation (see Section 4.3.3). During backtracking, several paths with equal minimum edit costs may be found. We choose the first path returned by the shortest path algorithm, and in the case of a tie between moves, we prioritize the diagonal move if the column patterns being compared are equal.

4.3.3 Pattern wrangler. Tasheeh collects the aligned patterns together with the minimum cost alignment from the previous step. It then passes the aligned column patterns one at a time from the row patterns to the transformation engine, the *pattern wrangler*, which applies the necessary transformations. The transformation engine stores the results of each column pattern transformation in a transformation queue and continues to apply transformations to the remaining column patterns (see Algorithm 1). Once all transformations are complete, it applies the preferred transformations from the queue to the corresponding actual data rows.

Table 4: Pattern sequence alignment operators, where underlined transformation operators were used for both Tasheeh and Baseline transformation strategies.

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

```
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 $T$
1 $T \leftarrow []$
2 foreach $0 \le i \le |\mathcal{P}_p|$ do
3     $transformations \leftarrow$
        $\text{ApplicableTransformations}(\mathcal{P}_d[i], \mathcal{P}_p[i], A[i])$
4     $T_c \leftarrow \text{GenerateCombinations}(transformations)$
5     $T \leftarrow T \cup \arg\min_{c \in T_c} \Big(\text{Distance}\big(\mathcal{P}_d[i],\ c(\mathcal{P}_p[i])\big)\Big)$
6 end
7 return $T$
```

In the following, we explain the alignment operators listed in Table 4 and their corresponding transformation operators in the context of the pattern wrangler.

Match. When aligning row patterns, one often encounters identical column patterns that do not require a transformation. Such patterns are marked with the "match" alignment operator, with the corresponding transformation operator Ignore, which skips the identical column patterns without applying a transformation. For example, in Figure 5, we marked column patterns as "match" if they are identical, indicating that no transformation is required.

Mismatch. Although the "match" and "mismatch" alignment operators follow a diagonal direction, the operator tag in the minimum cost edit path differs for non-identical column patterns, suggesting the need for transformation(s) in the wanted column pattern. However, not every "mismatched" pattern requires a transformation: for example, column patterns "$\langle SEQD\rangle.\langle SEQD\rangle\%$" and "$\langle SEQD\rangle\%$" may appear dissimilar, but can both be used to represent data values within the same column. Transforming these patterns may result in undesired output. Nonetheless, the transformation engine identifies such cases using the abstractions hierarchy and applies the Ignore operator to the mismatched patterns, thereby preventing unintended results.

The column patterns "$\langle SEQD\rangle.\langle SEQD\rangle\%$" and "`0.0% # in progress`" in Table 2 provide an example of where it is necessary to apply transformations to column patterns with the mismatched marked operator. The inconsistency is that the metadata (`# in progress`) are appended to the data part (`0.0%`) in the wanted pattern. Before applying transformations, the engine determines the necessary operators based on the abstractions present in the pattern. In

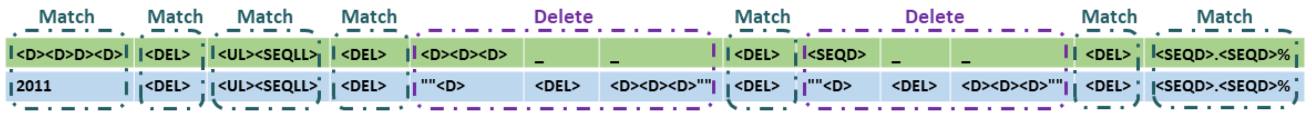


Figure 5: Minimum cost edit path alignment between row patterns $\mathcal{P}_d$ (Figure 2 → rows 6-8) and $\mathcal{P}_{p2}$ (Figure 2 → rows 30-32) together with marked transformation directions.

this case, since the wanted pattern does not contain dialect characters (delimiter, quote, quote escape), other operators, such as Re-quote, Re-escape, Re-delimit, and Re-line are not utilized. The available operators for the transformation engine are Drop, Replace, and Extract. The Drop operator is the least preferred and is used when other operators fail to produce accurate results. The goal is to decrease the pattern distance between the transformed wanted pattern and the dominant pattern, indicating a closer match. Thus, the engine first applies Replace and Extract individually and then in combination, if necessary, to achieve a transformed wanted pattern, closer to the dominant pattern. Note that often a single operator can yield the best results, as in our example, where the Extract operator alone suffices.

After applying the Extract operator to the example column patterns, the resulting pattern is "`0.0%`", which is the desired output. Let us delve into how the Extract operator works. We employ the same sequence alignment framework designed for row patterns, aligning the individual literals, symbols, and abstractions of a pair of column patterns. The table below depicts the alignment between the elements of the example column patterns, with gap characters "-" indicating the deletion operation.

$\langle SEQD \rangle$	.	$\langle SEQD \rangle$	%	-	-	-	-	-	-	-	-	-	-	-	-	-	-
`0`	.	`0`	%		`#`		`i`	`n`		`P`	`r`	`o`	`g`	`r`	`e`	`s`	`s`

The transformation engine then stores the Extract operator in the transformation queue and moves on to the next column pattern.

Insert. As previously stated, it is common for CSV files to have ill-formed wanted rows with a varying number of columns. This inconsistency is manifested in the alignment of the column patterns, where the alignment operator "insert" is marked in the edit path, indicating the insertion of missing columns. The transformation engine uses information from the alignment to identify the position of the missing parts and applies the corresponding operators (Pad, Permute) to resolve this inconsistency. The trivial options are at the beginning or end of the row patterns, where we pad additional column patterns by inserting a field separator.

A more challenging scenario is when the engine needs to add column patterns in the middle of a row pattern, requiring the decision of the appropriate position. Imagine a situation where a file contains sensor data, and new data are frequently being added. Due to a sensor malfunction, some columns are absent, resulting in fewer columns in the impacted rows. The problem is further complicated as the missing parts in the middle cause a backward shift in the column values, resulting in shift inconsistency. In such a scenario, the transformation engine does not merely add separators between columns, but finds the best position by iterating through all the possible indices combinations provided by the alignment (see Algorithm 2).

Delete. If a pattern of the wanted rows has more column patterns than a dominant pattern, the alignment framework inserts gaps in the dominant pattern and marks column patterns with the "delete" alignment operator in the edit path. The presence of such deletions can be attributed to inconsistencies caused by shifted values

Algorithm 2: Pattern Padding

```
  Input: Dominant pattern $\mathcal{P}_d$, Potential pattern $\mathcal{P}_p$,
         Insertion indices $I_n$
  Output: Optimal positions in $\mathcal{P}_p$ for Padding
1 $n\_pad \leftarrow ||\mathcal{P}_d| - |\mathcal{P}_p||$
2 $\mathcal{P}'_p \leftarrow \text{Pad}(\mathcal{P}_p[I_n[0]], n\_pad)$
3 $O \leftarrow \text{PERMUTATIONS}(\mathcal{P}'_p, I_n[0], I_n[|I_n| - 1])$
4 $o^* \leftarrow \arg\min_{o \in O} \Big(\text{DISTANCE}\big(\mathcal{P}_d,\ \text{Permute}(\mathcal{P}'_p, o)\big)\Big)$
5 return $o^*$
```

resulting from missing/broken quotes or quotes escape characters. Or it may be caused by additional columns being appended to the row due to a missing new-line separator. For example, in Figure 5, the wanted column patterns "$""\langle D \rangle \langle DEL \rangle \langle D \rangle \langle D \rangle \langle D \rangle""$" are aligned with the dominant column pattern "$\langle D \rangle \langle D \rangle \langle D \rangle$" and are marked with the delete alignment operator indicating shifted or additional column inconsistency.

The transformation engine starts with the Merge operator and stores the intermediate results by combining all the column patterns and updating the positions of these patterns, which are later used to transform the real data rows. Since potential quote '"' and quote's escape'"' characters are present in the column patterns, the system applies the Re-quote and the Re-escape operators for possible transformations. At present, the transformation operators only support single'" and double quotes'"' as quote and quote's escape characters. We searched through thousands of files from the four repositories we crawled and could not find any file that used other characters for quotes or escaped the quotes. Nevertheless, we can modify the settings to allow for more characters if there are valid file dialects with other characters. The transformation engine follows the RFC 4180 standard [29] specifications as described in Section 1 to standardize the incorrect quote and escape characters.

After the quotes and escapes are standardized, the Re-delimit operator is applied, which replaces the incorrectly treated $\langle DEL \rangle$ with the correct literal symbol and updates the intermediate results in the Merge operator. The final output obtained after applying the transformations is as follows: "$"\langle D \rangle, \langle D \rangle \langle D \rangle \langle D \rangle"$".
The engine then stores the final result in the transformation queue.

4.3.4 *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 experiments, beginning with a description of the datasets and our annotation process. We then present the performance results of our pattern classifier at different distance score thresholds, along with a comparison against a Baseline approach and the state-of-the-art row classifiers. Following this, we present an experimental analysis to demonstrate the efficacy of Tasheeh's transformations again in comparison to our Baseline approach. Finally, we conclude this section with a runtime analysis and a usability case study.

5.1 Datasets and Annotation

We use datasets collected from four open data sources: DataGov, Mendeley, GitHub, and UKGov, leveraging files from our previous work Suragh [21] and supplementing them with additional files. The statistics for each of these data sources are summarized in Table 5. We randomly selected files from each data source and manually inspected them for inconsistencies. To reduce the manual annotation workload, they selected only one file from each group of similarly structured files (e.g., monthly project reports with identical schemata that likely contain similar inconsistencies), resulting in a unique set of diverse files for each data source.

The files for UKGov and GitHub were taken from prior research on dialect detection [65]. We manually checked a random sample of 1 000 CSV files from each dataset, loading each file into a RDBMS and selected those that abort the loading process. After determining files with ill-formed rows in our manual check, we removed files with similar structures and found 24 unique files in the UKGov subset and 28 in the GitHub subset. For the DataGov dataset, we crawled CSV files from www.data.gov. From a random selection of 2 500 files, we observed 449 files with ill-formed rows by loading them into the RDBMS. Again, we sampled 62 files that have a unique structure for our experiments. For the Mendeley dataset, we crawled projects from data.mendeley.com and performed a random selection of 180 projects, which already displayed a large variety of ill-formed rows. While these projects contain various file formats, such as .XML, .XLSX, etc., we focused only on those with at least one CSV file. After manual inspection, we selected 34 unique files from different projects to include in our experiments.

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 ± 1760.4	819.9 ± 1695.5	51.5 ± 173.6	6.2 ± 10.5
Mendeley	34	2909.6 ± 3707.2	2841.4 ± 3690.6	51.0 ± 112.3	17.2 ± 43.4
GitHub	28	662.1 ± 1013.4	627.7 ± 1009.4	17.4 ± 29.5	17.1 ± 41.7
UKGov	24	1186.2 ± 2865.9	1153.3 ± 2845.1	30.4 ± 50.1	2.5 ± 2.3

We extended the annotated data provided in our previous work Suragh [21] following the same annotation strategy. In addition to manually annotating each row as ill-formed or well-formed, we refined the classifications by manually annotating wanted and unwanted rows, and created a ground truth of 200 351 rows across all datasets. Furthermore, we carefully created a ground truth of manually cleaned "wanted" rows that were used for the 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 to compare patterns.

In accordance with Definition 3.1, a detected ill-formed wanted row is considered a true positive if its corresponding row in the ground truth is labeled as an ill-formed wanted row. If not, it is considered a false positive. We use the standard precision P and recall R metrics to assess the effectiveness of our system:

$$P = \frac{|\,\textit{true ill-formed wanted rows detected}\,|}{|\,\textit{true \& false ill-formed wanted rows detected}\,|}$$

$$R = \frac{|\,\textit{true ill-formed wanted rows detected}\,|}{|\,\textit{total ill-formed wanted rows}\,|}$$

In files that have no ill-formed wanted rows, we set the precision and recall scores to 1 if the classifier returned no false positives and no false negatives, respectively, and to 0 otherwise.

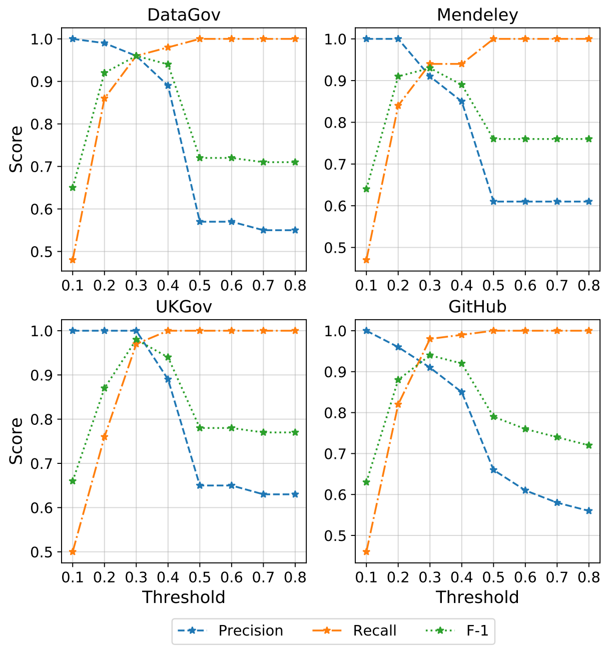


Figure 6: Precision, recall and F-1 measures at different distance score threshold values

We conducted several experiments with different threshold values to determine the best configuration for the classification task. The precision, recall, and F-1 scores at different threshold values for all datasets are displayed in Figure 6. The threshold value "0.3" yielded the highest F-1 score.

Our rationale behind the consistent optimal threshold across all datasets is that it appears to be independent of the datasets themselves. Rather, it seems to serve as a correction factor for potential bias or noise introduced during the Suragh pattern extraction phase. Since row patterns are extracted with a consistent method, the identified threshold is likely compensating for individual wanted rows whose extracted patterns should but do not conform to the dominant pattern in the first place.

In the following, we discuss state-of-the-art row classifiers: Pytheas and Strudel, a popular data analytics tool Pandas, and a straightforward Baseline approach against which we compared our system Tasheeh.

The *Baseline* classifier was inspired by the ad-hoc approach Tabular [1], which uses column counts in the header and other rows to assess row completeness. Based on this idea, our Baseline classifier counts the number of column patterns in the dominant pattern. If the potential pattern has the same number of

Table 6: Row classification comparison overview

Source	# files	# rows	Baseline			Pandas [52]			Pytheas [7]			Strudel [31]			Tasheeh		
			P	R	F-1	P	R	F-1	P	R	F-1	P	R	F-1	P	R	F-1
DataGov	62	54 416	0.42	0.76	0.54	0.56	0.81	0.66	0.75	0.96	0.84	0.87	0.92	0.89	**0.96**	**0.96**	**0.96**
Mendeley	34	98 927	0.39	0.73	0.51	0.49	0.82	0.61	0.71	0.88	0.79	0.79	0.94	0.86	**0.91**	**0.94**	**0.93**
GitHub	28	18 538	0.45	0.74	0.56	0.59	0.77	0.67	0.70	0.84	0.76	0.76	0.87	0.81	**0.91**	**0.98**	**0.95**
UKGov	24	28 469	0.52	0.75	0.62	0.63	0.90	0.74	0.85	0.90	0.87	0.91	0.95	0.93	**1.00**	**0.97**	**0.98**

column patterns as the dominant pattern, it is classified as wanted; otherwise, it is considered to be unwanted.

Pandas is a Python module for data analysis. We compared our approach with Pandas by considering rows that were successfully parsed and loaded into a data frame as wanted, while any other rows the tool skipped[4] due to inconsistencies were considered unwanted. Note that we set the recall score to 1 if Pandas loads every row, thus not misclassifying any unwanted row.

Pytheas is a pattern-based table discovery system that employs a rule-based approach to identify tables and row classes in CSV files [7]. As Pytheas is a supervised learning approach, the authors provided a model with weights, which we used to classify the rows in our dataset. To compare our approach with Pytheas, we considered the rows that Pytheas classified as "data" to be wanted, the others were considered unwanted.

Strudel is a multi-class random forest classifier designed for CSV file row classification [31]. We trained Strudel using the original code and data publicly available at the Strudel project page[5]. To compare our approach with Strudel, we considered the rows classified as "data" by Strudel to be wanted, the others were considered unwanted.

When Pytheas and Strudel classified all rows as data, the recall score was 1, indicating that no wanted rows were misclassified.

Table 6 presents the results of the row classification for all systems. Precision, recall, and F1-score metrics are reported to assess the classification performance of each system. Our proposed approach, Tasheeh, achieved the best performance across all metrics, outperforming both state-of-the-art systems, Pytheas and Strudel. This achievement is particularly significant considering that Pytheas and Strudel use supervised learning approaches that require labeled data for training, while our approach is unsupervised and requires no labeled data. By eliminating the requirement for labeled data, Tasheeh offers greater flexibility and adaptability for analyzing a wide range of CSV files and makes it easier to scale to new datasets without the need for costly and time-consuming labeling efforts.

Pytheas and Strudel faced several challenges in classifying the data rows in some of the files in our collection. For instance, Strudel had difficulty distinguishing between data rows and non-data rows (comments, notes) with the same number of fields as data rows, which were present in some files in our collection. In addition, Pytheas' rules for identifying data rows were based on the majority of the content in each row, and when data and metadata appeared in the same rows, Pytheas misclassified those rows if the majority of the content were metadata. Another challenge for both systems was data rows with fewer columns at the bottom of the file, which were misclassified as notes. Finally, both systems sometimes misclassified rows with new line separators between cell values as group headers.

In situations of very few classification errors, where the system mistakenly identifies unwanted rows as wanted (resulting in false positives), this typically occurs when a dominant pattern and an ill-formed unwanted row mainly comprise the same pattern sequences. For example, when numeric headers like "year values" (e.g.,1990, 1991), are present, and the column values also contain identical information, it becomes challenging to distinguish between the two. Similarly, this challenge applies to cases of solely textual headers, such as "first name" and "last name", when the values in the column contain names. For a wanted row classified as unwanted (a false negative) a scenario occurs when an ill-formed wanted row has additional details in several columns. For example, a dominant pattern may include date format for column values, while the wanted rows contain both date and time format for several columns, leading to a high pattern distance.

5.3 Transformation Performance Evaluation

We evaluate the effectiveness of the transformations using the accuracy metric:

$$A = \frac{|\textit{correctly cleaned ill-formed rows}|}{|\textit{total ill-formed rows}|}$$

A row is considered to be correctly cleaned only if the output produced by the system *matches exactly* the corresponding row in the transformation ground truth, which was manually created. For unwanted rows, the correct cleaning operation is to delete the row.

As we discussed in Section 2, there has been no prior research on automatically cleaning the structure of ill-formed rows in CSV files. Therefore, we compared Tasheeh against a Baseline transformation strategy that uses a simplified set of transformation operations, shown underlined in Table 4.

For evaluating the transformation performance of both Tasheeh and the Baseline transformation strategy, 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. The experiments with the Tasheeh classifier included 7 866 ill-formed rows for DataGov, 3 957 for Mendeley, 1 392 for GitHub, and 2 871 for UKGov, which also included misclassified rows from Suragh. For the Perfect classifier, we used the manually annotated wanted rows for each dataset, with 3 578 for DataGov, 2 319 for Mendeley, 963 for GitHub, and 789 for UKGov. In Figure 7, we present the results of the Baseline and Tasheeh transformation strategies in cleaning ill-formed rows.

Note that with the Perfect classifier for both Baseline and Tasheeh transformation strategies, if a file contains no wanted rows, we set the accuracy score to 1.

[4]Using `on_bad_lines = 'skip'`
[5]https://hpi.de/naumann/s/strudel

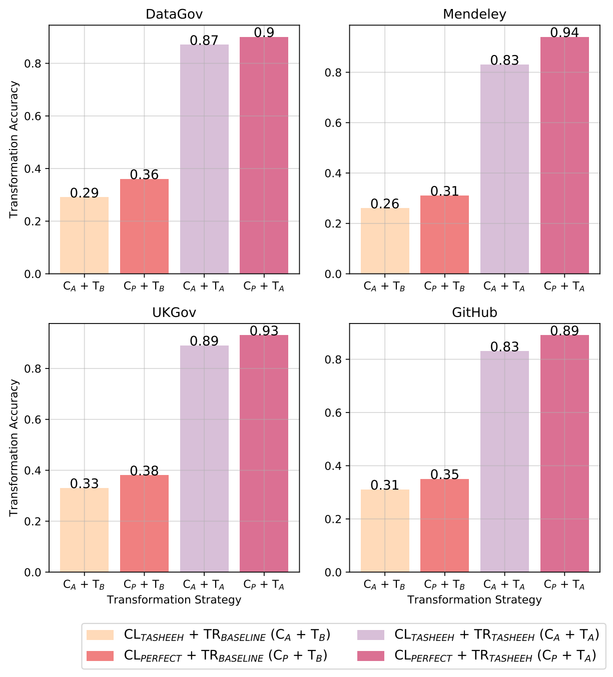


Figure 7: BASELINE and TASHEEH transformation effectiveness with TASHEEH and PERFECT row classifiers

5.3.1 Ill-formed rows transformation evaluation. The evaluation results in Figure 7 highlight the performance of both BASELINE and TASHEEH transformation strategies in combination with both TASHEEH and PERFECT row classifiers.

The performance of the BASELINE strategy was particularly notable in files where the errors were limited to unwanted rows, requiring only the deletion operation. This resulted in a substantial enhancement in the overall performance of the BASELINE strategy. We also observed that the BASELINE strategy performed well in scenarios where padding cells at the start or end of a row pattern was the correct transformation. Additionally, the BASELINE strategy achieved high accuracy when the transformation only involved deleting an entire column pattern.

The experimental results indicate that the combination of TASHEEH transformation with a PERFECT classification achieved the highest overall performance. However, when TASHEEH is used for both classification and transformation, the results are almost as good as those obtained with a PERFECT classifier. These findings highlight the effectiveness of the TASHEEH transformation strategy and its associated classifier in identifying and cleaning ill-formed rows, leading to improved overall performance.

In cases where TASHEEH failed to generate accurate transformations, we observed that the problems mainly stemmed from domain-specific issues. For example, in the Mendeley dataset, the issues were related to the formatting of numbers, where the patterns used scientific notation differently, such as using exponents "E" or "e" in some cases. As a result, the Extract operator removed these values, considering them as non-data parts due to their low pattern frequency. Even in combination with different operators, the Extract operator struggled to capture these domain-specific variations accurately. We also encountered difficulties with complex strings, such as URLs or long addresses, with significantly different patterns in the UKGov, GitHub, and DataGov datasets, resulting in either incorrect information being extracted or dropping the column pattern entirely. These observations highlight the challenge of dealing with complex data types, which requires more understanding of recognizing the data types to handle such variations in patterns effectively. Overall, TASHEEH is a general-purpose system that has effectively handled various data transformation tasks. For more domain-specific cases, TASHEEH can be customized by implementing domain-specific rules.

5.3.2 SRFN - TASHEEH comparison. SRFN is the only other system that addresses a specific type of structural inconsistency in data rows by "swapping repair using a fixed set of neighbors" [60]. SRFN focuses on fixing shifted values in CSV files by leveraging the likelihood of neighboring attribute values and swapping them to determine the correct position. We evaluated the performance of SRFN on our dataset by utilizing the available artifacts on the project's GitHub repository[6]. As access to the author's dataset was not available, we tested the system on our own files only. SRFN requires three inputs to be specified: (1) fixed attributes that should not be modified during the repair process, (2) rows that are to be considered for repair, and (3) the number k of nearest neighbors to be considered. The authors suggest that to determine a proper k, users should test different values on their own data to find the optimal setting. In their paper, the authors tested a minimum of 2 nearest neighbors up to a maximum of 864 neighbors, depending on the length of the file. Following the same setting, we started our experiments with 2 neighbors and tried up to 864 neighbors if the file was longer than 864 rows. We found that SRFN is capable of addressing the problem of swapping misplaced values, such as swapping the position of name and passport values if misplaced. However, for the dataset files used in our experiments, the SRFN system could not fix any inconsistency, e.g., shifted values, even after applying it with all possible parameter settings, having an overall transformation accuracy of 0.

5.3.3 Large language models for structural tasks. There has been increasing interest in leveraging large language models (LLMs) for traditional data wrangling and cleaning tasks. One intriguing aspect is their potential for zero-shot or few-shot inference, where models can perform tasks without specific training on those tasks. However, despite the allure of these capabilities, our own exploratory analysis using the state-of-the-art language model GPT 3.5 (in its version *davinci-003*, like in [42]) revealed several challenges. (1) *Prompt engineering*: The performance of the model was found to be highly sensitive to the specific wording of the input prompt and the content of the file. (2) *Repeatability challenges*: While the prompt engineering process yielded reasonable results, achieving repeatability remains a significant challenge. There is no guarantee that using the same prompt will consistently produce similar outcomes. Multiple attempts with the same prompt often yielded different results, making it difficult to replicate and rely on specific outcomes. (3) *Reproducibility challenges*: The closed-source nature of the language model poses obstacles to achieving reproducibility. Limited access hinders the ability to reproduce and verify results. Although efforts are being made to open-source these architectures [64], the reliance on substantial hardware resources adds another layer of complexity to the reproducibility process. (4) *Adaptability challenges*: Despite the impressive performance of the model for language modeling and its ability to follow instructions, its performance varies

[6]https://github.com/SwappingRepair/SRFN

greatly depending on the specific task at hand [69]. Adapting it to different tasks, such as file structure cleaning, remains a significant challenge, as it tends to exhibit hallucination when confronted with tasks beyond its specific training.

5.4 Runtime Analysis

Tasheeh achieved an average classification time of 6.47 ± 9.24 ms per file with the global distance score threshold. The transformation times averaged at 4.45 ± 6.32 ms per file on a computer with a 4-core Intel Core i7 2.3G CPU and 16GB of RAM.

Figure 8 shows the runtime of Tasheeh on the files in our datasets. Additionally, to test on larger files, we included six large files by extending existing files with duplicate rows. Although the overall runtime of our approach scales quadratically, we observed a high variance in the results due to the quite different pattern complexity of individual files. All-around, in the complete error detection and correction pipeline, the row pattern generation process of Suragh dominates the processing time.

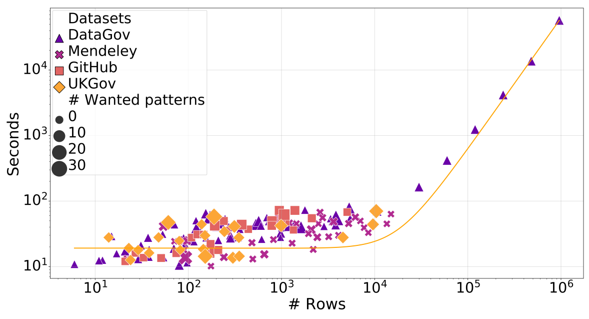


Figure 8: Tasheeh classification and transformation efficiency, together with Suragh (quadratic interpolation). The size of the marks indicates the number of wanted patterns within the file, i.e., indirectly its degree of inconsistency.

5.5 Usability Case Study

To demonstrate the usability of Tasheeh, we conducted a user study, to measure 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 mentioned in Section 5.1. These files exhibit an average number of rows of 904 ± 842, with an average number of ill-formed rows of 64 ± 47. Before the study, we provided them with a clear explanation of the task, i.e., row structure cleaning. They were free to use any tool or programming environment they preferred. Each participant was assigned the same set of files to work on. We measure the file-wise time taken for completion and the accuracy of the cleaning rows with the same measure of Section 5.

Figure 9 shows the results for both manual cleaning and Tasheeh for the sample files (each expert is represented by the same color and marker type). Manually cleaning required a significant amount of time, averaging 67 ± 18 minutes across all experts. Additionally, the accuracy achieved is not always perfect, averaging 83 ± 17 % across all experts: sometimes experts simply removed the inconsistencies they did not understand, e.g., misplaced delimiter. Also, in some cases, certain unwanted rows, such as aggregation rows or expanded group headers were erroneously treated as wanted, resulting in a significant negative

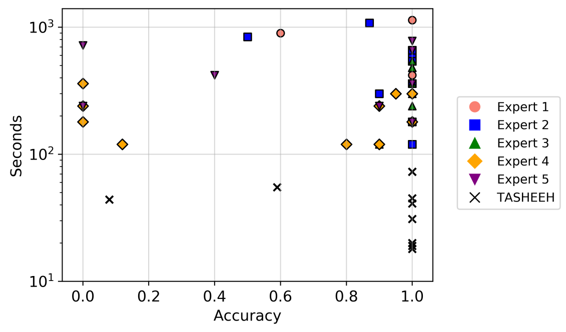


Figure 9: Comparison of manual cleaning with Tasheeh

impact on the overall score. In contrast, when utilizing the Tasheeh system, the accuracy is significantly higher, with 8 out of 10 files achieving a perfect cleaning result, averaging 87%. Moreover, the time required for cleaning using Tasheeh is remarkably low, averaging 6.80 ± 0.34 minutes (across three experimental runs). Even if we consider that the user would manually clean the two files that were not perfectly cleaned by Tasheeh, the time required to achieve a completely flawless result would be 9 ± 3 minutes, which was the average time the experts took for these two files during manual cleanup. To summarize, the use of Tasheeh not only significantly reduces the overall cleaning time to 15.80 ± 3.34 minutes but also delivers improved accuracy compared to a fully manual approach.

6 CONCLUSION

Our work introduces Tasheeh, a data preparation system designed to identify and clean ill-formed data rows in raw CSV files. It utilizes the pattern language introduced in our previous work Suragh [21], 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 repairs the structural inconsistencies in the ill-formed wanted rows using a pattern transformation algebra.

To evaluate the effectiveness of Tasheeh, we extended the annotated data provided in our previous work Suragh and created a ground truth of 200 351 rows across 148 files, each with at least one loading problem. Moreover, we created a distinct ground truth of manually cleaned ill-formed wanted rows. Our results show that Tasheeh achieves an average precision of 95% and an average recall of 96% in identifying wanted rows across all files. In addition, 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.

As Tasheeh is extensible, it allows for the addition of new transformation operators as needed, ensuring that the system can be adapted to handle new use cases without requiring a complete overhaul of the underlying architecture.

In addition to its primary goal of reducing human effort during raw data preparation, Tasheeh functionalities offer promising future directions, e.g., data annotation, data augmentation, preparation suggestion, and preparation estimation.

ACKNOWLEDGMENTS

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

REFERENCES

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

[2] 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? *PVLDB* 9, 12 (2016), 993–1004.

[3] Ziawasch Abedjan, John Morcos, Ihab F Ilyas, Mourad Ouzzani, Paolo Papotti, and Michael Stonebraker. 2016. DataXformer: A robust transformation discovery system. In *Proceedings of the International Conference on Data Engineering (ICDE)*. IEEE, 1134–1145.

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

[5] Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers, and David J Lipman. 1990. Basic local alignment search tool. *Journal of molecular biology* 215, 3 (1990), 403–410.

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

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

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

[9] Xu Chu, Ihab F Ilyas, and Paolo Papotti. 2013. Discovering denial constraints. *PVLDB* 6, 13 (2013), 1498–1509.

[10] 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)*. 1247–1261.

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

[12] Edsger W. Dijkstra. 1959. A note on two problems in connexion with graphs. *Numer. Math.* 1 (1959), 269–271.

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

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

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

[16] Anna Fariha, Ashish Tiwari, Alexandra Meliou, Arjun Radhakrishna, and Sumit Gulwani. 2021. Coco: Interactive exploration of conformance constraints for data understanding and data cleaning. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. 2706–2710.

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

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

[19] Inc. Google. 2022. *OpenRefine*. www.openrefine.org (last accessed August 30th, 2022).

[20] Mazhar Hameed and Felix Naumann. 2020. Data Preparation: A Survey of Commercial Tools. *SIGMOD Record* 49, 3 (2020), 18–29.

[21] Mazhar Hameed, Gerardo Vitagliano, Lan Jiang, and Felix Naumann. 2022. SURAGH: Syntactic Pattern Matching to Identify Ill-Formed Records.. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*. 143–154.

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

[23] Jeffrey Heer, Joseph M Hellerstein, and Sean Kandel. 2015. Predictive Interaction for Data Transformation.. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*. Citeseer.

[24] 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)*. 829–846.

[25] Joseph M Hellerstein. 2008. Quantitative data cleaning for large databases. *United Nations Economic Commission for Europe (UNECE)* 25 (2008), 1–42.

[26] Joseph M Hellerstein, Jeffrey Heer, and Sean Kandel. 2018. Self-Service Data Preparation: Research to Practice. *IEEE Data Engineering Bulletin* 41, 2 (2018), 23–34.

[27] Severin Holzer and Kurt Stockinger. 2022. Detecting errors in databases with bidirectional recurrent neural networks. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*.

[28] 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)*. 1377–1392.

[29] IETF 2005. RFC 4180. https://tools.ietf.org/html/rfc4180. (last accessed February 7th, 2023).

[30] Trifacta Inc. 2022. *Trifacta Data Engineering Cloud*. www.trifacta.com (last accessed August 30th, 2022).

[31] Lan Jiang, Gerardo Vitagliano, and Felix Naumann. 2021. Structure Detection in Verbose CSV Files. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*. 193–204.

[32] Zhongjun Jin, Michael R Anderson, Michael Cafarella, and HV Jagadish. 2017. Foofah: Transforming data by example. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. 683–698.

[33] Zhongjun Jin, Michael Cafarella, HV Jagadish, Sean Kandel, Michael Minar, and Joseph M Hellerstein. 2019. CLX: Towards verifiable PBE data transformation. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*. 265–276.

[34] Zhongjun Jin, Yeye He, and Surajit Chauduri. 2020. Auto-transform: learning-to-transform by patterns. *PVLDB* 13, 12 (2020), 2368–2381.

[35] Elvis Koci, Maik Thiele, Wolfgang Lehner, and Oscar Romero. 2018. Table recognition in spreadsheets via a graph representation. In *2018 13th IAPR International Workshop on Document Analysis Systems (DAS)*. IEEE, 139–144.

[36] Elvis Koci, Maik Thiele, Oscar Romero, and Wolfgang Lehner. 2019. Cell classification for layout recognition in spreadsheets. In *Knowledge Discovery, Knowledge Engineering and Knowledge Management: 8th International Joint Conference, IC3K 2016, Porto, Portugal, November 9–11, 2016, Revised Selected Papers 8*. Springer, 78–100.

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

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

[39] Mohammad Mahdavi and Ziawasch Abedjan. 2020. Baran: Effective error correction via a unified context representation and transfer learning. *PVLDB* 13, 12 (2020), 1948–1961.

[40] 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)*. 865–882.

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

[42] Avanika Narayan, Ines Chami, Laurel J. Orr, and Christopher Ré. 2022. Can Foundation Models Wrangle Your Data? *PVLDB* 16, 4 (2022), 738–746.

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

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

[45] Dan Olteanu. 2020. The Relational Data Borg is Learning. *PVLDB* 13, 12 (2020), 3502–3515.

[46] Jinfeng Peng, Derong Shen, Nan Tang, Tieying Liu, Yue Kou, Tiezheng Nie, Hang Cui, and Ge Yu. 2022. Self-supervised and Interpretable Data Cleaning with Sequence Generative Adversarial Networks. *PVLDB* 16, 3 (2022), 433–446.

[47] David Pinto, Andrew McCallum, Xing Wei, and W Bruce Croft. 2003. Table extraction using conditional random fields. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*. 235–242.

[48] Gil Press. 2016. Cleaning Data: Most Time-Consuming, Least Enjoyable Data Science Task. *Forbes* (March 2016).

[49] Abdulhakim Qahtan, Nan Tang, Mourad Ouzzani, Yang Cao, and Michael Stonebraker. 2020. Pattern functional dependencies for data cleaning. *PVLDB* 13, 5 (2020), 684–697.

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

[51] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving Language Understanding by Generative Pre-Training.

[52] 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. 2022. *pandas-dev/pandas: Pandas 1.4.3*. https://doi.org/10.5281/zenodo.6702671

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

[54] Yoones A Sekhavat, Francesco Di Paolo, Denilson Barbosa, and Paolo Merialdo. 2014. Knowledge base augmentation using tabular data. In *Proceedings of the*

Workshop on Linked Data on the Web (LDOW).

[55] Vraj Shah and Arun Kumar. 2019. 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*. 1–4.

[56] Rishabh Singh. 2016. Blinkfill: Semi-supervised programming by example for syntactic string transformations. *PVLDB* 9, 10 (2016), 816–827.

[57] Rishabh Singh and Sumit Gulwani. 2012. Learning semantic string transformations from examples. *PVLDB* 5, 8 (2012), 740—-751.

[58] Shaoxu Song, Aoqian Zhang, Lei Chen, and Jianmin Wang. 2015. Enriching data imputation with extensive similarity neighbors. *PVLDB* 8, 11 (2015), 1286–1297.

[59] Elias Stehle and Hans-Arno Jacobsen. 2020. ParPaRaw: Massively Parallel Parsing of Delimiter-Separated Raw Data. *PVLDB* 13, 5 (2020), 616–628.

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

[61] LLC Tableau Software. 2022. *Tableau.* www.tableau.com (last accessed August 30th, 2022).

[62] Nan Tang, Ju Fan, Fangyi Li, Jianhong Tu, Xiaoyong Du, Guoliang Li, Sam Madden, and Mourad Ouzzani. 2021. RPT: Relational Pre-trained Transformer Is Almost All You Need towards Democratizing Data Preparation. *PVLDB* 14, 8 (2021), 1254–1261.

[63] Ignacio G Terrizzano, Peter M Schwarz, Mary Roth, and John E Colino. 2015. Data Wrangling: The Challenging Journey from the Wild to the Lake. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*.

[64] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. *CoRR* abs/2302.13971 (2023).

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

[66] Gerardo Vitagliano, Mazhar Hameed, Lan Jiang, Lucas Reisener, Eugene Wu, and Felix Naumann. 2023. Pollock: A Data Loading Benchmark. *PVLDB* 16, 8 (2023), 1870–1882.

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

[68] 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)*. 811–828.

[69] Chaoning Zhang, Chenshuang Zhang, Chenghao Li, Yu Qiao, Sheng Zheng, Sumit Kumar Dam, Mengchun Zhang, Jung Uk Kim, Seong Tae Kim, Jinwoo Choi, et al. 2023. One Small Step for Generative AI, One Giant Leap for AGI: A Complete Survey on ChatGPT in AIGC Era. *CoRR* abs/2304.06488 (2023).