UNIVERSITY OF POTSDAM
HASSO PLATTNER INSTITUTE
INFORMATION SYSTEMS GROUP

Structural Preparation of Raw Data Files

Dissertation
zur Erlangung des akademischen Grades
“Doktor der Ingenieurwissenschaften”

(Dr.-Ing.)

in der Wissenschaftsdisziplin
“Informationssysteme”

eingereicht an der
Digital Engineering Fakultät
der Universität Potsdam

von
Mazhar Hameed

Potsdam, 18. December 2023

Unless otherwise indicated, this work is licensed under a Creative Commons License Attribution – NonCommercial – ShareAlike 4.0 International.
This does not apply to quoted content and works based on other permissions.
To view a copy of this licence visit:
https://creativecommons.org/licenses/by-nc-sa/4.0/legalcode.en

Reviewers

Prof. Dr. Felix Naumann
Hasso Plattner Institute, University of Potsdam

Prof. Dr. Holger Schwarz
Institute for Parallel and Distributed Systems, University of Stuttgart

Prof. Dr. Giovanni Simonini
Department of Engineering "Enzo Ferrari", University of Modena and Reggio Emilia

Published online on the
Publication Server of the University of Potsdam:
https://doi.org/10.25932/publishup-65567
https://nbn-resolving.org/urn:nbn:de:kobv:517-opus4-655678

Abstract

Data preparation stands as a cornerstone in the landscape of data science workflows, commanding a significant portion—approximately 80%—of a data scientist's time. The extensive time consumption in data preparation is primarily attributed to the intricate challenge faced by data scientists in devising tailored solutions for downstream tasks. This complexity is further magnified by the inadequate availability of metadata, the often ad-hoc nature of preparation tasks, and the necessity for data scientists to grapple with a diverse range of sophisticated tools, each presenting its unique intricacies and demands for proficiency.

Previous research in data management has traditionally concentrated on preparing the content within columns and rows of a relational table, addressing tasks, such as string disambiguation, date standardization, or numeric value normalization, commonly referred to as data cleaning. This focus assumes a perfectly structured input table. Consequently, the mentioned data cleaning tasks can be effectively applied only after the table has been successfully loaded into the respective data cleaning environment, typically in the later stages of the data processing pipeline.

While current data cleaning tools are well-suited for relational tables, extensive data repositories frequently contain data stored in plain text files, such as CSV files, due to their adaptable standard. Consequently, these files often exhibit tables with a flexible layout of rows and columns, lacking a relational structure. This flexibility often results in data being distributed across cells in arbitrary positions, typically guided by user-specified formatting guidelines.

Effectively extracting and leveraging these tables in subsequent processing stages necessitates accurate parsing. This thesis emphasizes what we define as the "structure" of a data file—the fundamental characters within a file essential for parsing and comprehending its content. Concentrating on the initial stages of the data preprocessing pipeline, this thesis addresses two crucial aspects: *comprehending* the structural layout of a table within a raw data file and automatically *identifying* and *rectifying* any structural issues that might hinder its parsing. Although these issues may not directly impact the table's content, they pose significant challenges in parsing the table within the file.

Our initial contribution comprises an extensive survey of commercially available data preparation tools. This survey thoroughly examines their distinct features, the lacking features, and the necessity for preliminary data processing despite these tools. The primary goal is to elucidate the current state-of-the-art in data preparation systems while identifying areas for enhancement. Furthermore, the survey explores the encountered challenges in data preprocessing, emphasizing opportunities for future research and improvement.

Next, we propose a novel data preparation pipeline designed for detecting and correcting structural errors. The aim of this pipeline is to assist users at the initial preprocessing stage by ensuring the correct loading of their data into their preferred systems. Our approach begins by introducing SURAGH, an unsupervised system that utilizes a pattern-based method to identify dominant patterns within a file, independent of external information, such as data types, row structures, or schemata. By identifying deviations from the dominant pattern, it detects *ill-formed* rows. Subsequently, our structure correction system, TASHEEH, gathers the identified ill-formed rows along with dominant patterns and employs a novel pattern transformation algebra to automatically rectify errors. Our pipeline serves as an end-to-end solution, transforming a structurally broken CSV file into a well-formatted one, usually suitable for seamless loading.

Finally, we introduce MORPHER, a user-friendly GUI integrating the functionalities of both SURAGH and TASHEEH. This interface empowers users to access the pipeline's features through visual elements. Our extensive experiments demonstrate the effectiveness of our data preparation systems, requiring no user involvement. Both SURAGH and TASHEEH outperform existing state-of-the-art methods significantly in both precision and recall.

Zusammenfassung

Die Datenaufbereitung ist ein wesentlicher Bestandteil von Data-Science-Workflows und nimmt einen beträchtlichen Teil - etwa 80% - der Zeit eines Datenwissenschaftlers in Anspruch. Der hohe Zeitaufwand für die Datenaufbereitung ist in erster Linie auf die komplizierte Herausforderung zurückzuführen, der sich Datenwissenschaftler bei der Entwicklung maßgeschneiderter Lösungen für nachgelagerte Aufgaben gegenübersehen. Diese Komplexität wird noch verstärkt durch die unzureichende Verfügbarkeit von Metadaten, den oft ad-hoc-Charakter der Aufbereitungsaufgaben und die Notwendigkeit für Datenwissenschaftler, sich mit einer Vielzahl von hochentwickelten Tools auseinanderzusetzen, von denen jedes seine eigenen Schwierigkeiten und Anforderungen an dessen Beherrschung aufweist.

Bisherige Forschung im Bereich der Datenverwaltung konzentriert sich traditionell auf die Aufbereitung der Inhalte innerhalb der Spalten und Zeilen einer relationalen Tabelle und befasst sich mit Aufgaben wie der Disambiguierung von Zeichenketten, der Standardisierung von Datumsangaben oder der Normalisierung numerischer Werte, die gemeinhin unter dem Begriff der Datenbereinigung zusammengefasst werden. Dieser Forschungsschwerpunkt geht von einer perfekt strukturierten Eingabetabelle aus. Folglich können die genannten Datenbereinigungsaufgaben erst dann effektiv durchgeführt werden, wenn die Tabelle erfolgreich in die entsprechende Datenbereinigungsumgebung geladen wurde, was in der Regel in den späteren Phasen der Datenverarbeitungspipeline geschieht.

Während aktuelle Datenbereinigungstools gut für relationale Tabellen geeignet sind, enthalten große Datenrepositories aufgrund ihres flexiblen Standards häufig Daten, die in reinen Textdateien, wie z. B. CSV-Dateien, gespeichert sind. Folglich weisen diese Dateien oft Tabellen mit einem flexiblen Layout von Zeilen und Spalten auf, denen eine relationale Struktur fehlt. Diese Flexibilität führt häufig dazu, dass die Daten beliebig über die einzelnen Zellen der Tabelle verteilt sind, was in der Regel durch benutzerdefinierte Formatierungsrichtlinien gesteuert wird.

Um diese Tabellen effektiv zu extrahieren und in den nachgelagerten Verarbeitungsschritten nutzen zu können, ist ein präzises Parsen erforderlich. In dieser Arbeit wird der Schwerpunkt auf das gelegt, was wir als "Struktur" einer Datendatei definieren - die grundlegenden Zeichen innerhalb einer Datei, die für das Parsen und Verstehen ihres Inhalts wesentlich sind. Die vorliegende Arbeit konzentriert sich auf die ersten Stufen der Datenvorverarbeitung und behandelt zwei entscheidende Aspekte: Das *Verstehen* des strukturellen Layouts einer Tabelle in einer Rohdatendatei und das automatische *Erkennen* und *Korrigieren* von strukturellen Problemen, die das Parsen der Datei erschweren könnten. Auch wenn sich diese Probleme nicht direkt auf den Inhalt der Tabelle auswirken, stellen sie eine große Herausforderung beim Parsen der Tabelle in der Datei dar.

Unser erster Beitrag besteht aus einem umfassenden Überblick über kommerziell verfügbare Datenaufbereitungstools. In dieser Übersicht werden ihre besonderen Merkmale, die fehlenden Merkmale und die Notwendigkeit einer vorläufigen Datenverarbeitung trotz dieser Werkzeuge eingehend untersucht. Das primäre Ziel ist es, den aktuellen Stand der Technik bei den Datenaufbereitungssystemen zu ermitteln und gleichzeitig Bereiche zu identifizieren, die verbessert werden können. Darüber hinaus werden die bei der Datenvorverarbeitung aufgetretenen Herausforderungen untersucht und Möglichkeiten für künftige Forschung und Weiterentwicklungen aufgezeigt.

Als Nächstes schlagen wir eine neuartige Datenaufbereitungspipeline zur Erkennung und Korrektur von strukturellen Fehlern vor. Ziel dieser Pipeline ist es, die Nutzer in der anfänglichen Vorverarbeitungsphase zu unterstützen, indem das korrekte Laden ihrer Daten in ihre bevorzugten Systeme sichergestellt wird. Unser Ansatz beginnt mit der Einführung von SURAGH, einem unüberwachten System, das eine musterbasierte Methode verwendet, um dominante Muster innerhalb einer Datei zu identifizieren, unabhängig von externen Informationen wie Datentypen, Zeilenstrukturen oder Schemata. Durch die Identifizierung von Abweichungen vom vorherrschenden Muster werden *fehlerhafte* Zeilen erkannt. Anschließend sammelt unser Strukturkorrektursystem, TASHEEH, die identifizierten fehlerhaften Zeilen zusammen mit den dominanten Mustern und verwendet eine neuartige Mustertransformationsalgebra, um Fehler automatisch zu korrigieren. Unsere Pipeline dient als End-to-End-Lösung, die eine strukturell fehlerhafte CSV-Datei in eine gut formatierte Datei umwandelt, die in der Regel für ein nahtloses Laden geeignet ist.

Schließlich stellen wir MORPHER vor, eine benutzerfreundliche GUI, die die Funktionen von SURAGH und TASHEEH integriert. Mit Hilfe von visuellen Elementen ermöglicht diese Schnittstelle den Benutzern den Zugriff auf die Funktionen der Pipeline. Unsere umfangreichen Experimente zeigen die Effektivität unserer Datenaufbereitungssysteme, die kein Eingreifen des Benutzers erfordern. Sowohl SURAGH als auch TASHEEH übertreffen bestehende State-of-the-Art-Methoden in den beiden Metriken Precision und Recall deutlich.

Acknowledgements

I would like to commence with a quote from Caliph Ali, a close companion of Prophet Muhammad (PBUH), who said:

"One who taught me a single word made me eternally indebted to them."

This quote exemplifies the respect students should offer their teachers.

Through these words, I wish to express my profound gratitude to my advisor, Professor Felix Naumann. I will forever be in your debt for the invaluable knowledge you have imparted to me throughout the years.

Next, I want to express my love and gratitude to my wife, Hiba. Your unwavering support, love, and encouragement have been crucial in helping me reach this milestone. I could not have achieved this without you.

Furthermore, I am deeply grateful to my loving parents and my loving sister for their countless prayers and unwavering moral support throughout this journey. My heartfelt thanks also go to my parents-in-law and sisters-in-law for their encouragement and support.

Finally, I would like to acknowledge my wonderful colleagues and friends at the Information Systems group, especially my Data Preparation teammates. You all made my time incredibly enjoyable, to the extent that I can confidently say I have never had such an exceptional team experience before.

We can only see a short distance ahead, but we can see plenty there that needs to be done.

— Alan M. Turing

Contents

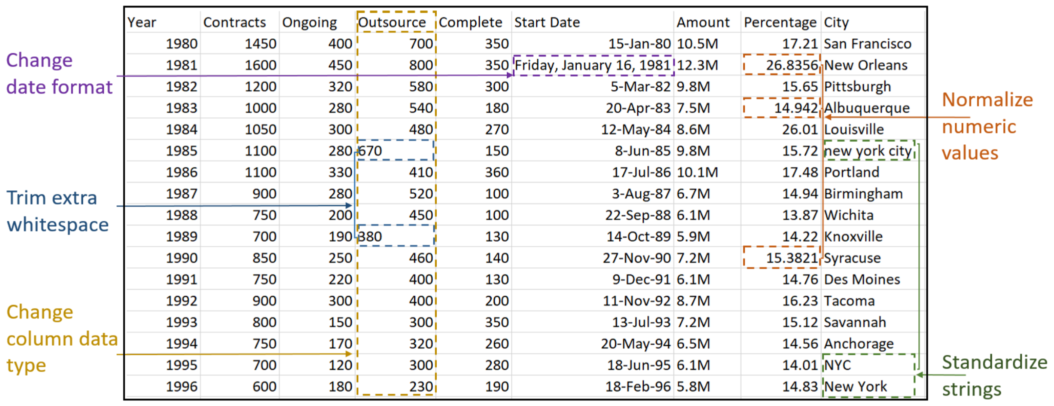
Chapter 1

STRUCTURAL DATA PREPARATION

We live in an age of technology where data are the new oil [8, 24, 73], and much like its predecessor, data must be extracted and refined before they have any practical use. However, unlike its predecessor, the amount of data generated is enormous and growing exponentially, spurred by surveillance devices generating sensor data, social media platforms, government data portals, medical research projects, etc. It is estimated by the international data corporation (IDC) that a substantial growth in the number of connected IoT devices to reach 55.7 billion by 2025, leading to the generation of nearly 80 zettabytes (ZB) of data [44]. This increase in data generation is also reflected in the IDC projections, which expect the volume of digital data produced, acquired, and copied across the world to reach 175 ZB by 2025, up from 59 ZB in 2020 [28].

Data obtained from various devices and platforms often exist in a raw format, lacking a standardized structure [3, 49, 103]. Consequently, parsing such data introduces numerous structural inconsistencies, including invalid characters resulting from incorrect parsing, column shifting due to incorrect escaping, inconsistent formatting, and more. These inconsistencies pose challenges during data ingestion, requiring significant efforts from data scientists and machine learning engineers, who spend a substantial amount of time on the laborious task of data preparation [64, 71, 82]. Studies have indicated that data scientists typically spend a significant portion of their time, ranging from 50% to 80%, on data preparation tasks, which include activities such as gathering, loading, cleaning, and transforming data [41, 77, 98]. Unfortunately, data preparation is often regarded as the least enjoyable aspect of their work. In fact, a survey conducted by Forbes revealed that 76% of the 80 interviewed data scientists considered data preparation to be the least enjoyable part of their job [77]. These studies highlight the considerable time and effort invested in this crucial but often labor-intensive process.

Data scientists dedicate considerable time to data preparation, primarily driven by the need to prevent the “garbage in, garbage out” (GIGO) situation. GIGO refers to the scenario where using low-quality input data leads to inaccurate or unreliable output insights. One common practice in data preparation is conducting trivial but critical transformations on raw data, as illustrated in Figure 2. By applying this pipeline to the unprepared data depicted in Figure 1, the resulting prepared data are shown in Figure 3. However, building ad-hoc scripts for these transformations is inefficient, as the scripts are usually implemented based on a user's cursory observation of data and might not be robust to erroneous content hidden within. In addition, these scripts also assume that the input file is free of preprocessing errors and can be seamlessly loaded into the downstream application or development platform without encountering any ingestion issues, as depicted in Figure 1, where the cell boundaries are clearly defined and no additional metadata are present. Furthermore, such scripts assume the execution environment at hand of the respective developer, making them challenging to repeat or adapt to other environments. Consequently, data scientists and analysts often spend additional time tailoring scripts to the data in particular environments.


Change date format
Trim extra whitespace
Change column data type
Normalize numeric values
Standardize strings

Year	Contracts	Ongoing	Outsource	Complete	Start Date	Amount	Percentage	City
1980	1450	400	700	350	15-Jan-80	10.5M	17.21	San Francisco
1981	1600	450	800	350	Friday, January 16, 1981	12.3M	26.8356	New Orleans
1982	1200	320	580	300	5-Mar-82	9.8M	15.65	Pittsburgh
1983	1000	280	540	180	20-Apr-83	7.5M	14.942	Albuquerque
1984	1050	300	480	270	12-May-84	8.6M	26.01	Louisville
1985	1100	280	670	150	8-Jun-85	9.8M	15.72	new york city
1986	1100	330	410	360	17-Jul-86	10.1M	17.48	Portland
1987	900	280	520	100	3-Aug-87	6.7M	14.94	Birmingham
1988	750	200	450	100	22-Sep-88	6.1M	13.87	Wichita
1989	700	190	380	130	14-Oct-89	5.9M	14.22	Knoxville
1990	850	250	460	140	27-Nov-90	7.2M	15.3821	Syracuse
1991	750	220	400	130	9-Dec-91	6.1M	14.76	Des Moines
1992	900	300	400	200	11-Nov-92	8.7M	16.23	Tacoma
1993	800	150	300	350	13-Jul-93	7.2M	15.12	Savannah
1994	750	170	320	260	20-May-94	6.5M	14.56	Anchorage
1995	700	120	300	280	18-Jun-95	6.1M	14.01	NYC
1996	600	180	230	190	18-Feb-96	5.8M	14.83	New York

Figure 1: An overview of unprepared data marked with the preparation steps specified in Figure 2.

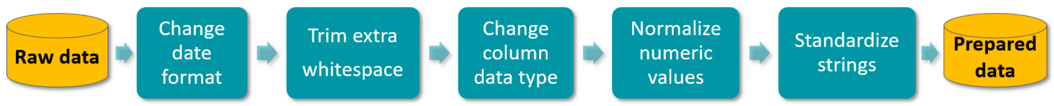
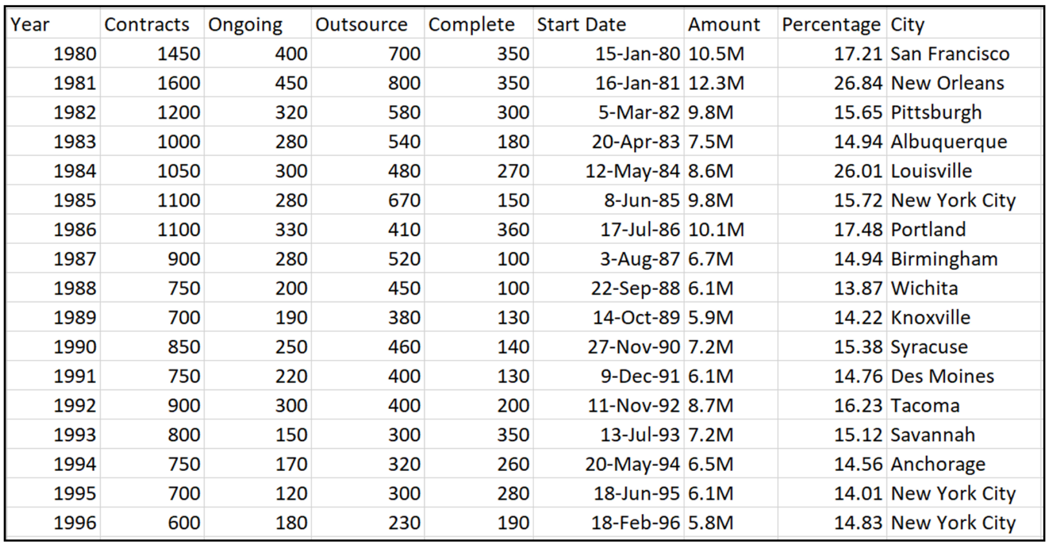


Figure 2: A sample pipeline of multiple transformations describing a common data preparation task.

The traditional data preparation workflow involves two primary roles, business experts and data experts, with the former specifying data preparation requirements and the latter implementing and executing corresponding preparation scripts or pipelines. However, communication overhead often becomes a bottleneck in agile data preparation, leading to more sophisticated data preparation systems, such as Open-Refine [31], Tableau [96], and Trifacta [48]. These systems aim to bridge the communication gap and enhance efficiency by offering interactive interfaces and menu-based transformation options. This



Year	Contracts	Ongoing	Outsource	Complete	Start Date	Amount	Percentage	City
1980	1450	400	700	350	15-Jan-80	10.5M	17.21	San Francisco
1981	1600	450	800	350	16-Jan-81	12.3M	26.84	New Orleans
1982	1200	320	580	300	5-Mar-82	9.8M	15.65	Pittsburgh
1983	1000	280	540	180	20-Apr-83	7.5M	14.94	Albuquerque
1984	1050	300	480	270	12-May-84	8.6M	26.01	Louisville
1985	1100	280	670	150	8-Jun-85	9.8M	15.72	New York City
1986	1100	330	410	360	17-Jul-86	10.1M	17.48	Portland
1987	900	280	520	100	3-Aug-87	6.7M	14.94	Birmingham
1988	750	200	450	100	22-Sep-88	6.1M	13.87	Wichita
1989	700	190	380	130	14-Oct-89	5.9M	14.22	Knoxville
1990	850	250	460	140	27-Nov-90	7.2M	15.38	Syracuse
1991	750	220	400	130	9-Dec-91	6.1M	14.76	Des Moines
1992	900	300	400	200	11-Nov-92	8.7M	16.23	Tacoma
1993	800	150	300	350	13-Jul-93	7.2M	15.12	Savannah
1994	750	170	320	260	20-May-94	6.5M	14.56	Anchorage
1995	700	120	300	280	18-Jun-95	6.1M	14.01	New York City
1996	600	180	230	190	18-Feb-96	5.8M	14.83	New York City

Figure 3: Prepared data after applying the data preparation steps specified in Figure 2.

feature empowers even non-expert users to perform self-service data preparation tasks. With the growing demand for high-quality data, these systems have become essential for data science projects. However, their efficiency and effectiveness may vary based on the user's level of expertise, the size of the dataset, and the complexity of the inconsistencies appearing in the data.

1.1 Structural (Syntactic) and Semantic Inconsistencies

In the realm of data files, inconsistencies can be classified into two main categories: *semantic* inconsistencies and *syntactic* inconsistencies. Semantic inconsistencies arise due to an inadequate understanding of the contextual information associated with data values, leading to violations of integrity constraints [47], the presence of outliers [9], missing values [79], duplicates [69], and other issues [3]. These inconsistencies can have a significant impact on the quality of data values within a file, leading to erroneous analyses and unreliable outcomes. Fortunately, several data cleaning approaches have been proposed to detect these inconsistencies [13, 38, 42, 43, 63, 79, 105], as well as for their correction [15, 27, 52, 62, 75, 78, 86]. Moreover, ongoing research aims to uncover the symptoms, causes, and origins of data errors, shedding light on the factors that contribute to these inconsistencies [46].

On the other hand, syntactic or structural inconsistencies emerge as a consequence of devi-

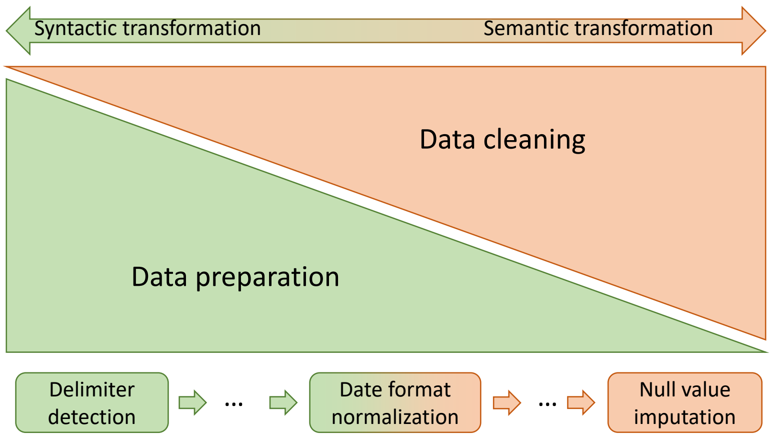
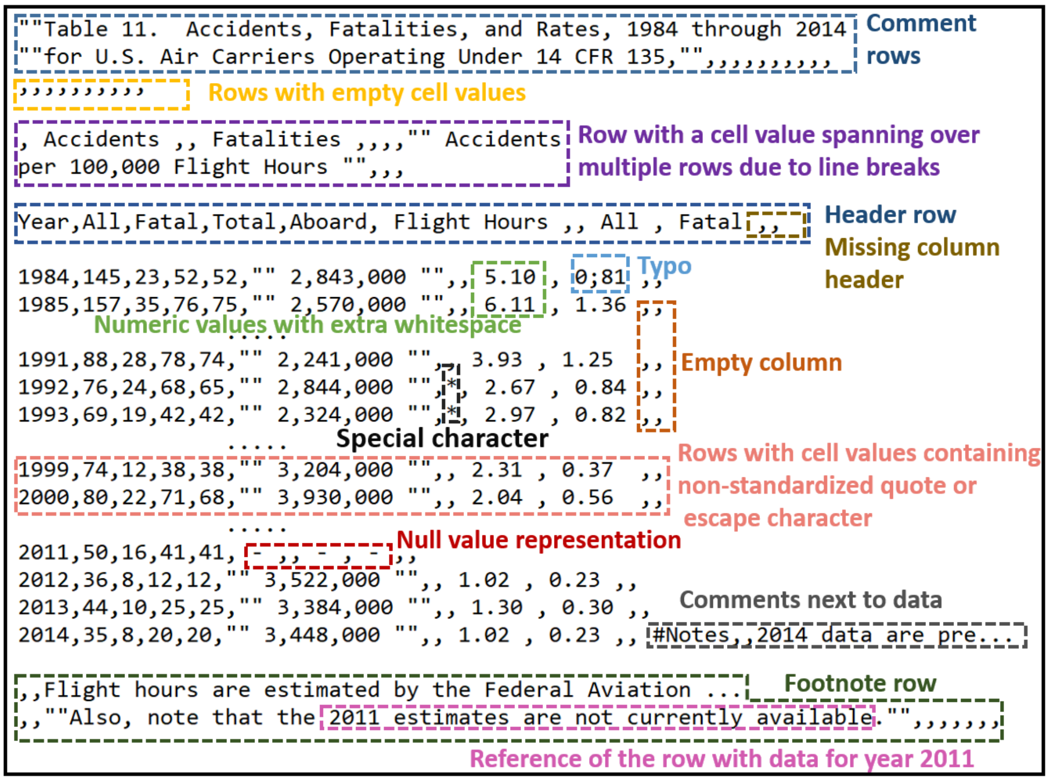


Figure 4: Data preparation vs. data cleaning [32]

ations from prescribed formatting guidelines, established standards, or the defined schema. These structural inconsistencies introduce a multitude of challenges, such as the presence of invalid characters resulting from incorrect parsing, different number of cells across data rows, different variants of quoted fields due to user specifications, etc. While semantic inconsistencies appear at the content level, syntactic inconsistencies occur at the structural level of a file. Addressing these inconsistencies adds an intriguing new dimension to the task of ensuring data quality and facilitating seamless data ingestion for downstream applications.

To effectively handle semantic inconsistencies, a comprehensive understanding of the meaning and context of the data values is essential. Addressing structural inconsistencies involves enforcing predefined structural constraints and adhering to specific formatting guidelines and standards. We define data preparation as the set of preprocessing operations performed at the structural level in the early stages of a data processing pipeline [32]. Throughout the thesis, we provide numerous examples of such transformations. Data cleaning, on the other hand, involves subsequent transformations and corrections at the semantic level (Figure 4).

In the next section, we discuss the challenges of data ingestion with structural inconsistencies, emphasizing their impact on the overall process. Subsequently, we discuss the structural inconsistencies of raw CSV files, along with the associated research challenges related to detecting and correcting these inconsistencies. Finally, we conclude this chapter by outlining the structure of the subsequent chapters and summarizing their respective contributions.



```
""Table 11.  Accidents, Fatalities, and Rates, 1984 through 2014
""for U.S. Air Carriers Operating Under 14 CFR 135,"",,,,,,,,,,
,,,,,,,,,,
, Accidents ,, Fatalities ,,,,"" Accidents
per 100,000 Flight Hours "",,,
Year,All,Fatal,Total,Aboard, Flight Hours ,, All , Fatal,,
1984,145,23,52,52,"" 2,843,000 "",, 5.10 , 0;81 ,,
1985,157,35,76,75,"" 2,570,000 "",, 6.11 , 1.36 ,,
                    .....
1991,88,28,78,74,"" 2,241,000 "",, 3.93 , 1.25 ,,
1992,76,24,68,65,"" 2,844,000 "",*, 2.67 , 0.84 ,,
1993,69,19,42,42,"" 2,324,000 "",*, 2.97 , 0.82 ,,
                    .....
1999,74,12,38,38,"" 3,204,000 "",, 2.31 , 0.37  ,,
2000,80,22,71,68,"" 3,930,000 "",, 2.04 , 0.56  ,,
                    .....
2011,50,16,41,41, - ,, - , - ,,
2012,36,8,12,12,"" 3,522,000 "",, 1.02 , 0.23 ,,
2013,44,10,25,25,"" 3,384,000 "",, 1.30 , 0.30 ,,
2014,35,8,20,20,"" 3,448,000 "",, 1.02 , 0.23 ,, #Notes,,2014 data are pre...

,,Flight hours are estimated by the Federal Aviation ...
,,""Also, note that the 2011 estimates are not currently available."",,,,,,,,
```

Comment rows
Rows with empty cell values
Row with a cell value spanning over multiple rows due to line breaks
Header row
Missing column header
Typo
Numeric values with extra whitespace
Empty column
Special character
Rows with cell values containing non-standardized quote or escape character
Null value representation
Comments next to data
Footnote row
Reference of the row with data for year 2011

Figure 5: A sample of a raw CSV file with ill-formed rows due to structural inconsistencies at both column- and row-levels.

1.2 Data Ingestion with Structural Inconsistencies

For data-driven projects, the first step is typically loading the data. Even to prepare data, they must first be successfully loaded into the platform or tool being used for data preparation. However, existing tools and systems often assume structurally preprocessed files that adhere to specific standards and guidelines, making the data ingestion process more seamless [32]. In practice, data files can be challenging to ingest due to their non-standardized formats and the presence of structural inconsistencies that deviate from established standards [33]. These inconsistencies pose significant challenges during the data ingestion phase and often necessitate significant human intervention to ensure successful ingestion in downstream applications. In this context, two prominent research problems are the detection and correction of rows that exhibit structural inconsistencies, which we refer to as *ill-formed* rows (see Section 3.2 for a formal definition). Such rows do not adhere to the expected structure and can disrupt data-driven tasks, leading to aborted loading processes, incorrect parsing of data, and interference with the training process of machine learning algorithms.

While detecting and cleaning ill-formed rows is a general problem across different types of data files, this thesis specifically focuses on CSV files – a type of plain-text data file used to store semi-structured data. The CSV file format is popular for its simplicity, read/write capabilities, and compatibility with numerous systems. However, this flexibility can also be a drawback, leading to non-standard file formats and a lack of appropriate metadata in real-world datasets. CSV files found on open data portals, for instance, often exhibit significant structural variations, such as multiple tables stacked on top of each other, differing numbers of cells across rows, data and metadata appearing in the same rows, user-specified quote and escape characters, differing numbers of row separators across the file as well as empty visual separators, etc. For example, Figure 5 depicts a typical CSV file, where groups of ill-formed rows with different inconsistencies are highlighted.

1.3 A Taxonomy of Structural Inconsistencies

Structural inconsistencies in raw CSV files can be categorized into different groups. In our proposed taxonomy, we have identified five distinct groups: (i) file level structural inconsistencies, (ii) table level structural inconsistencies, (iii) row level structural inconsistencies, (iv) column level structural inconsistencies, and (v) cell level structural inconsistencies. Figure 6 provides an overview of these groups and includes examples for each category. In the following sections, we delve into each group, providing a detailed explanation of the specific types of structural inconsistencies they encompass, along with concrete examples.

1.3.1 File Level Structural Inconsistencies

Structural inconsistencies in raw CSV files pose significant challenges to data quality and usability. At file level, structural inconsistencies emerge at the earliest stage of data processing, giving rise to issues, such as incompatible file encodings, presence of multiple tables within a single file, ambiguity in table boundaries and headers, etc. These inconsistencies have a profound impact on the subsequent stages of data ingestion and analysis processes, as they pose distinct challenges depending on the downstream application. For instance, raw CSV files, known for their flexible standard, may contain vertically or horizontally stacked tables, making it challenging to accurately identify and extract individual tables by determining precise table boundaries. Moreover, recurring headers in stacked tables and the differentiation between header and data sections when both contain numbers or strings further complicate the interpretation. In the case of horizontally stacked tables,

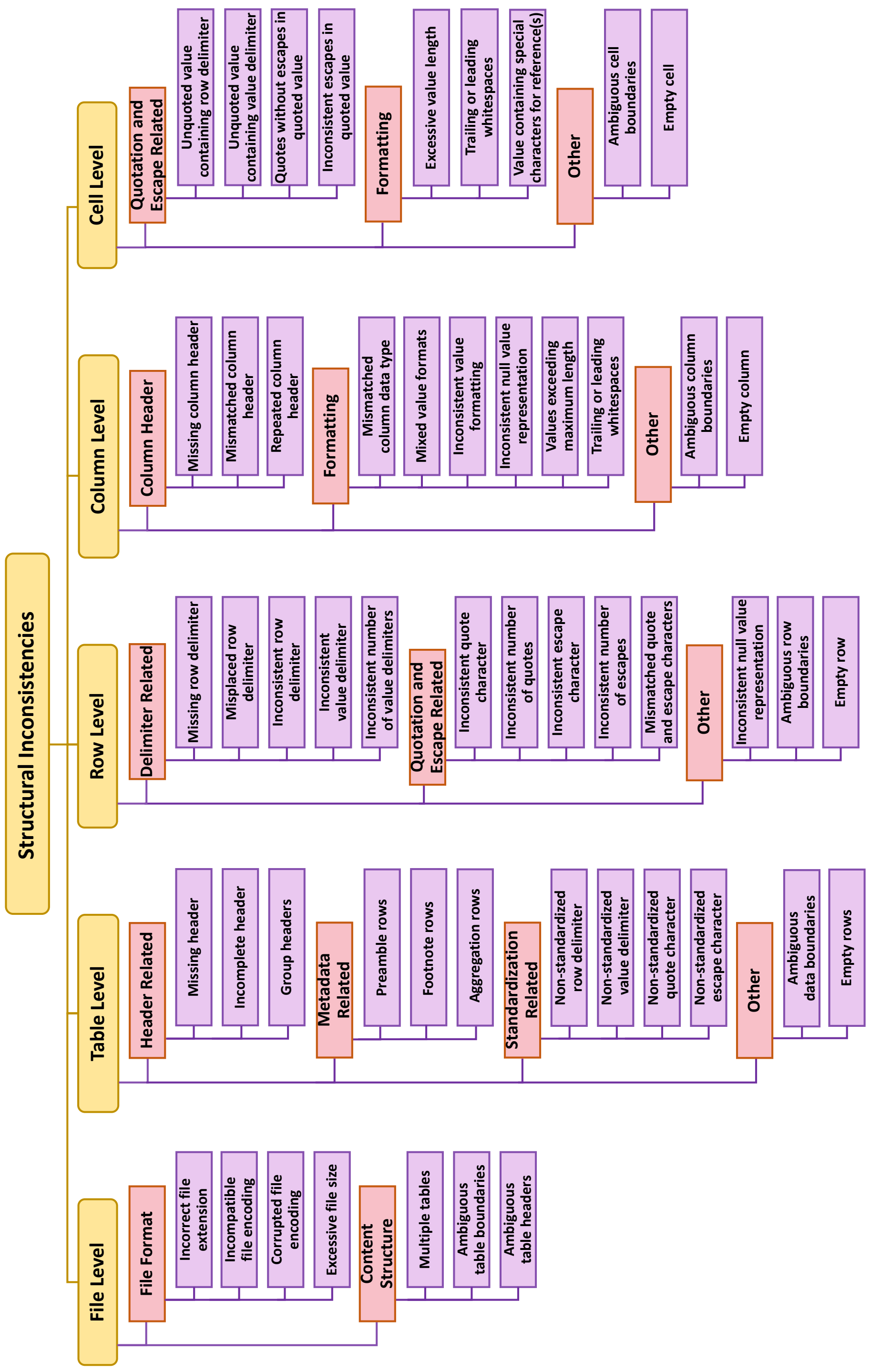

Structural Inconsistencies
File Level
File Format
Incorrect file extension
Incompatible file encoding
Corrupted file encoding
Excessive file size
Content Structure
Multiple tables
Ambiguous table boundaries
Ambiguous table headers
Table Level
Header Related
Missing header
Incomplete header
Group headers
Metadata Related
Preamble rows
Footnote rows
Aggregation rows
Standardization Related
Non-standardized row delimiter
Non-standardized value delimiter
Non-standardized quote character
Non-standardized escape character
Other
Ambiguous data boundaries
Empty rows
Row Level
Delimiter Related
Missing row delimiter
Misplaced row delimiter
Inconsistent row delimiter
Inconsistent value delimiter
Inconsistent number of value delimiters
Quotation and Escape Related
Inconsistent quote character
Inconsistent number of quotes
Inconsistent escape character
Inconsistent number of escapes
Mismatched quote and escape characters
Other
Inconsistent null value representation
Ambiguous row boundaries
Empty row
Column Level
Column Header
Missing column header
Mismatched column header
Repeated column header
Formatting
Mismatched column data type
Mixed value formats
Inconsistent value formatting
Inconsistent null value representation
Values exceeding maximum length
Trailing or leading whitespaces
Other
Ambiguous column boundaries
Empty column
Cell Level
Quotation and Escape Related
Unquoted value containing row delimiter
Unquoted value containing value delimiter
Quotes without escapes in quoted value
Inconsistent escapes in quoted value
Formatting
Excessive value length
Trailing or leading whitespaces
Value containing special characters for reference(s)
Other
Ambiguous cell boundaries
Empty cell

Figure 6: A taxonomy of structural inconsistencies in raw CSV files

the absence of empty visual separators makes identifying the start and end of each table region a significant challenge. Resolving these structural inconsistencies is crucial for enabling effective data interpretation and facilitating seamless data processing workflows. Note, this thesis focuses on detecting structural inconsistencies within single table files. It assumes that the tables have already been extracted [101] and that the file encodings are correct and compatible. Therefore, the scope of this thesis does not include addressing structural inconsistencies at the file level. Instead, it emphasizes dealing with structural inconsistencies at the other four levels (table, row, column, cell).

1.3.2 Table Level Structural Inconsistencies

Raw CSV files consist of tables that can have diverse structures, ranging from arbitrary shapes to those conforming to the RFC 4180 standard for relational tables. Each table in a raw CSV file may display distinct structural inconsistencies. These inconsistencies include ambiguous data boundaries, non-standardized dialect characters (delimiters, quote, escape), multi-layer hierarchical headers, aggregations that establish arithmetic relationships between numeric cells and sets of other numeric cells, etc. Additionally, tables that do not conform to the RFC 4180 standard can exhibit various types of rows with distinct purposes. For instance, comment rows can be found at the beginning of the table, providing additional information about the tables themselves. Footnote rows, located at the bottom of the table, typically contain references to other rows and supplementary metadata. Similarly, in the middle of the table, there may be group headers that provide additional information about the data rows, and users may insert notes for future reference in between rows. Addressing such inconsistencies necessitates a comprehensive approach that involves gaining an understanding of the special characters' role in defining the structure of data within a table, as well as comprehending the actual data content. Moreover, it entails identifying recurring patterns to effectively distinguish between data and non-data elements, as well as comprehending numeric values to distinguish between aggregations and their associated cells, among other essential considerations. In this thesis, we address these structural inconsistencies and provide effective solutions.

1.3.3 Row Level Structural Inconsistencies

As mentioned earlier, CSV files are prone to inconsistencies not only at the table level but also at more granular levels, such as row, column, and even cell levels. At row level, these inconsistencies can appear in various forms, such as ambiguous row boundaries caused by missing new-line separators or cell values containing new-line separators without appropriate quote or escape characters. Another common occurrence is the use of different characters as cell separators within the same table, often resulting from data integration from multiple sources or individuals contributing to the file's creation. Moreover, rows in CSV files often exhibit inconsistent delimiters, deviating from the expected schema. This inconsistency hinders the understanding and definition of the table's structure, resulting in the emergence of multiple columns without headers and containing null values across different rows. Additionally, inconsistent representation of empty values adds to the complexity of data interpretation. Resolving these inconsistencies requires a holistic approach that involves grasping both table-level solutions and patterns specific to row-level inconsistencies. This entails identifying common row patterns and comprehending the structural characters that define the columns within each row. Like the inconsistencies at the table level, this thesis addresses and offers solutions for row level inconsistencies.

1.3.4 Column level structural inconsistencies

Column level structural inconsistencies appear within individual table columns. Unlike standard CSV files, which typically contain a single relational table with columns exhibiting consistent structures, raw CSV files can have columns with varying arrangements of cells serving different purposes. This can result in the absence of optional headers, the presence of diverse value formats within the same column, and inconsistencies between the declared column data type and the actual values present. Resolving these column-level inconsistencies necessitates the application of comprehensive approaches, including techniques for value structure transformation and format normalization. Like other structural inconsistencies, such column-specific challenges are addressed as part of the structure standardization process.

1.3.5 Cell Level Structural Inconsistencies

Due to the lack of a standardized row-wise or column-wise structure in raw CSV files, cell-level structural inconsistencies can arise, demanding careful consideration. These inconsistencies encompass various aspects, including missing or misplaced cells, irregular formatting, values without proper quoting containing value or row delimiters, and inconsistent value representations. Resolving these cell-level inconsistencies requires thorough examination and analysis of individual cells within the file. Techniques, such as data validation and cleansing, can be employed to address these issues and ensure the integrity and consistency of the data at a granular level. Similar to column-level inconsistencies, cell-level challenges are inherently addressed as part of the structure standardization process.

In the preceding discussion, we have outlined a taxonomy comprising five groups of structural inconsistencies commonly found in raw CSV files. It is important to acknowledge that the taxonomy is not exhaustive and represents a partial classification, and there may be additional categories of inconsistencies that can be included. The taxonomy can be expanded both horizontally, by incorporating more diverse categories, and vertically, by encompassing a wider range of specific inconsistencies within each category. Detecting and repairing structural inconsistencies in raw CSV files presents a valuable yet intricate challenge, given the unique characteristics and complexity of file structures.

1.4 Thesis Structure and Contributions

In the previous sections, we introduced the problem of detecting and correcting structural inconsistencies in raw CSV files and discussed the associated challenges. In the following four chapters, we present our main contributions, which emphasize the need for data preparation and offer comprehensive solutions for detecting and correcting a wide range of structural inconsistencies in raw CSV files. These contributions are summarized as follows.

* DATA PREPARATION FROM AN INDUSTRY PERSPECTIVE: A SURVEY

  In Chapter 2, we present a survey that provides an extensive analysis of commercial data preparation tools, and is based on our publication [32]. The survey thoroughly examines prominent data preparation tools, the distinctive features offered by these tools, the need for preliminary data processing even with these tools, and the features that are still lacking. The survey aims to shed light on the current state-of-the-art data preparation systems and identify areas for improvement. To achieve this, we initially compiled over 100 tools from the web, offering data quality features. We narrowed down the selection to 42 tools with data preparation offerings for our preliminary study. To ensure a systematic approach, we organized various data preparation tasks into six broad categories and identified 40 common data preparation steps that fell within these categories. The survey also includes the evaluation of the final selected tools and the creation of a comprehensive feature matrix, showcasing the capabilities of each tool for specific data preparation steps. Additionally, the survey delved into the challenges associated with data preprocessing that the tools encountered, highlighting areas for improvement and presenting opportunities for future research. The survey was conducted by Hameed, while Naumann contributed valuable discussions.

* SURAGH: A STRUCTURAL ERROR DETECTION SYSTEM

  In Chapter 3, we present our system SURAGH, which detects structural errors in CSV files, and is based on our publication [33]. During our survey, we encountered data preprocessing challenges while evaluating the state-of-the-art data preparation tools, particularly when it comes to ingesting data into these systems. The major obstacle we identified was the difficulty in effectively handling the inconsistent structure of rows within files. These inconsistencies occurred in various forms, including variations in the number of attributes across rows, non-standardized formatting of cell values, and the presence of additional information such as comments at the top or concluding remarks as footnotes. To detect rows with such structural inconsistencies (*ill-formed rows*), we developed, SURAGH which abstracts row structures into structural patterns based on a syntactic pattern grammar. Using the pattern grammar, SURAGH generates syntax-based patterns at the cell, column, and row levels, subsequently identifying the frequent (dominant) row patterns in the input file. These dominant patterns are then used to classify rows into ill-formed and well-formed rows. The experimental findings highlight the superior performance of SURAGH compared to existing state-of-the-art row classifiers and pattern-based error detectors. SURAGH was developed by Hameed, while Vitagliano, Jiang and Naumann contributed valuable discussions.

∗ TASHEEH: A STRUCTURAL ERROR CORRECTION SYSTEM

In Chapter 4, we present TASHEEH, an extension of SURAGH, which is detailed in our publication [35]. Building upon the classification of rows as either ill-formed or well-formed using SURAGH, the primary objective of TASHEEH is to *repair* the structural inconsistencies present in the ill-formed rows. To this end, these ill-formed rows are further classified into two categories: ill-formed unwanted (rows with no data, e.g., table titles, footnotes, or empty rows) and ill-formed wanted (rows containing data but exhibiting additional structural or formatting information and possibly additional columns). After successful identification of the ill-formed wanted rows, TASHEEH effectively repairs the structural inconsistencies within them. To achieve the classification, TASHEEH utilizes a pattern-level distance measure, inspired by sequence alignment, that helps TASHEEH determine the extent to which ill-formed rows differ structurally from well-formed rows. For the transformation, TASHEEH uses a pattern transformation algebra to transform the ill-formed wanted rows into well-formed ones. Our experimental results demonstrate that TASHEEH outperforms current state-of-the-art table extractors, row classifiers, and powerful analytics tools. TASHEEH was developed by Hameed, while Vitagliano, Panse, and Naumann contributed valuable discussions.

∗ MORPHER: DATA PREPARATION WITH SURAGH AND TASHEEH

In Chapter 5, we present our system MORPHER, a desktop-based system featuring a graphical user interface that incorporates the capabilities of both SURAGH and TASHEEH, and is based on our publication [34]. MORPHER serves as a user-friendly tool that allows users to visualize, for an input file, a classification of ill-formed wanted and ill-formed unwanted rows with a corresponding cleaned version. It provides a seamless export of the final results as both CSV and Microsoft Excel workbook (.xlsx) formats for convenient use. MORPHER was developed by Hameed, while Vitagliano and Naumann contributed valuable discussions.

Finally, in Chapter 6, we provide a summary of our work on detecting and correcting structural inconsistencies in raw CSV files. Building upon our findings, we propose future directions for the detection and correction of such inconsistencies, considering both research and practical perspectives. By placing our work within the broader context of data preparation, we highlight the importance of addressing various structural inconsistencies to facilitate efficient data processing.

Chapter 2

DATA PREPARATION FROM AN INDUSTRY PERSPECTIVE: A SURVEY

Data preparation is a crucial step in handling raw data, which often come in messy formats with various encodings, poorly structured rows, and inconsistent patterns. The act of obtaining information from raw data relies on some data preparation process. It is integral to advanced data analysis and data management, not just limited to data science but also for any data-driven application. While there are existing data preparation tools that are operational and useful, there is still scope for improvement and optimization. The demand for prepared data is increasing steadily, especially considering the growing volume and complexity of data. To gain a better understanding of the available data preparation systems, this chapter focuses on several aspects:

Prominent data preparation tools: We identified the leading data preparation tools currently being used in the industry.

Distinctive tool features: We explored the unique features and capabilities that set these tools apart from one another.

The need for preliminary data processing: Even with data preparation tools, preliminary processing is often necessary to handle the messy nature of raw data effectively.

Missing features and abilities: We identified areas where current data preparation tools may be lacking and require further development.

Based on our findings, we advocate for the advancement of automatic and intelligent data preparation techniques beyond the traditional and simplistic methods currently employed. With the continuous growth of data and their challenging nature, there is a pressing need for more sophisticated and efficient data preparation solutions to meet the rising demand for high-quality prepared data.

Our work in this chapter is based on our publication [32], and encompasses the following key contributions:

Organization: We propose and define six comprehensive categories of data preparation tasks and compile a total of 40 common data preparation steps that fall within these categories.

Documentation: To provide practical insights, we conduct a thorough examination of seven selected data preparation tools. We validate the presence and functionality of the identified data preparation features and broader categories for each tool and document our findings in a well-structured preparator matrix[1].

Evaluation: We perform a detailed evaluation of the selected features and functionalities offered by the surveyed data preparation tools. This evaluation enables us to determine whether each tool adequately supports the stated data preparation tasks.

Recommendation: Based on our evaluation and analysis, we identify certain limitations and shortcomings of commercial data preparation tools in general. Additionally, we invite researchers to explore and innovate further in the field of data preparation to address the identified challenges effectively.

The rest of the chapter is organized as follows: Section 2.1 introduces the need for data preparation and presents a real-world data preparation use case to illustrate its importance. In Section 2.2, broader categories of data preparation tasks are described, along with the introduction of "data preparators", representing individual data preparation steps. Section 2.3 details the process of collecting data preparation tools and the selection criteria for the final tools, accompanied by a feature matrix showcasing the offered capabilities. Section 2.4 presents the evaluation criteria and methodology used to assess the selected tools. In Section 2.5, preprocessing challenges encountered during the survey are discussed. Finally, Section 2.6 concludes the chapter, summarizing key findings and emphasizing the role of data preparation in enhancing data quality and decision-making in diverse data-driven applications.

2.1 The Data-to-Application Process

To ensure that data are suitable for the consuming application, several essential phases are involved in the data-to-application life cycle. As outlined in Chapter 1, data generation initially occurs in raw format, often stored in data lakes. However, before sending this raw data to applications, it becomes crucial to enhance its structure and content to make

[1] In the remainder of the chapter, the terms *Preparator Matrix* and *Feature Matrix* are used interchangeably to refer to the set of functionalities of data preparation tools.

it readable and machine understandable. This process involves a series of steps, such as (1) data exploration [18, 53, 88, 90], (2) data collection [60, 107], (3) data profiling [25, 68, 74], (4) data preparation [16, 41, 84], (5) data integration [22, 59, 94], and (6) data cleaning [3, 17, 83] in various orders and iterations.

These aforementioned steps are applied to originally 'raw data', before they are sent to the main application for further processing. In our research focus, and based on evidence from noted surveys, a critical and important step is data preparation. Studies have shown that data scientists spend approximately 80% of the time on preparing the data and about 20% on actual model implementation and deployment [41, 77, 98]. Trifacta's[2] data preparation study shows that 72% of respondents indicated that data preparation by data users is critical, while 88% indicated at least its importance, and only 4% indicated that it is not important for the user [1]. Clearly, these numbers cannot be reduced to 0%, due to the semantic difficulties of understanding and interpreting data. However, the time spent on data preparation can be decreased to a significant amount using sophisticated data preparation techniques, and, in turn, data scientists attain more time for model implementation and deployment, ultimately enhancing the overall efficiency of the data-to-application process.

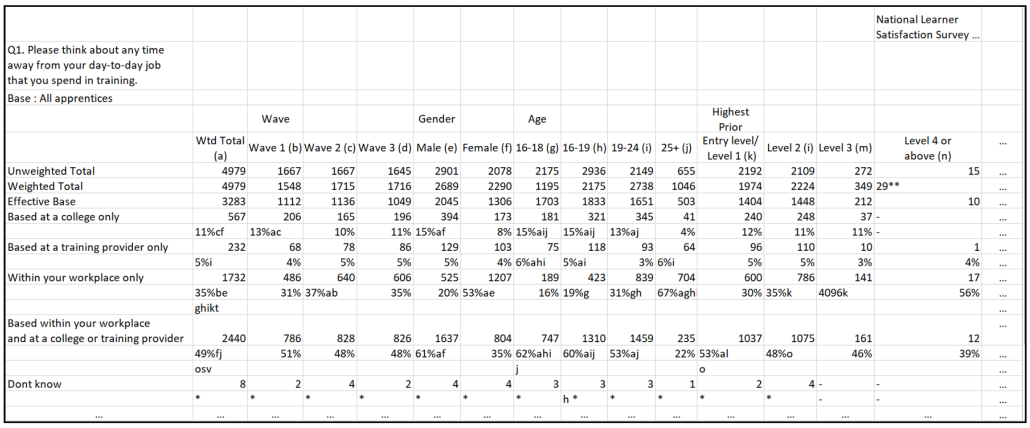
Recognizing the significance and impact of data preparation, developers and researchers have contributed a multitude of techniques to streamline the data preparation process [14, 15, 17, 33, 49, 51, 52, 58, 78, 79, 86, 89, 97, 101]. However, we are still far from creating a fully automated data processing pipeline, in part due to open challenges of raw data files.

Acknowledging the challenges posed by data preparation and its overall importance, many tools have been designed by not only the industry [31, 48, 96], but also by research and academia [11, 27, 102] to address varying use cases. In light of that, we have surveyed commercial data preparation tools to analyze available features and methods. In selecting to focus primarily on commercial data preparation systems, our rationale stemmed from the extensive range of features and services they offer. These commercial tools often cater to a broader spectrum of data-related requirements, making them a valuable reference point for assessing the state of the industry. Moreover, the intent behind this choice was to identify gaps in the market where we can potentially contribute by addressing unmet needs. It is important to note that the survey is not designed for direct comparisons or explicit evaluations of individual tools. Instead, the primary objective is to present an overview of the available features and methods in these tools, showcasing their capabilities and potential impact on the data preparation process.

Through this survey, researchers and practitioners can gain a better understanding of the existing landscape of data preparation tools, enabling them to make informed decisions about tool selection based on specific requirements. Additionally, it provides an avenue for the data preparation community to identify areas for improvement, innovation, and

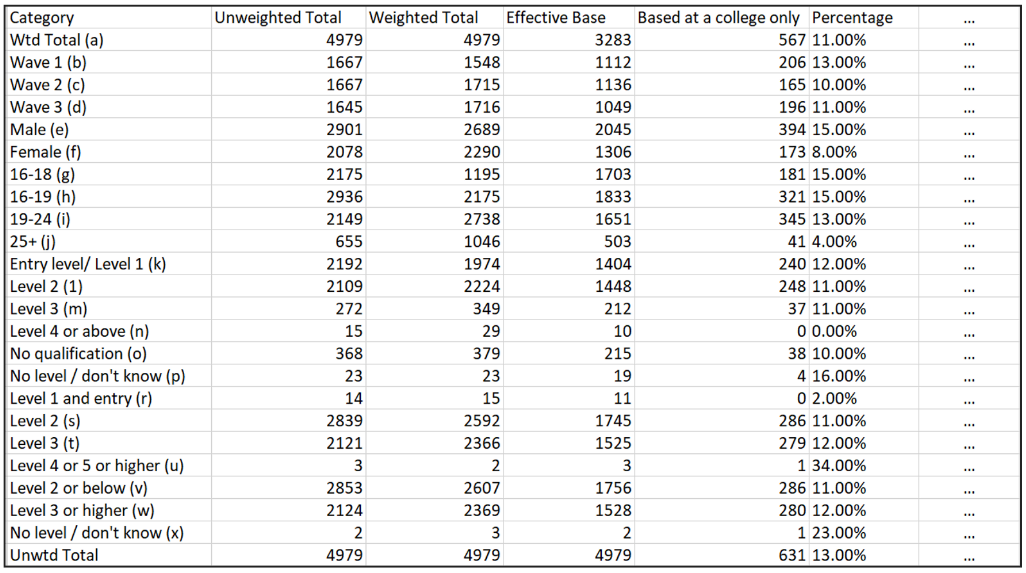
[2] `https://www.trifacta.com/`

collaboration, ultimately contributing to the advancement of data preparation techniques and enhancing the overall efficiency and effectiveness of data processing in the data-to-application life cycle.



														National Learner Satisfaction Survey ...	
Q1. Please think about any time away from your day-to-day job that you spend in training.															
Base : All apprentices															
		Wave			Gender		Age				Highest Prior				
	Wtd Total (a)	Wave 1 (b)	Wave 2 (c)	Wave 3 (d)	Male (e)	Female (f)	16-18 (g)	16-19 (h)	19-24 (i)	25+ (j)	Entry level/ Level 1 (k)	Level 2 (i)	Level 3 (m)	Level 4 or above (n)	...
Unweighted Total	4979	1667	1667	1645	2901	2078	2175	2936	2149	655	2192	2109	272	15	...
Weighted Total	4979	1548	1715	1716	2689	2290	1195	2175	2738	1046	1974	2224	349	29**	...
Effective Base	3283	1112	1136	1049	2045	1306	1703	1833	1651	503	1404	1448	212	10	...
Based at a college only	567	206	165	196	394	173	181	321	345	41	240	248	37	-	...
	11%cf	13%ac	10%	11%	15%af	8%	15%aij	15%aij	13%aj	4%	12%	11%	11%	-	...
Based at a training provider only	232	68	78	86	129	103	75	118	93	64	96	110	10	1	...
	5%i	4%	5%	5%	5%	4%	6%ahi	5%ai	3%	6%i	5%	5%	3%	4%	...
Within your workplace only	1732	486	640	606	525	1207	189	423	839	704	600	786	141	17	...
	35%be	31%	37%ab	35%	20%	53%ae	16%	19%g	31%gh	67%aghi	30%	35%k	4096k	56%	...
	ghikt														...
Based within your workplace and at a college or training provider	2440	786	828	826	1637	804	747	1310	1459	235	1037	1075	161	12	...
	49%fj	51%	48%	48%	61%af	35%	62%ahi	60%aij	53%aj	22%	53%al	48%o	46%	39%	...
	osv						j				o				...
Dont know	8	2	4	2	4	4	3	3	3	1	2	4	-	-	...
	*	*	*	*	*	*	*	h *	*	*	*	*	-	-	...
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

(a) An overview of unprepared data



Category	Unweighted Total	Weighted Total	Effective Base	Based at a college only	Percentage	...
Wtd Total (a)	4979	4979	3283	567	11.00%	...
Wave 1 (b)	1667	1548	1112	206	13.00%	...
Wave 2 (c)	1667	1715	1136	165	10.00%	...
Wave 3 (d)	1645	1716	1049	196	11.00%	...
Male (e)	2901	2689	2045	394	15.00%	...
Female (f)	2078	2290	1306	173	8.00%	...
16-18 (g)	2175	1195	1703	181	15.00%	...
16-19 (h)	2936	2175	1833	321	15.00%	...
19-24 (i)	2149	2738	1651	345	13.00%	...
25+ (j)	655	1046	503	41	4.00%	...
Entry level/ Level 1 (k)	2192	1974	1404	240	12.00%	...
Level 2 (1)	2109	2224	1448	248	11.00%	...
Level 3 (m)	272	349	212	37	11.00%	...
Level 4 or above (n)	15	29	10	0	0.00%	...
No qualification (o)	368	379	215	38	10.00%	...
No level / don't know (p)	23	23	19	4	16.00%	...
Level 1 and entry (r)	14	15	11	0	2.00%	...
Level 2 (s)	2839	2592	1745	286	11.00%	...
Level 3 (t)	2121	2366	1525	279	12.00%	...
Level 4 or 5 or higher (u)	3	2	3	1	34.00%	...
Level 2 or below (v)	2853	2607	1756	286	11.00%	...
Level 3 or higher (w)	2124	2369	1528	280	12.00%	...
No level / don't know (x)	2	3	2	1	23.00%	...
Unwtd Total	4979	4979	4979	631	13.00%	...

(b) An overview of prepared data

Figure 7: A data preparation example

Let us consider an example where a data scientist is given a CSV file, as shown in Figure 7a (viewed as a spreadsheet for easy navigation), from a government data portal[3] to analyze

[3] `http://webarchive.nationalarchives.gov.uk/+/http://www.bis.gov.uk/assets/biscore/further-education-skills/docs/n/11-708-data-nlss-2009.csv` (February, 2019)

how much time each employee is spending on training besides their regular work hours. However, upon opening the file, it becomes clear that the data are not in a coherent relational structure and are laid out in a somewhat human-readable format, making automated analysis impossible. Moreover, in this case, almost 1 000 tables are stacked one below each other (not shown), interleaved by metadata information in the form of preambles and comments that more often than not repeat themselves without meaningful addition. Moreover, inside the actual data values, alphanumeric characters appear in what seem to be otherwise numeric rows. There are apparently inconsistent representations for zeros/null values (e.g., '*','-', or empty cell). To make the data structured and machine-readable, as depicted in Figure 7b, the data scientist needs to perform a sequence of steps on each file before feeding them to the analysis tool. By doing so, the data scientist can avoid the cumbersome and time-consuming manual execution of these tasks and use the same sequence for future use cases.

The data scientist performs the following sequence of steps before feeding the data to their analysis tool:

1. Split the file to isolate one data table at a time. For each obtained table:
2. Remove preamble and comment rows.
3. Unify null-value representations.
4. Remove rows with no meaningful information, e.g., empty rows or rows with only null-values.
5. Clean numeric data rows by removing special characters.
6. Fill missing values, e.g., by value imputation or using functional dependencies.
7. Transpose table.
8. Add missing parts of the header.

It is evident from the aforementioned example that with the help of various data preparation steps, we were able to target messy data and convert them into clean and machine-readable data, highlighting the significance of data preparation in the market for both industry and academia. The application of simple data preparation tasks on raw data files improves their usability, readability, and interpretability. It is essential to emphasize that our focus is solely on addressing the structural errors in rows. Cleaning data errors present in the values themselves is beyond the scope of this chapter and the entire thesis.

2.2 Data Preparation Tasks

Data preparation is not a single step process. Rather, it usually comprises many individual preparation steps, implemented by what we call *preparators*, and which we have organized anew into six broader categories, defined here.

Data discovery involves the analysis and collection of data from various sources [39, 67]. The purpose is to identify data patterns, locate outliers, and detect missing data, among other tasks.

Data validation includes the application of rules and constraints to inspect the data for correctness, completeness, and other data quality constraints [51, 87]. It ensures that the data meets specific requirements and standards.

Data structuring encompasses tasks related to the creation, representation, and structuring of information. Examples include updating data schemas, detecting and changing encodings, and transforming data based on examples provided [36, 50].

Data enrichment involves adding value or supplementary information to existing data from separate sources [10, 106]. It often includes augmenting existing data with new or derived data values using data lookups, generating primary keys, and inserting metadata.

Data filtering focuses on generating a subset of the data to facilitate manual inspection and remove irregular data rows or values [33, 72]. Examples include extracting specific text parts and retaining or deleting filtered rows.

Data cleaning refers to the removal, addition, or replacement of less accurate or inaccurate data values with more suitable, accurate, or representative values [27, 86]. Common data cleaning tasks include deduplication, filling missing values, and trimming whitespace.

Despite our definition, which distinguishes data preparation and cleaning, we include data cleaning steps here as well, as most data preparation tools also venture into this area.

Our set of 40 individual preparators is shown and categorized in Table 4, which is introduced in the next section.

2.3 Data Preparation Tools and Preparator Matrix

Data preparation tools are vital to any data preparation process. They offer implementations of various preparators and provide a user-friendly frontend to apply preparations sequentially or define data preparation pipelines. The flexibility, robustness and intelligence of these tools contribute significantly towards the data analysis and data management tasks.

In this section, we discuss in detail a selection of tools for our research study that are supported by supplementary documentation for experimentation and guidance. Section 2.3.1 discusses the selected data preparation tools (see Table 2 for an overview) and Section 2.3.2 highlights our approach to populate the preparator matrix (see Table 4), organized by data preparator categories with selected preparation tasks. Additionally, we have collected a set of additional functional and non-functional features for these tools, which go beyond specific data preparation tasks (shown in Table 5).

2.3.1 Available Data Preparation Tools

Data preparation is a resource-intensive and time-consuming activity, particularly when lacking automated and mature data preparation tools. Traditionally, data scientists write custom preparation scripts tailored to the specific requirements of each project. Recently, the market has answered to some of the general needs of data preparation by offering commercial data preparation tools that alleviate the burden on data scientists.

To better understand commercial tools and their capabilities, we initiated our study with a discovery phase. We collected notable commercial data preparation tools gathered from business reports and analyses, company portals, and online demonstration videos. Our preliminary investigation resulted in 42 initial commercial tools (shown in Table 1), which we then examined for the extent of their *data preparation* capabilities. Note that the links provided in Table 1 for the discovered tools could be outdated or may have been redirected to different portals due to possible industry mergers and acquisitions[4].

Not all the initially collected tools were solely dedicated to data preparation. Many of them primarily targeted data visualization, data analysis, and business intelligence applications, with only some added data preparation features. To focus on the topic of our survey, we established, necessarily, soft criteria for tool selection. These criteria are as follows:

- Domain specificity: The tools that specifically address the data preparation tasks.
- Comprehensiveness: The extent and sophistication to which tools adequately cover preparation features listed in Section 2.2.
- Guides and documentation: The availability of documentation for the tools, i.e., useful, up-to-date documentation with listings of features and how-to guides.
- Trial availability: The availability of a trial version, giving us the opportunity to test the tools and validate their features.
- GUI: The availability of a comprehensive and intuitive graphical user interface to select and apply preparations.

[4]All tools and their corresponding documented preparators were gathered before 2nd September 2019.

Table 1: Discovered tools with asserted data preparation capabilities. The seven tools highlighted in bold were selected for a detailed analysis in our survey.

Tool name	URL
Altair Monarch Data Preparation	`https://www.datawatch.com/in-action/monarch-draft/`
Alteryx Data Preparation	`https://www.alteryx.com/solutions/analytics-need/data-preparation`
BigGorilla Data Preparation	`https://www.biggorilla.org/`
Cambridge Semantics Anzo	`https://www.cambridgesemantics.com/`
Datameer	`https://www.datameer.com/`
EasyMorph Data Preparation	`https://easymorph.com/`
Erwin	`https://erwin.com/`
FICO	`https://www.fico.com/`
Google Cloud Data Prep by Trifacta	`https://cloud.google.com/dataprep/`
Hitachi-Pentaho Business Analytics	`https://www.hitachivantara.com/en-us/products/data-management-analytics.html`
IBM Data Refinery	`https://www.ibm.com/cloud/data-refinery`
INFOGIX	`https://www.infogix.com/data3sixty/analyze/`
Informatica Enterprise Data Preparation	`https://www.informatica.com/products/data-catalog/enterprise-data-prep.html`
Looker	`https://looker.com/`
Lore IO	`https://www.getlore.io/`
Microsoft Power BI	`https://powerbi.microsoft.com/en-us/`
MicroStrategy	`https://www.microstrategy.com/us/product/analytics/data-visualization`
Modak-nabu	`https://modakanalytics.com/nabu.html`
OpenRefine	`http://openrefine.org/`
Oracle Analytics Cloud	`https://www.oracle.com/business-analytics/analytics-cloud.html`
Paxata Self Service Data Preparation	`https://www.paxata.com/self-service-data-prep/`
Qlik Data Catalyst	`https://www.qlik.com/us/products/qlik-data-catalyst`
Quest Toad Data Point	`https://www.quest.com/products/toad-data-point/`
Rapid Insight	`https://www.rapidinsight.com/solutions/data-preparation/`
RapidMiner Turbo Prep	`https://rapidminer.com/products/turbo-prep/`
SAP Agile Data Preparation	`https://www.sap.com/germany/products/data-preparation.html`
SAS Data Preparation	`https://www.sas.com/en_us/software/data-preparation.html`
Smarten Advanced Data Discovery	`https://www.smarten.com/self-serve-data-preparation.html`
Solix Common Data Platform	`https://www.solix.com/products/solix-common-data-platform/`
Sparkflows	`https://www.sparkflows.io/data-science`
Tableau Prep	`https://www.tableau.com/products/prep`
Talend Data Preparation	`https://www.talend.com/products/data-preparation/`
Tamr	`https://www.tamr.com/`
Teradata Vantage	`https://www.teradata.com/Products/Software/Vantage`
TIBCO Spotfire Analytics	`https://www.tibco.com/products/tibco-spotfire`
TMMData	`https://www.tmmdata.com/`
Trifacta Wrangler	`https://www.trifacta.com/products/wrangler-editions/`
Unifi Data Platform	`https://unifisoftware.com/platform/`
Waterline Data	`https://www.waterlinedata.com/`
Workday-Prism Analytics	`https://www.workday.com/en-us/applications/analytics/prism-analytics.html`
Yellowfin Data Prep	`https://www.yellowfinbi.com/suite/data-prep`
Zoho Analytics	`https://www.zoho.com/analytics/`

Table 2: Selected data preparation tools

Tool name	URL
Altair Monarch Data Preparation	`https://www.datawatch.com/in-action/monarch-draft/`
Paxata Self Service Data Preparation	`https://www.paxata.com/self-service-data-prep/`
SAP Agile Data Preparation	`https://www.sap.com/germany/products/data-preparation.html`
SAS Data Preparation	`https://www.sas.com/en_us/software/data-preparation.html`
Tableau Prep	`https://www.tableau.com/products/prep`
Talend Data Preparation	`https://www.talend.com/products/data-preparation/`
Trifacta Wrangler	`https://www.trifacta.com/products/wrangler-editions/`

Table 3: Selected data preparation tools with their current affiliations

Tool name	URL
Altair Monarch	`https://altair.com/monarch`
DataRobot (Paxata)	`https://www.datarobot.com/platform/dataprep/?redirect_source=paxata.com`
SAP Data Intelligence (SAP)	`https://www.sap.com/germany/products/technology-platform/data-intelligence.html`
SAS Viya (SAS)	`https://www.sas.com/en_us/trials/software/viya/viya-trial-form.html`
Tableau Prep Builder	`https://www.tableau.com/products/prep`
Talend a Qilk Company (Talend)	`https://www.talend.com/products/data-preparation/`
Alteryx (Trifacta Wrangler)	`https://www.alteryx.com/products/capabilities/data-preparation-tools`

- Customer assistance: Compliant support teams that assist users with generic and specific tool queries when needed.

By applying these criteria, we aimed to ensure that the selected tools are well-suited for data preparation tasks and offer a robust and user-friendly experience for data professionals. This approach allowed us to focus on the most relevant tools and gain meaningful insights into their capabilities and functionalities.

After carefully applying our selection criteria, we have identified seven qualifying tools for detailed investigation in our data preparation survey, as listed in Table 2. In the following, we will discuss each of these tools in alphabetical order, providing a comprehensive overview of their features.

As mentioned earlier, some links to the discovered tools may have become obsolete or may now refer to other portals due to mergers and acquisitions in the industry. This situation has also been observed for the selected tools listed in Table 2. To address this, we have now provided new links in Table 3, which correspond to the current versions[5] and names of the tools and their affiliated companies. Despite potential changes in the tool's names and associations, it is important to acknowledge that the survey presented in this chapter is

[5] Accessed on 1st August 2023.

based on the tools originally discussed in the published paper [32], and the focus remains on the functionalities of the tools as outlined in the original research publication.

Altair Monarch Data Preparation formerly known as Datawatch before the company's merger with Altair, offers a range of data preparators for structured data, as well as the unique capability to transform tables within PDF and text files into tabular data. The tool's table extractor feature allows users to extract files independently as tables or merge them with other tables or files using various join and union operations. This functionality provides data professionals with enhanced flexibility when working with diverse data sources and formats, making it a valuable addition to the data preparation toolkit.

Paxata Self-Service Data Preparation is a comprehensive tool that excels in organizing and preparing structured data, while also demonstrating efficient handling of semi-structured data. In addition to typical data preparation features, Paxata introduces a unique functionality called "data filtergrams". These filtergrams enable users to perform filter operations on data through various visual interactions, such as text filtergrams, numeric filtergrams, boolean filtergrams, and source filtergrams. The emphasis on user experience makes Paxata a user-friendly tool, accommodating both data experts and non-experts alike, thereby streamlining the data preparation process for a wider range of users. The tool is currently known as DataRobot.

SAP Agile Data Preparation is built on top of SAP's HANA database system and provides a comprehensive set of common data preparators with additional system-specific features. One notable feature is "Schedule Snapshot", enabling users to take periodic snapshots of data and retrieve information from remote sources as needed. This feature minimizes the necessity of manually writing queries for data retrieval, freeing up valuable time for data scientists to allocate towards other essential data-centric activities. The tool also offers interactive suggestions, assisting users in navigating and preparing data efficiently. Furthermore, the support for multi-user access allows collaborative data preparation, facilitating teamwork and enhancing productivity. With its powerful capabilities and collaborative features, SAP Agile Data Preparation empowers users to streamline their data preparation tasks within the SAP HANA ecosystem. Currently, SAP provides data preparation features through their new system called Data Intelligence.

SAS Data Preparation is integrated into the SAS Viya System Management, leveraging distributed in-memory processing for efficient operations. In addition to standard data preparation features, SAS offers code-based transformations, allowing users to write and share custom code for data transformation. This support for code re-usability enables users to create and reuse preparation pipelines, enhancing the flexibility and customization options for data processing tasks.

Tableau Prep adopts a workflow-based approach to efficiently organize and prepare untidy data. The tool's interactive interface and workspace plans empower users to execute multiple operations simultaneously. Tableau Prep consists of two components: Tableau Prep Builder, which facilitates the creation of data flows, data management, and the application of operations on the data, and Tableau Prep Conductor, which enables users to share, schedule, and monitor these data flows. With its seamless integration of data preparation tasks, Tableau Prep offers a user-friendly and comprehensive solution for handling complex data preparation workflows.

Talend Data Preparation stands out with its tailored and specific data preparation functionalities designed to address diverse tasks. For data cleaning, it offers various functions dedicated to handling numeric data values, strings, and date inputs. A notable feature is "selective sampling", allowing users to work with a subset of the data for insights and operations that can later be applied to the entire dataset. Talend actively addresses system-level challenges, exemplified by its intelligent pipeline automation feature, which enables the saving and reusability of data preparation tasks or steps. With its powerful capabilities and intelligent features, Talend Data Preparation remains a robust choice for handling data preparation tasks efficiently and effectively. The tool is currently associated with Qlik Analytics, following a merger between the Talend and Qilk companies.

Trifacta Wrangler excels in data preparation with its diverse range of preparation functions and intelligent pattern prediction, providing valuable suggestions to aid users in data transformation. Beyond the common preparation tasks, it offers intriguing additional features, including primary key generation, data transformation by example, and permitted character checks. Wrangler leverages regular expressions for many of its pattern-based functionalities. Notably, the preparators in Wrangler exhibit a high degree of sophistication. For instance, the locate outlier not only identifies outliers but also generates a histogram of the entire column, adding valuable insights for data analysis. The tool originated from the Wrangler project [54], and its continued development showcases its commitment to empowering users with advanced data preparation capabilities. Currently, the tool is part of Alteryx's data management solutions following the company's acquisition of Trifacta.

2.3.2 Preparator Matrix

Table 4 provides a preparator matrix showing which preparator is supported by which tool in each of the six categories. The population of this preparator matrix was not a trivial task. Initially, we analyzed the tool's documentation to gather all available preparators. We then downloaded trial versions of all tools and (generously) evaluated to determine whether they offered the functionality of each of the 40 preparators. The process of populating the matrix is detailed further in Section 2.4.

Table 4: Preparator matrix for data preparation tools

Categories	Available preparators	Data preparation tools						
		Altair	Paxata	SAP	SAS	Tableau	Talend	Trifacta
Data discovery	Locate missing values (nulls)	✓	✓	✓	✓	✓	✓	✓
	Locate outliers		✓		✓			✓
	Search by pattern	✓	✓	✓	✓	✓	✓	✓
	Sort data	✓	✓	✓	✓	✓	✓	✓
Data validation	Compare values (selection and join)	✓	✓	✓		✓	✓	✓
	Check data range	✓	✓	✓		✓	✓	✓
	Check permitted characters							✓
	Check column uniqueness	✓	✓	✓		✓	✓	✓
	Find type-mismatched data		✓	✓		✓	✓	✓
	Find data-mismatched datatypes		✓				✓	✓
Data structuring	Change column data type	✓	✓	✓	✓	✓	✓	✓
	Delete column	✓	✓	✓	✓	✓	✓	✓
	Detect & change encoding						✓	✓
	Pivot / unpivot	✓	✓	✓		✓		✓
	Rename column	✓	✓	✓	✓	✓	✓	✓
	Split column	✓	✓	✓	✓	✓	✓	✓
	Transform by example [50]						✓	✓
Data enrichment	Assign semantic data type				✓	✓	✓	
	Calculate column using expressions	✓	✓	✓	✓	✓	✓	✓
	Discover & merge external data	✓	✓	✓			✓	✓
	Duplicate column	✓	✓	✓		✓	✓	✓
	Generate primary key column			✓				✓
	Join & union	✓	✓	✓	✓	✓	✓	✓
	Merge columns	✓		✓		✓	✓	✓
	Normalize numeric values	✓	✓	✓	✓	✓	✓	✓
Data filtering	Delete/keep filtered rows	✓	✓	✓	✓	✓	✓	✓
	Delete empty and invalid rows	✓	✓	✓	✓	✓	✓	✓
	Extract value parts	✓			✓		✓	✓
	Filter with regular expressions							✓
Data cleaning	Change date & time format	✓	✓	✓	✓	✓	✓	✓
	Change letter case	✓	✓	✓	✓	✓	✓	✓
	Change number format	✓	✓	✓	✓	✓	✓	✓
	Deduplicate data	✓	✓	✓	✓		✓	✓
	Delete by pattern	✓	✓		✓	✓	✓	✓
	Edit & replace cell data	✓	✓	✓	✓	✓	✓	✓
	Fill empty cells	✓	✓				✓	✓
	Remove extra whitespace	✓	✓	✓	✓	✓	✓	✓
	Remove diacritics			✓				
	Standardize strings by pattern		✓	✓	✓	✓	✓	✓
	Standardize values in clusters		✓	✓	✓	✓	✓	✓

The basic functionality of most preparators is self-explanatory by their name – their precise implementation and parameterization might differ from tool to tool, and it would be beyond the scope of this chapter to describe each. Instead, we have selected three exemplary preparators to illustrate their function and the intricacies involved in even simple data preparation tasks. We use the same three preparators in Section 2.4 to highlight some capabilities of individual tools.

Keep or Delete Filtered Rows serves the purpose of customizing data views by applying filtering operations based on specified predicates. This preparator enables users to filter data that may need to be deleted, extracted, or modified for subsequent analysis. In its basic form, the filtering process involves applying simple predicates similar to SQL conditions. However, a more advanced and intelligent approach would involve utilizing a richer language, such as regular expressions, to perform filtering tasks. This enhanced filtering capability allows for more complex and flexible data manipulation to suit diverse data preparation requirements. For instance, consider a dataset of customer reviews, and a user wants to extract all reviews that contain positive sentiments. The user could use a regular expression like `.*\b(?:good|excellent|satisfactory)\b.*` to *filter* out only the rows with positive feedback, providing a positive subset for analysis.

Value Standardization is a common data preparation operation used to transform the values of a column to adhere to a specific standard. This standard can either be derived from frequent patterns found within the data or obtained from an external authoritative source. A more advanced preparator could help in automatically detecting relevant data clusters, making the standardization process more efficient and effective. One popular technique employed in this context is fuzzy matching for clustering, which allows for a more accurate and meaningful representation of the data during standardization. By applying value standardization, datasets can be homogenized, leading to improved consistency and comparability, which is crucial in various data analysis and integration tasks. For example, consider a dataset containing information about countries and their currencies. The currency column in this dataset might have values for US currency like “US Dollars”, “USD”, and “$”, which can create inconsistency and make it challenging to analyze the data accurately. To standardize the values in the currency column, a *Value Standardization* preparator can automatically detect relevant data clusters and apply fuzzy matching. It recognizes that all these variations should be standardized to one consistent version and may choose, for instance, “USD” for consistency.

Split Column is valuable for dealing with messy data containing values comprising multiple atomic parts. This preparator allows data to be divided into multiple columns based on defined criteria, such as splitting after a comma or at the last whitespace within a string. A more sophisticated preparator could identify split column cases by using existing patterns in data, and be able to handle splits into more than two columns. For example, consider a dataset with a column containing full names in a single field. To extract the first and last names separately, a *Split Column* preparator can be applied. It can be configured to split the column based on a word separator (e.g., whitespace), separating the full names into two distinct columns: “First Name” and “Last Name”.

In our survey, we came across many functional and non-functional system features that did not cater to our data preparation focus. Nonetheless, these features are important and interesting when explored and utilized. Thus, we have gathered them in Table 5.

2.4 Evaluation of Selected Tools

In this section, we delve into our assessment of data preparation tools, highlighting their performance in data quality enhancement features. Overall, the evaluation aimed to provide insights into the strengths of each data preparation tool, helping readers understand which tool might be the best fit for their specific data preparation needs.

We evaluated each of the preparators on three datasets downloaded from public data repositories: (i) Kaggle – 120 years of Olympic history (athletes and results)[6], (ii) IMDb – data about movies[7], and (iii) UK government web archive, as mentioned in Section 2.1.

To better explain how we evaluated the preparators, we provide an example for each of the three preparators discussed in the previous section. In general, even the simplest versions of the respective preparators earned the tool a checkmark in our matrix (Table 4). More sophisticated versions could incorporate preparators that intelligently detect relevant problems and actively provide suggestions for their configuration, e.g., suitable regular expressions or standard formats.

Keep or Delete Filtered Rows: Data filtering techniques play a crucial role in enhancing data quality by employing predefined criteria, such as removing rows with empty values or those that do not adhere to specific user-defined patterns. A wide range of data preparation tools provides diverse filtering options. For example, Talend Data Preparation offers filters based on patterns using pre-defined syntactic data types, allowing users to apply precise and efficient data filtering strategies to tailor data to their specific needs.

Example 1 *Using pattern filtering, a user might want to keep only official email addresses. Using Talend's syntax, corresponding patterns might be:*

`[word]@ibm.[word]`, `[char].[word]@ibm.[word]`

Thus, private addresses such as `bob1992@gmail.com` *or* `alice25@yahoo.com` *would be filtered, while* `a.peter@ibm.com` *would be retained.*

[6]https://www.kaggle.com/heesoo37/120-years-of-olympic-history-athletes-and-results
[7]ftp://ftp.fu-berlin.de/pub/misc/movies/database/frozendata/

Table 5: Further features of data preparation tools

Altair	Paxata	SAP	SAS	Tableau	Talend	Trifcata
Advanced Filtering	Add Comments	Action History	Create Custom Code	Adjust Sample Size	Advanced Filtering	Add Comments
Audit User Actions	Advanced Filtering	Advanced Filtering	Data Lineage	Advanced Filtering	Aggregation Using Charts	Advanced Filtering
Comparison Functions	Aggregation	Aggregation	Data Sampling	Aggregation	Audit User Actions	Aggregation
Copy and Paste Columns	Check Spelling	Comparison Functions	Job Monitoring	Change Color Scheme	Calendar Formats	Comparison Functions
Create Summaries	Cluster Data Prep Steps	Copy and Paste Columns	Job Scheduling	Check Spelling	Country Name into Codes	Copy and Paste Columns
Data Histogram	Comparison Functions	Data Quality Statistics	Maintain Log	Data Size Details	Data Masks	Data Histogram
Data Lineage	Copy and Paste Columns	Date & Time Formats	Math functions	External Data Use	Date & Time Formats	Data Profiling
Data Sampling	Data Histogram	Deduplication Statistics	Multi User Access	Group and Replace	External Data Use	Date & Time Formats
Data Size Details	Data Lineage	External Data Use	Preparation Versions	Group Tasks	Extract Quarter from Date	Diagnose Failed Jobs
Edit Field Values	Data Profiling	Hide Column	Refresh Data from Source	Intelligent Bar	Find and Group	External Data Use
External Data Use	Data Sampling	Intelligent Bar	Reorder Preparation Steps	Maintain Log	Intelligent Bar	Fix Dependency Issues
Hide Column	Data Size Details	Maintain Log	Search and Replace	Math Functions	Maintain Log	Group and Replace
Job Scheduling	Date & Time Formats	Math Functions	Share Dataset	Mini Maps	Math Functions	Initial Parsing Steps
Maintain Log	External Data Use	Multi User Access	String Functions	Multi Language Support	Multi Language Support	Intelligent Bar
Math Functions	Find and Group	Multiple Graphs for Visuals	Transpose Data	Preparation Versions	Preparation Versions	Logical Functions
Move Column	Group and Replace	Preparation Versions	View Table Properties	Publish Flows	Reorder Preparation Steps	Maintain Log
Preparation Versions	Intelligent Bar	Refresh Data from Source	Visual Feedback	Refresh Data from Source	Search and Replace	Manage Flows with Folders
Refresh Data from Source	Intelligent Ingest	Reorder Preparation Steps	Work with Plans	Reorder Preparation Steps	Share Dataset	Manage String Lengths
Row and Column Counts	Maintain Log	Search and Replace		Schedule Flows	String Functions	Math Functions
Search and Replace	Math Functions	Share Dataset		Search and Replace	Suggestions	Multi Language Support
String Functions	Multi Language Support	String Functions		Share Dataset	Swap Column Content	Preparation Versions
Transpose Data	Preparation Versions	Suggestions		String Functions	Use Metric Symbols	Reorder Preparation Steps
Visual Feedback	Publish Dataset			Suggestions	Visual Feedback	Row and Column Counts
	Reorder Preparation Steps			Visual Feedback		Search and Replace
	Search and Replace					Sequence Datasets
	Send Notifications					Share Dataset
	Share Dataset					String Functions
	String Functions					Suggestions
	Suggestions					Target-driven preparation
	Version History					Track Data Changes
	Visual Feedback					Visual Feedback
						Workflow Automation

Value Standardization: A typical step in data preparation, especially when dealing with heterogeneously formatted values, involves standardization using patterns. For instance, phone number patterns, datetime patterns, and patterns by example are commonly used to achieve standardization. Tools like Trifacta Wrangler offer valuable suggestions for applicable patterns and transform data to adhere to the suggested or selected standard.

Additionally, value standardization is essential for handling different representations of the same real-world value within a column. By grouping these varied representations and transforming them into a single, common representation, data consistency is achieved. This ensures that data are uniformly formatted, making it easier for analysis and facilitating accurate insights and decisions.

Example 2 *Trifacta might group rows with city values `NY`, `NYC`, and `New York City` and standardize all occurrences to `New York City`. Alternatively, users can review the cluster and manually choose the correct standard value.*

Split Column: Multi-valued columns reduce flexibility in handling data (and also their readability). The Split column preparator is designed to address this issue by dividing such columns based on a specified criterion. For instance, SAS Data Preparation offers multiple ways to implement this technique, such as splitting based on *on*, *before*, or *after* a delimiter, on a fixed length, and using "quick split", which intelligently identifies an appropriate split criterion. This operation not only enhances data organization but also streamlines subsequent data processing tasks, improving the overall efficiency of the data preparation workflow.

Example 3 *Using a comma as a delimiter, the user wants to split the location column (and implicitly trim accrued whitespace). In addition, the user specified headers for the output columns. As can be seen in the example, due to a missing value in the original data, the value "USA" is misplaced; a later validation step might identify this error.*

`Input:`

Location	...
Melbourne, Victoria, Australia	
San Francisco, USA	
Potsdam, Brandenburg, Germany	

`Output:`

City	***State***	***Country***	...
Melbourne	*Victoria*	*Australia*	
San Francisco	*USA*		
Potsdam	*Brandenburg*	*Germany*	

The three preparators discussed above are presented as examples among various available ones. We intend to provide readers with a better understanding of different types of data preparators, rather than focusing on specific tools or their comparative features. While the examples presented in the evaluation of preparators are basic, they were used to assess all the preparators listed in Table 4 or variants of these examples. This approach was chosen to maintain simplicity and a consistent evaluation level, although some tools showcased the ability to handle more complex problems.

2.5 Challenges

In this section, we present some of the most prominent challenges that we encountered during our research and survey. These challenges offer valuable insights and serve as key takeaways, contributing to a better understanding of data preparation tools and their potential areas of enhancement.

Dataset preprocessing: Interestingly, despite being data preparation tools, all tools that we have surveyed and explored require a pre-prepared or cleaned dataset as their input. For example, if the file had comment-lines, additional header or footer information, or poorly placed quotation marks, it was misinterpreted and loaded improperly. In fact, most tools make the following broad assumptions:

- Single table file (no multi-table files)
- Specific file encoding
- No preambles, comments, footnotes, etc.
- No intermediate headers
- Specific line-ending symbol
- Homogeneous delimiters
- Homogeneous escape symbols
- Same number of fields per row
- Relational data (no nested or graph-structured data, such as XML, JSON or RDF)

Some of the assumptions made by data preparation tools present intriguing research challenges, and researchers have addressed these challenges in isolated contexts. For instance, detecting tables in complex spreadsheets has been explored in works like [21, 101], while converting web tables to relations has been tackled in studies such as [14, 65]. These individual research endeavors shed light on specific aspects of data preparation and contribute to the ongoing advancement of techniques and tools in this domain.

In line with this progress, Chapter 3 introduces our system, SURAGH, designed to automatically identify rows with structural issues (preambles, heterogeneous row delimiters, rows with inconsistent cell counts, etc.) that might otherwise hinder the smooth ingestion of files. In continuation of our efforts, Chapter 4 presents an automated solution, TASHEEH, for repairing such rows, thereby facilitating seamless data ingestion.

User expertise needed: Another challenge we experienced was the level of expertise required to use data preparation tools effectively. Most tools require the user to be an expert in the dataset domain and have prior knowledge and understanding of the datasets and of the data preparation goal.

Moreover, many tools offer advanced features that go beyond simple predicates, such as the ability to use regular expressions for tasks like matching, splitting, or deleting data. However, crafting intricate regular expressions can be challenging for typical domain experts who may not possess extensive IT knowledge. This requirement for a certain level of technical expertise and familiarity with data domains may limit the accessibility and adoption of data preparation tools by non-technical users.

Automatic preparation: While all the surveyed data preparation tools offer valuable functions, the majority of them lack intelligent solutions for more automated data preparation tasks. For instance, the preparator Deduplicate data, which removes duplicate rows from a source, is typically limited to exact match conditions. A more sophisticated version would involve deduplication based on similarity measures. Another common issue is column heterogeneity, where columns contain data in multiple formats. Currently, users must manually filter and prepare these different groups separately. An automatic homogenization process would be helpful, but it also poses a challenging research problem.

The lack of more advanced and intelligent automation in data preparation tools underscores the potential for further research and development in this area. Introducing smart solutions to automate complex data preparation tasks would enhance the efficiency and effectiveness of data preparation processes, ultimately benefiting data scientists and machine learning engineers across various domains.

Unstructured data: The scope of our survey is that of preparing structured data. However, many datasets include some textual component, such as product descriptions, plot synopses, etc. Such textual data can also benefit from basic preparation steps, such as stopword-removal, lemmatization, or sentence breaking, to then, e.g., perform named entity extraction or sentiment analysis.

One outlook is to include such capabilities in the existing tools for structured data preparation. Another is to develop a dedicated framework and toolset for the case of unstructured data preparation (or text preparation), similar to the tools survey in this chapter.

Preparation pipelining: Data preparation is not a one-step process. Rather, it involves many subsequent steps, organized in a preparation pipeline to gradually transform a dataset towards the desired output. Creating and managing pipelines yields many system-level challenges and opportunities. For instance, preparation suggestion, pipeline adaption, and pipeline optimization, that need to be addressed accordingly. Such systematic data preparation can benefit from a comprehensive and well-defined yet extensible set of operators. By incorporating the ability to create and manage preparation pipelines, data preparation tools can be massively improved and generalized for more intelligent and self-service techniques. After a pipeline has been established, optimization and customization policies can be designed according to the needs of the problem at hand or business use cases under considerations.

To summarize, existing tools already cover basic data preparation needs by implementing simple and obvious preparators. In some instances, we observed more advanced capabilities, such as automated pattern suggestions or even preparator recommendations tailored to the data at hand. All of these tools are excellent platforms for further development in several dimensions, as outlined above. In our opinion, the need for self-service data preparation and tool capabilities goes beyond current technology, and we encourage research in this emerging field.

2.6 Conclusion

In this chapter, we have discussed and surveyed major commercial tools for data preparation. We have gathered and organized their capabilities in the form of "preparators", categorized into six distinct groups.

As the volume of data continues to grow, there is an increasing opportunity to derive value by integrating and analyzing these vast datasets. Consequently, the demand for data preparation and cleaning also grows, as data often come with various syntactic and semantic issues that require careful automated or manual handling. While current technology offers a range of data preparation tools, the process still heavily relies on manual intervention from data experts or domain experts with data engineering skills. To meet the demands of this rapidly evolving data market, the development of sophisticated and intelligent solutions for automatic data preparation is crucial. However, achieving a fully automated data preparation toolkit remains a significant challenge, requiring extensive research and development. Nonetheless, recognizing the critical need for such advancements, we encourage continuous efforts to explore and innovate in the field of data preparation.

Chapter 3

SURAGH: A STRUCTURAL ERROR DETECTION SYSTEM

Despite the presence of the established standard [45], data entry into CSV files is prone to errors because users and applications do not always adhere to this standard; moreover, the standard itself is rather loose. This behavior is also present in CSV files available on open data portals. Out of 2 066 files randomly selected from a government data portal[8], we were unable to directly load 418 (20.2%) of them into an RDBMS due to inconsistent row structures. We observe the same behavior with other tools, including business intelligence and data wrangling tools. We refer to such rows as "ill-formed" (see Section 3.2.1 for a formal definition). Eventually, such rows can hinder or completely halt data loading and other data processing operations. Similarly, such rows impose challenges for machine learning algorithms during data annotation, manipulation operations, and algorithm training phases. Manually identifying ill-formed rows involves substantial effort, domain expertise and remains susceptible to errors.

To address this challenge, this chapter introduces our automated solution, SURAGH[9], designed to identify ill-formed rows within a file by mapping column values to syntax-based patterns: syntactic patterns (see Definition 1).

In lieu of conventional regular expressions, we leverage syntactic patterns as a more streamlined form of representation. These patterns are easy to understand and represent data structures comprehensively. The syntactic patterns we introduce can assist the user in various tasks:

- **Data ingestion:** The core use-case of SURAGH is to prevent frustrating experiences of raw data not loading into a host system.
- **Table detection:** Ill-formed rows can cause ambiguity for table detection algorithms when detecting table boundaries. Using SURAGH, we can remove these rows and

[8]`https://www.data.gov/`

[9]SURAGH is an Urdu word that means to investigate an event; to obtain clues about something.

improve the accuracy of boundary detection. In addition, our approach can help users identify multiple vertically aligned tables in a file.

- **Data standardization:** Syntactic patterns can serve as a basis for standardizing data into a uniform format, as they reflect typical problems in ill-formed rows (outlier patterns) and desired structure (frequent patterns).

Our work in this chapter is based on our publication [33], and makes the following main contributions:

1. A formalization to describe file schema, syntactic patterns, and ill/well-formedness of rows.
2. A set of 131 files from five open data sources, each annotated for ill-formed and well-formed rows for a total of 210 550 rows. The datasets are publicly available together with the annotations and code at the project page[10].
3. A method, SURAGH, that automatically recognizes ill-formed rows by mapping column values to syntax-based patterns.
4. A wide range of experiments conducted to validate SURAGH and demonstrate the applicability of our approach.

The remainder of this chapter is structured as follows: Section 3.1 discusses data loading challenges and presents illustrative examples of ill-formed rows at both row and column levels. Section 3.2 defines relevant concepts, provides a formal definition of ill/well-formed rows, and describes the syntactic patterns grammar specified by EBNF rules [23]. The workflow of the proposed algorithm and the significance of pruning in distinct phases are detailed in Section 3.3. Section 3.4 presents the datasets and the annotation process. Our experimental evaluation of SURAGH is presented in Section 3.5. Section 3.6 discusses prominent research conducted in the field of pattern-based approaches, and finally, Section 3.7 concludes our study.

3.1 Data Loading Obstacles

Recall from Chapter 2, parsing and storing raw data without standardized formats harbors challenges, such as invalid characters due to incorrect parsing, preambles or comments, or inconsistent formatting that causes problems in data manipulation operations. Data scientists and machine learning engineers spend much of their development time cleaning and preparing data [41, 77, 98]. Existing data cleaning [15, 17, 62, 79, 86] and data preparation [32, 49, 58, 89, 108] techniques address data preprocessing tasks. However, improving

[10] `https://github.com/HPI-Information-Systems/SURAGH`

user experience in designing and carrying out an automated data preprocessing pipeline still has open challenges.

Among other challenges, detecting "ill-formed" rows in CSV files is a difficult problem. These rows do not adhere to the file's structural patterns and prevent data from being loaded, let alone being consumed by downstream applications. We expect that most readers have first-hand experience of a load-process aborting after many minutes due to some mistake, such as an incorrectly encoded value or a row too long towards the end of the input file. Another common experience is the need to "clean up" a file by removing preambles, footnotes, etc., before attempting to import the data into a database or other application. Our goal is to identify such problems in advance and alert users or machines about the problematic rows.

A row may be ill-formed due to inconsistencies that can exist at both column- and row-levels. *Column-level inconsistencies* include inconsistent column values, such as incorrectly handled escape characters, string values in numeric columns, inconsistent formatting, a null or a missing value, etc. Figure 8 shows several real-world examples.

Missing string quotes, resulting in creating an unintended additional column.

```
1,"A Lamusi","M",23,170,60,"China","CHN","2012 Summer",2012,"Summer","London","Judo",Speed Skating Women's 1,000 metres,NA
```

Mistakenly placed a delimiter (;) resulting in creating a new column, while expecting a new-line separator.

```
1;0.2789999999;831667;0.21100000000000002;0;4BJqT0PrAfrxzMOxytFOIz;0.878,10,0.665;-20.096;1;Piano Concerto;
```

Mistakenly placed (-) with (/) in the date column consequently, the entry no longer matches the other values in that column.

```
1,Bob Miller,University of California,Batch2020,12/10/1980,USA
```

Mistakenly placed postal code in city name column consequently, the entry does not match the expected content.

```
1,John,27,street 20 5th avenue new york,10001,USA
```

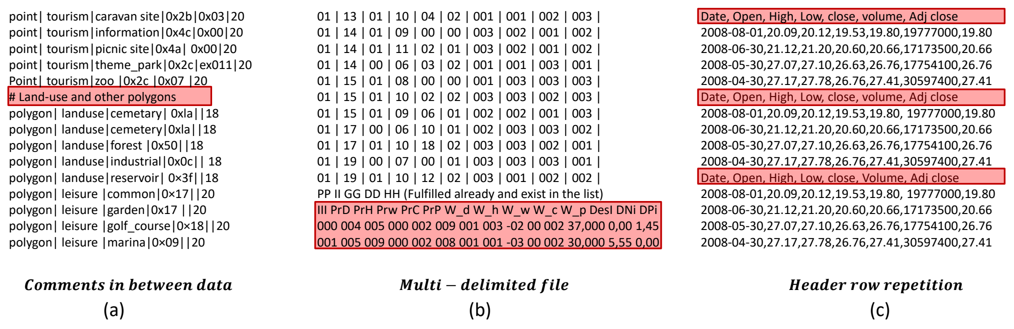
Due to the presence of a new-line separator between values, some values are split across multiple rows.

```
jqXKi/fIcxO8zBKMyaedfQ==,Propiedad,2019-10-14,9999-12-31,2019-10-14,"TEMPORADA 2020\r\nCon Hermosa pileta\r\nQuincho Asador"
```

Figure 8: Examples of ill-formed rows due to column-level inconsistencies

Row-level inconsistencies occur when entire rows appear in the file but do not contain the expected data. For instance, such rows may contain metadata or comments, group headers, or are due to misplaced delimiter or end-of-line characters. Figure 9 again shows several real-world examples.

Although CSV files are mainly comma-separated, we observe many CSV files with other delimiters, which are also in the scope of our work. In the context of our research, we do assume that a file contains tabular data. Dealing with semi-structured and unstructured data is beyond the scope of this work. Also, to limit search space size, our approach is designed for data files that contain ASCII values. The extension to larger character sets is conceptually easy but computationally expensive.



```
point| tourism|caravan site|0x2b|0x03|20
point| tourism|information|0x4c|0x00|20
point| tourism|picnic site|0x4a| 0x00|20
point| tourism|theme_park|0x2c|ex011|20
Point| tourism|zoo |0x2c |0x07 |20
# Land-use and other polygons
polygon| landuse|cemetary| 0xla||18
polygon| landuse|cemetery|0xla||18
polygon| landuse|forest |0x50||18
polygon| landuse|industrial|0x0c|| 18
polygon| landuse|reservoir| 0×3f||18
polygon| leisure |common|0×17||20
polygon| leisure |garden|0x17 ||20
polygon| leisure |golf_course|0×18||20
polygon| leisure |marina|0×09||20
```

Comments in between data

(a)

```
01 | 13 | 01 | 10 | 04 | 02 | 001 | 001 | 002 | 003 |
01 | 14 | 01 | 09 | 00 | 00 | 003 | 002 | 001 | 002 |
01 | 14 | 01 | 11 | 02 | 01 | 003 | 002 | 001 | 002 |
01 | 14 | 00 | 06 | 03 | 02 | 001 | 001 | 003 | 001 |
01 | 15 | 01 | 08 | 00 | 00 | 001 | 003 | 003 | 003 |
01 | 15 | 01 | 10 | 02 | 02 | 003 | 003 | 002 | 003 |
01 | 15 | 01 | 09 | 06 | 01 | 002 | 002 | 001 | 003 |
01 | 17 | 00 | 06 | 10 | 01 | 002 | 003 | 003 | 002 |
01 | 17 | 01 | 10 | 18 | 02 | 003 | 003 | 002 | 001 |
01 | 19 | 00 | 07 | 00 | 01 | 003 | 003 | 003 | 001 |
01 | 19 | 01 | 10 | 12 | 02 | 003 | 001 | 002 | 002 |
PP II GG DD HH (Fulfilled already and exist in the list)
III PrD PrH Prw PrC PrP W_d W_h W_w W_c W_p DesI DNi DPi
000 004 005 000 002 009 001 003 -02 00 002 37,000 0,00 1,45
001 005 009 000 002 008 001 001 -03 00 002 30,000 5,55 0,00
```

Multi – delimited file

(b)

```
Date, Open, High, Low, close, volume, Adj close
2008-08-01,20.09,20.12,19.53,19.80,19777000,19.80
2008-06-30,21.12,21.20,20.60,20.66,17173500,20.66
2008-05-30,27.07,27.10,26.63,26.76,17754100,26.76
2008-04-30,27.17,27.78,26.76,27.41,30597400,27.41
Date, Open, High, Low, close, volume, Adj close
2008-08-01,20.09,20.12,19.53,19.80, 19777000,19.80
2008-06-30,21.12,21.20,20.60,20.66,17173500,20.66
2008-05-30,27.07,27.10,26.63,26.76,17754100,26.76
2008-04-30,27.17,27.78,26.76,27.41,30597400,27.41
Date, Open, High, Low, close, Volume, Adj close
2008-08-01,20.09,20.12,19.53,19.80, 19777000,19.80
2008-06-30,21.12,21.20,20.60,20.66,17173500,20.66
2008-05-30,27.07,27.10,26.63,26.76,17754100,26.76
2008-04-30,27.17,27.78,26.76,27.41,30597400,27.41
```

Header row repetition

(c)

Figure 9: Examples of ill-formed rows due to row-level inconsistencies

Nonetheless, for files containing Non-ASCII values, users have the flexibility to adapt the code to their specific needs by following a straightforward configuration process outlined in the code repository[10]. Moreover, users can seamlessly expand the grammar rules to cater to Non-ASCII letters according to their preferences. This entails simply introducing new representation classes within the pattern generation module. The instructions for this process are available in the code repository, facilitating a clear understanding of code utilization.

One main challenge we address is system efficiency. Our solution comprises several phases that require both time and memory optimization. We achieve efficiency by introducing pruning techniques to optimize the different phases of the system (see Section 3.3).

3.2 Problem Definition

We first introduce the basic concepts involved in this work to then define the notion of syntactic patterns. Following this, we provide definitions for ill-formed and well-formed rows and outline the syntactic pattern grammar.

3.2.1 Ill-Formed and Well-Formed Rows

The input to our approach is a *file*, which is composed of a number of rows. A *row* is composed of horizontally aligned cells, where each cell belongs to a column and is separated by a delimiter. A *column* is composed of vertically aligned cells across rows, where each cell contains a value (including the empty value). To distinguish ill-formed and well-formed rows, we define a *pattern schema*: one or more ordered sequences of attributes, each attribute having a syntactic pattern (defined hereafter). A pattern schema can contain more than one ordered sequence because rows even in well-structured tables might follow different patterns. For example, a column with usernames might have a different number of tokens in each row.

We now define the central notion of a syntactic pattern:

Definition 1 *Syntactic patterns are a sequence of symbols to represent characters of an input value. The production rules of Table 6 transform each input value to one or more weighted* syntactic patterns*, where weights reflect different levels of pattern abstraction.*

For example, using our syntactic pattern grammar (presented in Section 3.2.2) two of the syntactic patterns for New York City zip codes (e.g., 10001) are $\langle D\rangle\langle D\rangle\langle D\rangle\langle D\rangle\langle D\rangle$ and $\langle SEQD\rangle$. We assign a higher weight to each abstraction of the pattern inversely proportional to its abstraction level (see Figure 10); thus, the abstraction "digit" $\langle D\rangle$ is given a higher weight than "sequence of digits" $\langle SEQD\rangle$. SURAGH uses these weights to avoid generalization during its pattern selection and to prune highly abstracted patterns. However, $\langle SEQD\rangle$ and other higher-level abstractions are also crucial in the process when necessary, e.g., a column with an index number from 1 to 1 000 000; there, it is not trivial to find a small set of patterns using low-level abstractions due to the many different cell values.

In general, we expect all the rows of a standard CSV file to conform to the same schema and thus contain values with the same syntactic patterns across columns. However, non-standard CSV files may include rows with different syntactic patterns, for example, if they contain multiple tables and therefore multiple schemata, or contain metadata rows, such as table titles, footnotes, etc.

Definition 2 *A row* conforms to *a pattern schema if it has the same number of attributes as the schema, and all column values of the row conform to the corresponding column patterns of one of the attribute sequences of the pattern schema. We call such a row* well-formed *and* ill-formed *otherwise.*

We now formally define our problem as follows: *Given an input file $F = \{r_1, r_2, \ldots, r_m\}$ of m rows, automatically generate its pattern schema and use it to classify each row as ill-formed or well-formed.*

Please note that we limit our solution to *syntactically* or structurally ill-formed rows. Identifying *semantically* ill-formed rows, e.g., rows with data errors, is beyond the goal of this research.

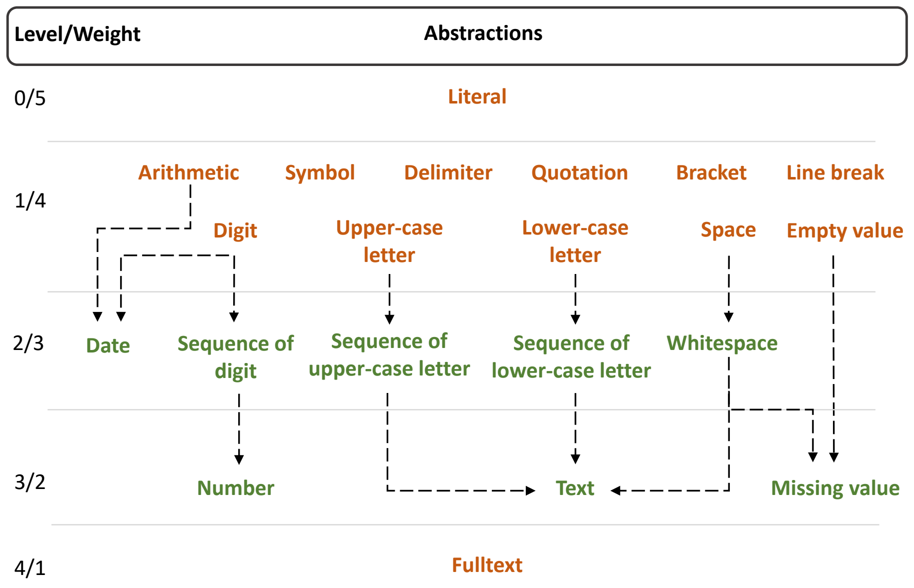


Figure 10: Abstractions dependency graph

3.2.2 A Grammar for Data Rows

To construct syntactic patterns, we define a set of production rules to transform values and call them abstractions. Table 6 lists these abstractions with their representation and also shows the associated grammar. We specify the grammar using the Extended Backus-Naur Form (EBNF) rules and notations [23]. The abstractions are of two types, namely (1) encoder and (2) aggregator. For a given value, the encoder abstractions map each character to a derived representation, while the aggregator abstractions that depend on the encoder abstractions combine representations resulting from other encoder and aggregator abstractions based on a given rule. The application of these rules is order-dependent; for example, the "sequence of upper-case letters" $\langle SEQUL \rangle$ abstraction is an aggregator abstraction that is applicable only to representations of the "upper-case letter" $\langle UL \rangle$ abstraction. Figure 10 shows the dependency graph between abstractions, where abstractions without edges can be executed in any order. The abstractions shown in orange are encoders, while those shown in green are aggregators.

For a given input file, SURAGH generates, collects, and constructs syntactic patterns at several levels; syntactic value pattern, syntactic column pattern, and syntactic row pattern each serves a purpose in the process of detecting ill-formed rows (see Section 3.3 for detailed definitions).

Table 6: Abstractions with weight and associated grammar (EBNF notation)

Weight	**Abstractions**	**Grammar**
5	Literal	Represented by the character of the field itself
4	Delimiter $\langle DEL\rangle$	"," | ";" | ":" | ?US-ASCII character 9? | "|"
	Upper-case letter $\langle UL\rangle$	"A" | "B" | "C" | "D" | "E" | "F" | ...| "Z"
	Lower-case letter $\langle LL\rangle$	"a" | "b | "c" | "d" | "e" | "f" | ...| "z"
	Digit $\langle D\rangle$	"0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
	Space $\langle S\rangle$	?US-ASCII character 32?
	Quotation ⟨”⟩	‘ " ’
	Arithmetic $\langle ARITH\rangle$	"*" | "+" | "-" | "/" | "%" | "=" | "<" | "> "
	Bracket $\langle BRKT\rangle$	"[" | "]" | "{" | "}" | "(" | ")"
	Symbol $\langle SYM\rangle$	"$" | "#" | "." | "?" | "@" | "\\" | "^" |" ‘ "| "~" | "_" | " ’ " | "&" | "!" | ‘ " ’ | "," | ";" | ":" | ?US-ASCII character 9? | "|"
	Line break $\langle LB\rangle$	?US-ASCII character 10? | ?US-ASCII character 13?
	Empty value $\langle EV\rangle$	null
3	Sequence of upper-case letters $\langle SEQUL\rangle$	$\langle UL\rangle$, {$\langle UL\rangle$}
	Sequence of lower-case letters $\langle SEQLL\rangle$	$\langle LL\rangle$, {$\langle LL\rangle$}
	Sequence of digits $\langle SEQD\rangle$	$\langle D\rangle$, {$\langle D\rangle$}
	Whitespace $\langle WS\rangle$	$\langle S\rangle$, {$\langle S\rangle$}
	Date $\langle DT\rangle$	2*$\langle D\rangle$,$\langle ARITH\rangle$, 2*$\langle D\rangle$, $\langle ARITH\rangle$, 4*$\langle D\rangle$ | 2*$\langle D\rangle$, $\langle ARITH\rangle$, 2*$\langle D\rangle$, $\langle ARITH\rangle$, 2 *$\langle D\rangle$ | 4*$\langle D\rangle$, $\langle ARITH\rangle$, 2*$\langle D\rangle$, $\langle ARITH\rangle$, 2*$\langle D\rangle$ | $\langle D\rangle$, $\langle ARITH\rangle$, $\langle D\rangle$, $\langle ARITH\rangle$, 2 *$\langle D\rangle$ | $\langle D\rangle$, $\langle ARITH\rangle$, $\langle D\rangle$, $\langle ARITH\rangle$, 4*$\langle D\rangle$ | $\langle D\rangle$, $\langle ARITH\rangle$, 2*$\langle D\rangle$, $\langle ARITH\rangle$,2 *$\langle D\rangle$| $\langle D\rangle$, $\langle ARITH\rangle$, 2*$\langle D\rangle$, $\langle ARITH\rangle$, 4*$\langle D\rangle$ | 4*$\langle D\rangle$, $\langle ARITH\rangle$, $\langle D\rangle$, $\langle ARITH\rangle$, $\langle D\rangle$ | 4*$\langle D\rangle$, $\langle ARITH\rangle$, 2*$\langle D\rangle$, $\langle ARITH\rangle$, $\langle D\rangle$ | 4*$\langle D\rangle$, $\langle ARITH\rangle$, $\langle D\rangle$, $\langle ARITH\rangle$, 2*$\langle D\rangle$
2	Number $\langle NUM\rangle$	"+" | "-", $\langle D\rangle$ | $\langle SEQD\rangle$ OR ["+" | "-"], $\langle D\rangle$ | $\langle SEQD\rangle$, ("," | "."), $\langle D\rangle$ | $\langle SEQD\rangle$, { ("," | "."), $\langle D\rangle$ | $\langle SEQD\rangle$}
	Text $\langle TXT\rangle$	$\langle UL\rangle$ |$\langle SEQUL\rangle$| $\langle LL\rangle$ | $\langle SEQLL\rangle$, ($\langle UL\rangle$ | $\langle SEQUL\rangle$| $\langle LL\rangle$ | $\langle SEQLL\rangle$| $\langle S\rangle$| $\langle WS\rangle$ |" - "|"_" | " ’ " | "\\" | "/" | "." | "&"), { $\langle UL\rangle$ | $\langle SEQUL\rangle$| $\langle LL\rangle$ | $\langle SEQLL\rangle$| $\langle S\rangle$| $\langle WS\rangle$ | "-" | "_" | " ’ " | "\\" | "/" | "." | "&"}
	Missing value $\langle MV\rangle$	$\langle EV\rangle$| | $\langle S\rangle$| | $\langle WS\rangle$| | ?US-ASCII character 9? | "Null" | "null" | "na" | "n/a" | "NA" | "N/A" | "NaN" | "nan" | "None" | "NONE" | ‘""’
1	Fulltext $\langle FTXT\rangle$	Very long string, e.g., cellValue.size() > 50

3.3 The SURAGH System

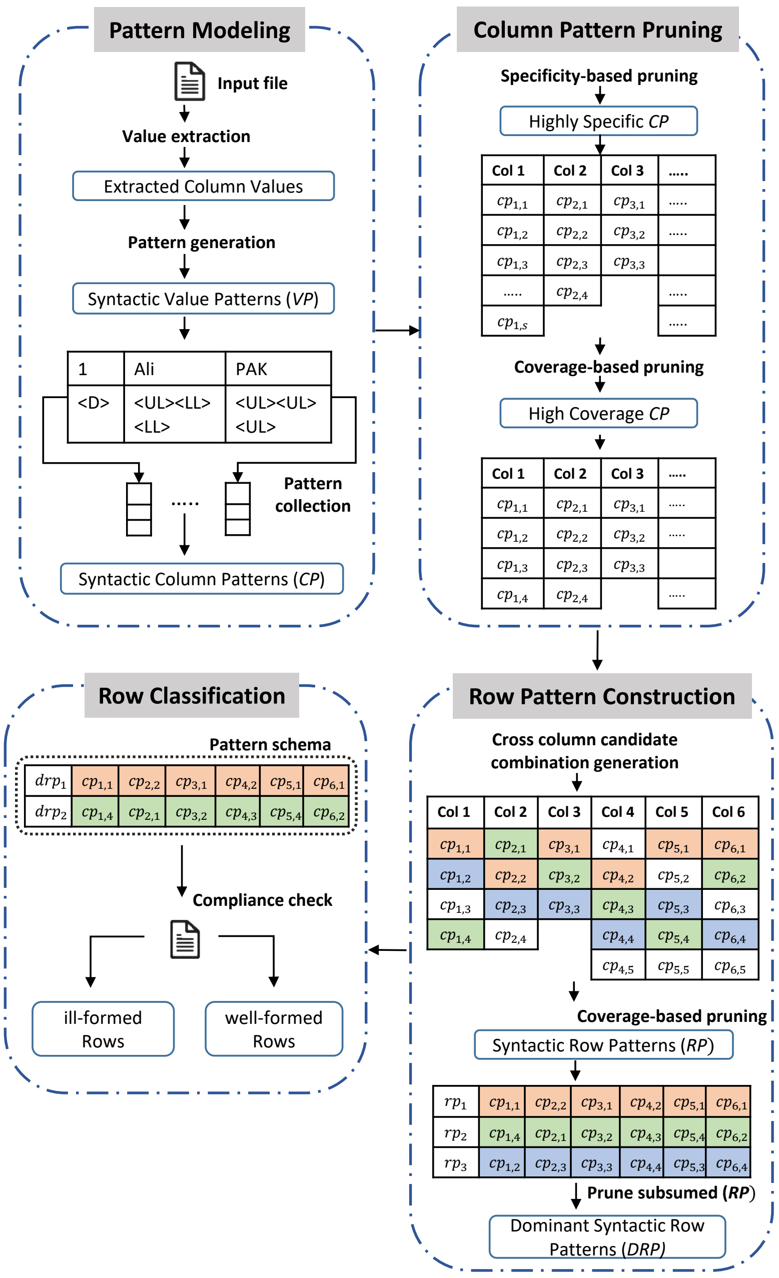


Figure 11: The workflow of SURAGH

In this section, we describe the workflow of SURAGH as depicted in Figure 11. Given an input file, SURAGH returns a list of indices of ill- and well-formed rows along with the dominant syntactic row patterns as its pattern schema. The workflow consists of four phases, and we discuss their functionalities in the following sections.

3.3.1 Pattern Modeling

In this phase, SURAGH first leverages dialect detection to extract values and then applies our abstraction grammar to these values to generate syntactic value patterns. Then, value patterns are collected to form a set of syntactic column patterns.

Value extraction

To extract column values, we must first detect the dialect of the file. The dialect of a file specifies a set of characters that define the structure of a file. A CSV file dialect consists of a delimiter, a quote character, a quote escape character, and a row separator. Dialect detection is a well-known problem in academia [20, 29, 100] and industry: SURAGH leverages the univocity parser[11] to identify a file's dialect and extract the individual column values. This dialect determines the scope of each column.

Pattern generation

As mentioned, an ill-formed row contains inconsistencies at column- and/or row-level. Some of these inconsistencies result in different row structures, such as incorrectly delimited rows, while others result in inconsistent column values, such as values not compliant to a schema, making an ill-formed row different from a well-formed row. Extracting the intended pattern schema helps recognize these ill-formed rows. For pattern schema detection, SURAGH generates one or more syntactic value patterns for each individual cell value using a syntactic pattern grammar that utilizes abstractions.

Definition 3 (Syntactic value pattern) *Given a cell value, a syntactic value pattern vp is a sequence of literals or abstractions from that value. We represent abstractions with their acronyms, where each abstraction represents a single character or a group of characters. The set of syntactic value patterns for a cell value is denoted with VP.*

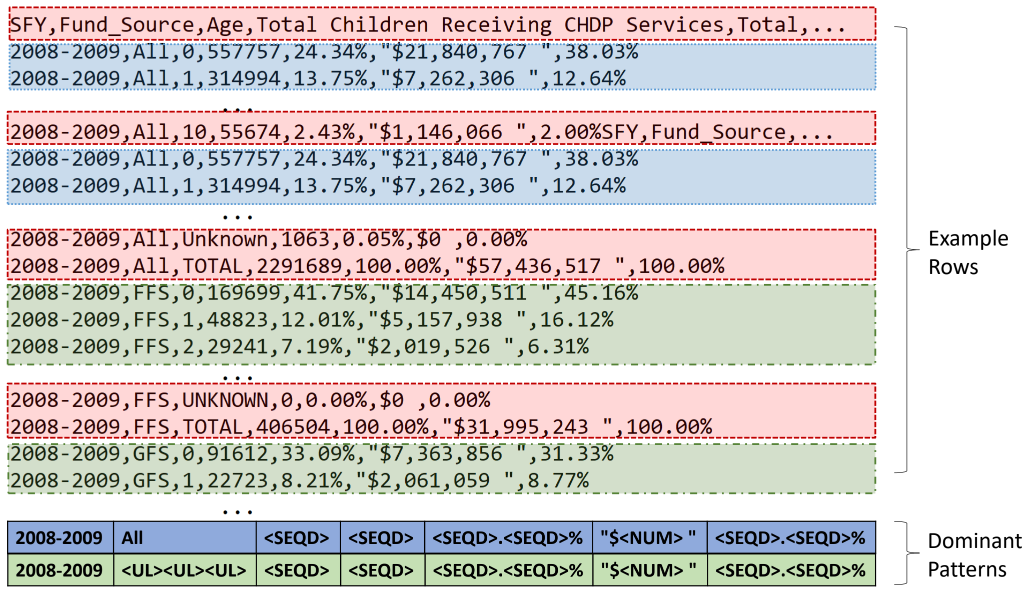
[11]`https://github.com/uniVocity/univocity-parsers`

Algorithm 1: Syntactic Value Pattern Generation

```
Input  : $c$: input column
         $A$: the set of abstractions
         $G$: the abstractions dependency graph
Output: $L_{VP}$ = the list of $VP$ that represents values in $c$
1  $L_{VP} \leftarrow []$
2  foreach $string \in c$ do
3      $VP \leftarrow \{\}$
4      $A \leftarrow$ GETABSTRACTIONS($string$)
5      $PS \leftarrow$ PRUNEABSTRACTIONCOMBINATIONS$\Big($POWERSET$(A), G\Big)$
6      foreach $set \in PS$ do
7          $vp \leftarrow string$
8          foreach $abstraction \in set$ do
9              $vp \leftarrow$ EXECUTEABSTRACTIONS($abstraction$ , $vp$)
10         end
11         if $vp \notin VP$ then
12             $VP \leftarrow VP \cup \{vp\}$
13         end
14     end
15     $L_{VP}.add(VP)$
16 end
17 return $L_{VP}$
```

The process of generating syntactic value patterns is presented in Algorithm 1. SURAGH uses abstractions to map each character in each value of a column and generates a set of value patterns. Formally, let $F = \{r_1, r_2, \ldots, r_n\}$ be the input file of n rows. Let $C = \{c_1, c_2, \ldots, c_m\}$ be the set of m columns. For each value in c_i, the algorithm first accesses the relevant abstractions $\{a_1, a_2, \ldots, a_t\}$ based on input value characters (line 4). Then, the algorithm creates a power set of the selected abstractions to capture all possible combinations of abstractions and uses them to generate all possible value patterns (line 5). After creating a power set, the algorithm prunes the subsets that violate the encoder-aggregator abstraction dependencies shown in Figure 10 (Page 38). For example, the algorithm prunes a subset if it contains "date" $\langle DT \rangle$ abstraction before "digit" $\langle D \rangle$ in the execution order. For each remaining subset, it maps each cell value character to its derived representation from the abstractions to generate the set of value patterns $VP = \{vp_1, vp_2, \ldots, vp_u\}$ (lines 6-14). It repeats the same process for all column values and populates a list that contains the set of value patterns for each value in c_i (line 15). For example, Figure 13(a) lists the set of value patterns for the cell value "`All`" in the input file

F in Figure 12[12].



```
SFY,Fund_Source,Age,Total Children Receiving CHDP Services,Total,...
2008-2009,All,0,557757,24.34%,"$21,840,767 ",38.03%
2008-2009,All,1,314994,13.75%,"$7,262,306 ",12.64%
                    ...
2008-2009,All,10,55674,2.43%,"$1,146,066 ",2.00%SFY,Fund_Source,...
2008-2009,All,0,557757,24.34%,"$21,840,767 ",38.03%
2008-2009,All,1,314994,13.75%,"$7,262,306 ",12.64%
                    ...
2008-2009,All,Unknown,1063,0.05%,$0 ,0.00%
2008-2009,All,TOTAL,2291689,100.00%,"$57,436,517 ",100.00%
2008-2009,FFS,0,169699,41.75%,"$14,450,511 ",45.16%
2008-2009,FFS,1,48823,12.01%,"$5,157,938 ",16.12%
2008-2009,FFS,2,29241,7.19%,"$2,019,526 ",6.31%
                    ...
2008-2009,FFS,UNKNOWN,0,0.00%,$0 ,0.00%
2008-2009,FFS,TOTAL,406504,100.00%,"$31,995,243 ",100.00%
2008-2009,GFS,0,91612,33.09%,"$7,363,856 ",31.33%
2008-2009,GFS,1,22723,8.21%,"$2,061,059 ",8.77%
                    ...
```

2008-2009	All	<SEQD>	<SEQD>	<SEQD>.<SEQD>%	"$<NUM> "	<SEQD>.<SEQD>%
2008-2009	<UL><UL><UL>	<SEQD>	<SEQD>	<SEQD>.<SEQD>%	"$<NUM> "	<SEQD>.<SEQD>%

Figure 12: Selected rows of a 100-row file[12] with dominant syntactic row patterns (shaded blue and green) and ill-formed rows (shaded red). The cell separators in the table indicate the $\langle DEL \rangle$ abstraction, which we omit due to space limitation.

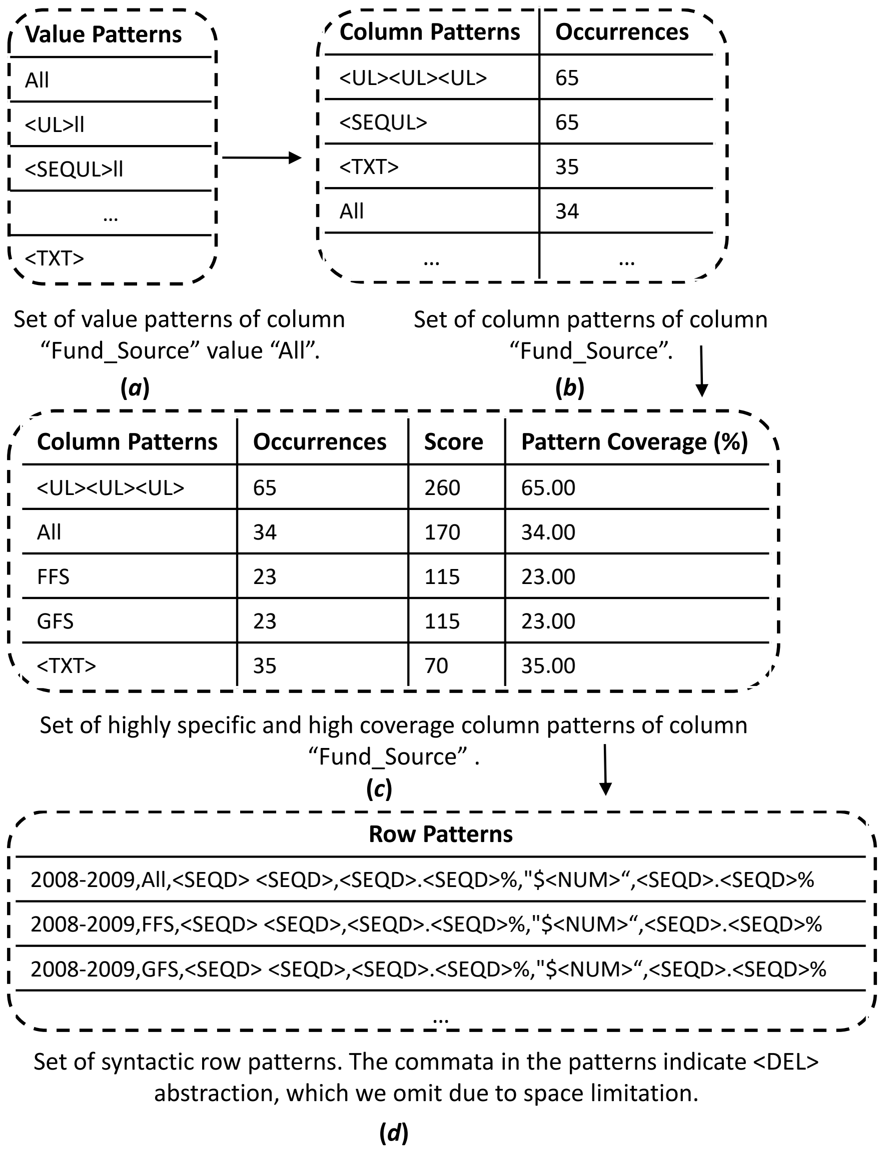
Pattern collection

SURAGH collects the syntactic value patterns generated in the previous step column by column and uses them as column patterns.

Definition 4 (Syntactic column pattern) *A syntactic column pattern cp is a syntactic value pattern that represents one or more cell values in a column. The set of syntactic column patterns for a column is denoted with CP.*

After collecting value patterns, SURAGH forms a set CP of syntactic column patterns $\{cp_1, cp_2, \ldots, cp_v\}$. This set represents a column in a file, and each pattern in this set represents one or more cell values in that column. For example, for the input file F in Figure 12, Figure 13(b) lists the set of column patterns for column "`Fund_Source`".

[12]`https://github.com/HPI-Information-Systems/SURAGH/blob/main/InputFile.csv`



Value Patterns
All
<UL>ll
<SEQUL>ll
...
<TXT>

Set of value patterns of column “Fund_Source” value “All”.

(*a*)

Column Patterns	Occurrences
<UL><UL><UL>	65
<SEQUL>	65
<TXT>	35
All	34
...	...

Set of column patterns of column “Fund_Source”.

(*b*)

Column Patterns	Occurrences	Score	Pattern Coverage (%)
<UL><UL><UL>	65	260	65.00
All	34	170	34.00
FFS	23	115	23.00
GFS	23	115	23.00
<TXT>	35	70	35.00

Set of highly specific and high coverage column patterns of column “Fund_Source” .

(*c*)

Row Patterns
2008-2009,All,<SEQD> <SEQD>,<SEQD>.<SEQD>%,"$<NUM>“,<SEQD>.<SEQD>%
2008-2009,FFS,<SEQD> <SEQD>,<SEQD>.<SEQD>%,"$<NUM>“,<SEQD>.<SEQD>%
2008-2009,GFS,<SEQD> <SEQD>,<SEQD>.<SEQD>%,"$<NUM>“,<SEQD>.<SEQD>%
...

Set of syntactic row patterns. The commata in the patterns indicate <DEL> abstraction, which we omit due to space limitation.

(*d*)

Figure 13: Value patterns, column patterns, highly specific and high coverage column patterns, and row patterns.

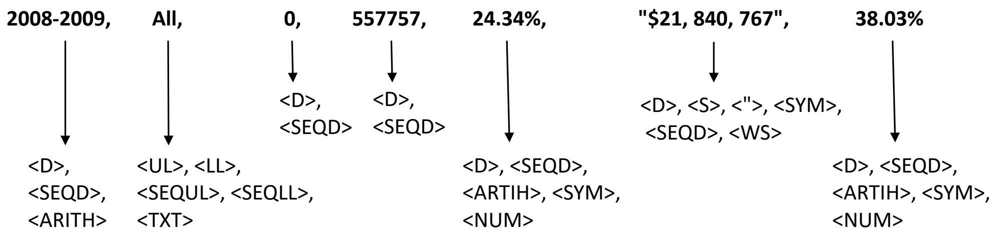

2008-2009,
All,
0,
557757,
24.34%,
"$21, 840, 767",
38.03%
<D>, <SEQD>, <ARITH>
<UL>, <LL>, <SEQUL>, <SEQLL>, <TXT>
<D>, <SEQD>
<D>, <SEQD>
<D>, <SEQD>, <ARTIH>, <SYM>, <NUM>
<D>, <S>, <">, <SYM>, <SEQD>, <WS>
<D>, <SEQD>, <ARTIH>, <SYM>, <NUM>

Figure 14: Mapping of Cell Values to abstractions (cell values are aligned with their corresponding applicable abstractions for clarity). The $\langle DEL \rangle$ abstraction, which corresponds to commata, is omitted to save space.

3.3.2 Column Pattern Pruning

During syntactic pattern generation, the algorithm aims to apply every combination of abstractions to generate all possible value patterns for representing cell values, requiring many system resources and affecting performance. We propose a cell value detection module that scans each value and uses only relevant abstractions (see Figure 14). For example, for the value "`All`" in the input file F in Figure 12, the relevant abstractions are $\{\langle UL \rangle$, $\langle LL \rangle$, $\langle SEQUL \rangle$, $\langle SEQLL \rangle$, and $\langle TXT \rangle\}$.

In addition, the dependency graph between the abstractions guides the process of reducing the number of abstraction combinations, which optimizes the time expense of the pattern generation process because the algorithm generates fewer patterns. However, not every generated pattern is suitable for the ill-formed row detection due to its generality and/or coverage.

To select the highly specific and high coverage subset of column patterns from the generated patterns, we make use of two pruning techniques: (1) specificity-based pruning and (2) coverage-based pruning. We preserve *Highly specific* and *high coverage* column patterns after applying the above pruning techniques on the given column pattern set *CP*.

Specificity-based pruning

SURAGH generates syntactic patterns with all possible abstraction combinations based on the content of the input values. These patterns represent column values from the most specific to the most generic way, depending on the pattern's abstractions. *The most specific column pattern* contains the fewest and the lowest level abstractions. In contrast, *the most generic column pattern* includes maximum and the highest level abstractions.

SURAGH aims to choose column patterns that are highly specific in a column. To do so, it applies specificity-based pruning and leverages the weighting of abstractions, and uses the relationship between encoder and aggregator abstractions to prune generic column patterns. The goals of this phase are:

- Preserve the most specific column patterns.
- Reduce the search space for optimization.

Weighting criteria: Each abstraction a_i has a weight w based on its abstraction level (see Figure 10): the higher the abstraction level, the lower the weight. SURAGH uses these weights to obtain a score for each column pattern cp. This score helps SURAGH prune column patterns by identifying the significance of these patterns in a column. To obtain a score for each column pattern, SURAGH scans through the column, counts cp's number of occurrences oc, and multiplies it by the mean value of the utilized abstractions weight:

$$Score(cp) = oc(cp) \, * \, \frac{1}{t}\sum_{i=1}^{t} w(a_i(cp)) \tag{1}$$

Consider, as an example, that the value "`male`" appeared in a column 80% of the time. To represent this value, three of the possible value patterns using the abstraction grammar are $\{$*male*, $\langle LL\rangle\langle LL\rangle\langle LL\rangle\langle LL\rangle$, and $\langle SEQLL\rangle\}$. As a user, we would like to see the value *male* itself as one of the top column patterns for this column due to its specificity and high coverage and because this pattern with the literal value itself represents the column very well. Thus, we assign the highest weight to the *literal* abstraction due to its specificity to prioritize the patterns with literals. The second-highest weighting applies to level 1 abstractions that help identify highly specific patterns when the topmost patterns are not predominantly literal values. For example, if a column contains different values distributed across the cells, e.g., a column containing the US state alpha codes, i.e., AL, AK, AZ, etc. In this case, despite the highest weighting, the literal column patterns would be ranked low due to their occurrences, so abstraction level 1 and higher abstractions would serve the purpose better. For example, two of the possible column patterns for the mentioned column are $\{\langle UL\rangle\langle UL\rangle$ and $\langle SEQUL\rangle\}$. According to our definition, we assign $\langle UL\rangle\langle UL\rangle$ a higher weight and $\langle SEQUL\rangle$ a lower weight based on their abstraction level.

Pruning generic column patterns: As mentioned earlier, not every generated pattern is suitable. For example, in Figure 13(b), the column patterns $\langle UL\rangle\langle UL\rangle$ and $\langle SEQUL\rangle$ occur the same number of times and represent the same set of values. Therefore, we prune $\langle SEQUL\rangle$ since it is just a higher abstraction of the same values in this case. Note that occurrences is not the only criterion to prune these patterns. We also check the dependency between the encoder and the aggregator abstractions of these patterns and the pattern score. In the above example, $\langle SEQUL \rangle$ is dependent on $\langle UL \rangle$, and due to its higher abstraction, it also has a lower score.

SURAGH first sorts column patterns by their score, starts with the highest-scored column pattern, compares it to all subsequent column patterns by examining their occurrences, checks encoder-aggregator dependency on those that meet the occurrence criterion, and removes those with higher abstractions. In this way, SURAGH removes the most generic patterns, as they are just another representation of specific patterns. However, there are other cases where a column contains many different values, and it is not possible to represent values with low-level abstractions due to a maximum variation, such as columns with addresses or website URLs. Therefore, despite the lower weight, due to the number of occurrences of higher-level abstractions, the top patterns of the columns SURAGH returns are generic. Nevertheless, in the case of a column comprising web addresses, patterns might appear generic, specific components tailored to them still exist, such as the common protocol prefix for URLs followed by their unique addresses.

Coverage-based pruning

In coverage-based pruning, we introduce a coverage threshold to prune column patterns at the column-level (column coverage threshold β) and column pattern combinations at the row-level (row coverage threshold γ, see Section 3.3.3). Here, *Column-level pattern coverage:* means how many column values are covered by a column pattern, and *Row-level pattern coverage:* implies how many rows are covered by a combination of column patterns across all columns. In Section 3.5 we test combinations of seven different thresholds at both column- and row-level with a step size of 5, starting from 1% to 30% of the pattern coverage. For the first threshold, we use 1% since 0% pattern coverage is meaningless. In the subsequent discussion on column and row coverage thresholds, we refer to these same seven thresholds as part of the process. Here, percentage implies the coverage of column patterns at column-level and column pattern combinations at row-level. The goals of this phase are:

- Preserve high coverage column patterns (column-level).
- Preserve high coverage column pattern combinations (row-level).
- Reduce the search space for optimization.

Let β be the specified column coverage threshold; we prune all column patterns that provide column coverage less than or equal to the specified β. We use seven different thresholds for column pattern pruning. The goal is to select the set of high coverage column patterns using a global column coverage threshold.

For example, for the input file F in Figure 12, Figure 13(c) shows the highly specific and high coverage column patterns for column "`Fund_Source`" that we obtain after specificity- and coverage-based pruning, where $\beta = 20\%$. For the column coverage threshold of 25%, the remaining column patterns for the mentioned column are $\{\langle UL\rangle\langle UL\rangle\langle UL\rangle$, *All*, and $\langle TXT\rangle\}$.

3.3.3 Row Pattern Construction

In this phase, SURAGH uses the highly specific and high coverage column patterns from the column pattern pruning phase and first constructs the set *RP* of syntactic row patterns $\{rp_1, rp_2, \ldots, rp_y\}$ that represents rows. It then selects the set *DRP* of dominant syntactic row patterns $\{drp_1, drp_2, \ldots, drp_z\}$ by pruning the set of syntactic row patterns to generate the pattern schema for the input file.

Coverage-based pruning

Similar to selecting column patterns with high coverage at the column-level, the goal here is to select the combinations of column patterns with high coverage when constructing row patterns, which leads us to select high coverage row patterns.

Let γ be the specified row coverage threshold; we prune all cross column candidate combinations that provide row coverage less than or equal to the specified γ. Here, *coverage* is the number of rows corresponding to a combination of column patterns across all columns. We use the same thresholds as at the column-level to prune column patterns combinations during row pattern construction.

For example, for the input file F in Figure 12, a combination of column patterns {*2008-2009* and *FFS*} covers 23% rows for the columns {"`SFY`" and "`Fund_Source`"} (see Figure 13(c) for column "`Fund_Source`" column patterns). For the row coverage threshold of 25%, SURAGH prunes this combination and stops the cross-column combination generation process for this candidate.

Cross column candidate combination generation

For row pattern construction, SURAGH creates all combinations of column patterns across all columns while pruning based on coverage (γ). Furthermore, the algorithm generates column pattern combinations for all columns and constructs all row patterns with high coverage.

Definition 5 (Syntactic row pattern) *A syntactic row pattern rp is an ordered sequence of (highly specific and high coverage) column patterns that represents one or more rows. The set of syntactic row patterns for a file is denoted with RP.*

The process of constructing the set of syntactic row patterns is presented in Algorithm 2. Formally, let $\{c_1, c_2, \ldots, c_m\}$ be the set of columns and let $CP_i = \{cp_{i,1}, \ldots, cp_{i,v}\}$ be the corresponding set of highly specific and high coverage column patterns *cp*s for a column c_i. SURAGH first starts comparing CP_i and CP_{i+1}, and then progressively moves towards CP_m (lines 2-13). For each pair of columns, SURAGH generates cross-column combinations by comparing each $cp \in CP_i$ with each $cp \in CP_{i+1}$ governed by coverage-based pruning (lines 4-11). *Intersection* (line 6) computes the row coverage of a specified *cp* combination, and the *coverage* check (line 7) allows *cp* combinations above the specified threshold γ. Then, *combinePattern* concatenates the *cp* combination, and stores it into a column combination set (line 8). This column combination set contains *cp* combinations across columns, and serves as new input after each iteration (line 12). The algorithm repeats the same process and creates all possible column pattern combinations until c_m and test them against the coverage threshold. To avoid the multiplicative nature of this approach, SURAGH systematically grows row patterns by one column at a time in a branch & bound fashion. It prunes those intermediate patterns that already violate the coverage threshold γ and returns the set of row patterns (line 14). For example, for the input file *F* in Figure 12, Figure 13(d) lists the set of syntactic row patterns.

Prune subsumed syntactic row pattern

The set of syntactic row patterns that SURAGH constructs may contain one or more row patterns, where one row pattern may represent all rows that are also represented by another row pattern, resulting in pattern redundancy. To avoid this redundancy, SURAGH detects and removes the row patterns that are subsumed by any other row patterns.

Definition 6 (Pattern subsumption) *A row pattern $rp_i \in RP$ is subsumed by another row pattern $rp_k \in RP$ if the set of all rows it represents is a proper subset of the set of those represented by rp_k.*

Definition 7 (Syntactic row pattern dominance) *A dominant syntactic row pattern drp is a syntactic row pattern that is not subsumed by any other syntactic row pattern. The set of dominant syntactic row patterns for a file is denoted with DRP.*

Algorithm 2: Cross Column Candidate Combination Generation

```
Input  : $\{CP_1, CP_2, \ldots, CP_n\}$, row coverage threshold $\gamma$ (see Section 3.3.2)
Output: $RP$ = set of syntactic row patterns
1  $RP \longleftarrow CP_1$                              ▷ initialization
2  foreach $CP_i$, $i > 1$ do
3      $CC \longleftarrow \{\}$                          ▷ column combinations
4      foreach $cp_r \in RP$ do
5          foreach $cp_c \in CP_i$ do
6              $PC \longleftarrow \text{INTERSECTION}(cp_r, cp_c)$        ▷ pattern coverage
7              if $coverage(PC) > \gamma$ then
8                  $CC \longleftarrow CC \cup \text{COMBINEPATTERN}(cp_r, cp_c)$
9              end
10         end
11     end
12     $RP \longleftarrow CC$
13 end
14 return $RP$
```

3.3.4 Row Classification

In the final phase of SURAGH, we utilize the constructed set of dominant syntactic row patterns from the row pattern construction phase to generate a pattern schema for the input file and classify rows based on this schema. First, SURAGH accumulates dominant row patterns obtained from the row construction phase and generates a pattern schema for the input file. Then, SURAGH checks each row whether it conforms to the generated schema.

SURAGH classifies non-conforming rows as ill-formed, whereas conforming rows as well-formed. For example, Figure 12 shows the input file, the set of dominant syntactic row patterns that form its pattern schema, the rows that comply with the schema, and the non-compliant (ill-formed) rows.

3.4 Datasets and Annotation

This section lists the datasets used in our evaluations and specification of the annotated ill- and well-formed rows.

3.4.1 Datasets

We conducted our experiments with datasets collected from five different open data sources: DataGov, Mendeley, GitHub, UKGov, NYCData. The statistics for each data source are summarized in Table 8. The files of each source have at least some column and/or row-level inconsistencies. To find files with inconsistencies, we first randomly selected files from each source and manually loaded each into an RDBMS[13]. If the data loading process aborted, we kept the file as one of our test files. Later on, we utilized the same RDBMS along with data wrangling[14] and business intelligence[15] tools to investigate where loading operations are interrupted and used this information to annotate rows in our files. To reduce the manual annotation workload, we 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 diverse set of files for each data source. The code artifacts together with datasets and annotations are publicly available[10].

The UKGov and GitHub dataset files were taken from related work on dialect detection [100]. We randomly selected 1 000 CSV files from both of these datasets and manually checked each file by loading it into the RDBMS. After reviewing each file, we collected those containing ill-formed rows, which gives us 23 and 25 diverse files from UKGov and GitHub, respectively. From the randomly selected 2 066 files, there were 418 files with ill-formed rows that we observed by loading them into the RDBMS. We then used 40 manually selected files for our experiments. Our Mendeley dataset was created by selecting files from `data.mendeley.com` where files are grouped by research projects. We crawled all 2 214 projects and selected the lexicographical first 170, which already displayed a large variety of ill-formed rows. These projects contain not only CSV files but also other formats, such as .XML, .XLSX, etc. We kept only projects with at least one CSV file and manually selected 33 files for our experiments, each file coming from a different project. NYCData dataset includes a random subset of files downloaded from `opendata.cityofnewyork.us` and was kept unseen in the development of SURAGH to ensure generalizability.

Moreover, we utilized a command line tool[16] to check whether a file is compliant with RFC 4180. The purpose is to include both standardized and non-standardized files in our datasets and emphasize that also standard files can stop the data loading process, for instance, due to values not compliant to file schema. In the presence of inconsistent rows, we manually checked each file and ensured that each selected file contained at least 50% non-empty values at both column and row-levels to avoid empty values and missing values ($\langle EV \rangle$, $\langle MV \rangle$) to dominate the pattern search.

[13]Microsoft SQL Server 2019. `https://www.microsoft.com/en-us/sql-server/sql-server-2019`
[14]Trifacta Wrangler. `https://www.trifacta.com/products/wrangler-editions/`
[15]Tableau. `https://www.tableau.com/`
[16]`https://github.com/Clever/csvlint/`

The difference in the number of rows per file is significant, as shown by the high standard deviation in Table 8, for two reasons: 1) A few of the files in our datasets are very large. 2) To ensure a diverse set of files, we selected only one file from sets of similar files, resulting in a different number of rows per file.

Table 8: Datasets: number of files (F), average number of rows (R), average ill-formed (IF) rows per file, average well-formed (WF) rows per file.

Source	**# F**	**Avg # R**	**Avg # IF**	**Avg # WF**
DataGov	40	1143.2 ± 2060.6	70.9 ± 165.9	1072.3 ± 2014.1
Mendeley	33	2688.8 ± 3664.3	56.8 ± 109.1	2632.0 ± 3636.1
GitHub	25	791.9 ± 1094.6	110.6 ± 300.8	681.4 ± 963.5
UKGov	23	1548.1 ± 3179.2	98.7 ± 202.1	1449.4 ± 3076.1
NYCData	10	2068.7 ± 3159.4	713.0 ± 1970.3	1355.7 ± 1507.8

3.4.2 Data Annotation

We annotated each row in each file as ill- or well-formed and created a ground truth of 210 550 rows across all datasets. As part of the SURAGH annotation process, we assessed each file in our datasets using the commercial tools mentioned earlier. We observed that all tools recognized the dialect for files that conformed to the RFC 4180 standard [45], but the RDBMS aborted the data loading process due to the ill-formed rows in each file. We observed different behavior with the data wrangling and business intelligence tools for standardized files: in most cases, the tools successfully load our files correctly by simply mending the cell value, such as correcting the date format type based on the common pattern in a column. However, in some cases, this automatic correction led to incorrect results, such as assigning the most generic data type (string) to a column in the case of values that do not match a schema.

We then manually identified column-level and row-level inconsistencies in these ill-formed rows. Based on these findings, we manually determined the most specific syntactic patterns for each given column in the file and manually labeled them as part of the pattern schema. Finally, we annotated each row as well-formed if it conforms to the manually labeled pattern schema and ill-formed otherwise.

In addition, for most files that do not conform to the RFC standard, the RDBMS does not recognize the correct dialect and loads all the data into one column. For the few files where the RDBMS did recognize the dialect, it aborted the data loading process due to various issues, such as incorrect quotation marks, misplaced delimiters, inconsistent number of columns, etc. While the data wrangling and business intelligence tools, due to their built-in intelligence, do recognize the correct dialect in most cases, they may still load data incorrectly: for instance, they might assign the most generic data type (string) to each column or add new columns due to an incorrect splitting, or by simply deleting rows they could not parse. A few cases where these tools do not recognize the correct dialect load all the data into one column, similar to the RDBMS.

Note that for our study, we annotated the header row in all CSV files as ill-formed: We cannot assume that all tools have a built-in feature to distinguish header rows from data rows, or at least provide a user with a feature to mark the first row as a header row so that it does not become a part of the data.

3.5 Experiments

First, we present our experimental evaluation of SURAGH, which includes the global threshold selection and its results for each dataset. Subsequently, we compare SURAGH with the current state-of-the-art syntactic pattern and line classification approaches. Finally, we conclude this section with an error analysis.

3.5.1 Performance Evaluation

As per Definition 2 (Page 37), a row in a file is ill-formed if it does not conform to the pattern schema of that file. We call a detected ill-formed row *true* if its corresponding row in the ground truth is labeled as ill-formed, and *false* if it does not. To show the effectiveness of our approach, we use the precision and recall metrics:

$$P = \frac{|\textit{true ill-formed rows detected}\,|}{|\textit{true \& false ill-formed rows detected}\,|} \tag{2}$$

$$R = \frac{|\textit{true ill-formed rows detected}\,|}{|\textit{total ill-formed rows}|} \tag{3}$$

Threshold setting evaluation

To evaluate the performance of SURAGH, we test combinations of seven different threshold values for both column- and row-levels presented in Section 3.3.2, leading to 49 experiments for each dataset. Due to varying numbers of ill-formed and well-formed rows in CSV files, selecting experiments at the row-level, where all ill-formed and well-formed rows from all CSV files in a dataset are treated equally, irrespective of their file origins, could introduce bias towards larger files when analyzing the results. To avoid this potential bias, we conducted our experiments at the file-level. We compared the set of ill-formed rows detected by SURAGH with the set of true ill-formed rows of ground truth for each CSV file.

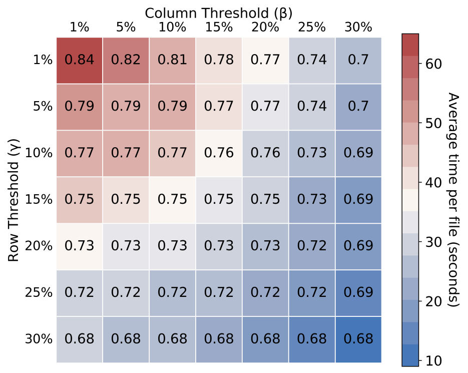
Figures 15a, 15b, and 15c show precision, recall, and F-1 scores, respectively, for each threshold-combinations where the color codes reflect the average runtimes associated with each threshold-combination. Each cell reports the average score across all files of all datasets (excluding the unseen dataset NYCData) for a combination of column- and row-level thresholds. As expected, we can observe that low thresholds lead to less pruning and, thus, to higher runtime.

For the first threshold, we use 1% since 0% pattern coverage is meaningless. We kept a maximum threshold of 30% for two reasons: 1) we achieved the highest recall by retaining only patterns with the highest coverage, 2) precision decreased due to many false positives.

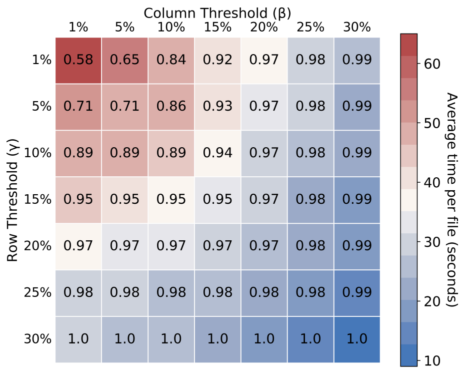
The goal of these experiments was to determine the global threshold setting (column- and row-level thresholds combination) and provide a user with a tool including a built-in threshold setting. However, users have the option to customize these thresholds according to their specific needs and preferences. Details on how to customize the code can be found in the SURAGH code repository[10].

To determine the global threshold setting, we start by computing precision and recall scores. We then calculate the F-1 score across all datasets by considering paired combinations of both thresholds. The first experiment starts with a threshold setting of 1%, where the system only prunes column patterns and column pattern combinations with coverage less than or equal to 1% at both column- and row levels. This setting generates numerous row patterns that may hinder the identification of ill-formed rows, leading to an increased number of false negatives. However, the substantial construction of row patterns minimizes the likelihood of classifying well-formed rows as ill-formed, resulting in fewer false positives. Consequently, this threshold setting achieves the highest precision score but at the expense of the lowest recall score, as illustrated in Figure 15a and Figure 15b. Similarly, Figure 15c illustrates that this threshold setting yields the lowest F-1 score.

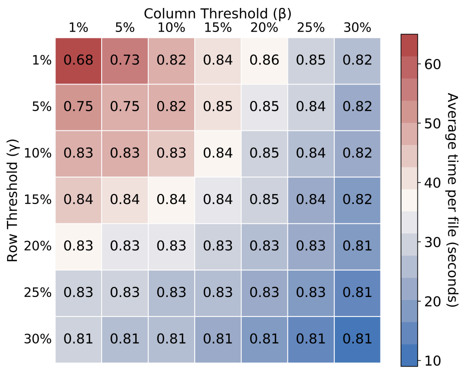
In the subsequent experiment, we kept the row threshold constant while increasing the step size for the column threshold, calculating precision and recall scores accordingly. After completing the initial seven experiments, we proceeded by increasing the step size for the row threshold and repeating the same procedure. After completing all the experiments, we observed a common pattern in both precision and recall scores. With each step size increase in the threshold setting, we noted a decrease in precision score. Consequently, this results in the classification of many well-formed rows as ill-formed, leading to an increase in false



(a) Precision score



(b) Recall score



(c) F-1 score

Figure 15: Average precision, average recall, and average F-1 score across all datasets. The color code shows runtimes (same for all three plots).

positives. However, with each step size increase in the threshold setting, we observed an increase in the recall score. This phenomenon occurs because fewer rows conform to one of the selected patterns, causing the system to classify more rows as ill-formed, resulting in fewer false negatives.

Given this observed pattern, we leverage the F-1 score to determine the global threshold setting. As mentioned earlier, we noticed that when aiming for the highest precision score, it often resulted in a lower recall score, and vice versa, especially at the extreme threshold settings. This trade-off is evident in the F-1 score, as depicted in Figure 15c.

The F-1 score shows an increasing trend at the beginning and a peak in the middle of the threshold setting, which then decreases as the threshold step size increases. The decreasing trend of the F-1 score from the middle of the threshold setting to both ends is due to the lowest recall at the minimum threshold setting and the lowest precision at the maximum threshold setting. The threshold setting (β = 20%, γ = 1%) is the one with the highest F-1 score, which we also tested on unseen data. The promising results indicate its suitability as a recommended global setting.

We observed that in our datasets, most column patterns covering $\leq$ 20% of cell values, and column pattern combinations covering $\leq$ 1% of rows, refer to ill-formed rows. Since we first apply the column-level threshold β, the larger step size of β perfectly balances with the smaller step size of the row threshold γ, as most of the patterns that might refer to ill-formed rows are already pruned after applying the formal threshold. This behavior is reflected in Figures 15a and 15b where precision decreases by about 7%, but the recall score increases significantly by about 39% between $\beta = 1\%$ to 20% when $\gamma = 1\%$.

For a more detailed examination of performance across individual datasets, Figures 16 to 19 present precision and recall measures specific to each dataset.

Unseen dataset

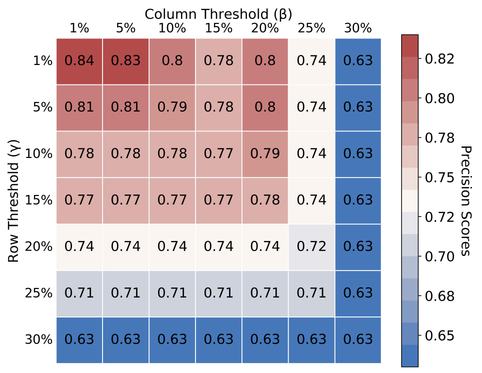
To test the generalizability of SURAGH, we applied our approach to the unseen dataset, NYCData. The results obtained by SURAGH, utilizing the global threshold setting determined from the other four datasets, are presented in Table 9. Similar to the performance on all the other datasets, SURAGH achieved a recall of over 90% and a precision of 70%.

Table 9: Unseen dataset results

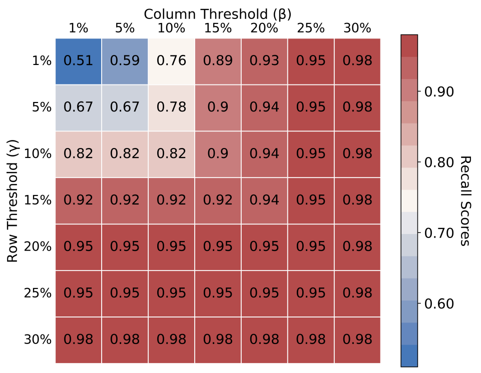
Source	**# files**	**# rows**	**Precision**	**Recall**	**F-1**
NYCData	10	20 687	0.70	0.95	0.81

System efficiency

SURAGH achieved an average classification time of 9 ms per row with the global threshold setting. All experiments are performed on a 4-core Intel Core i7 2.3G CPU with 16GB RAM. Figure 20 shows the runtime of SURAGH for the files in our dataset, including the six synthetically generated ones. While we observe a quadratic scalability overall, variance is quite high due to the quite different pattern complexity of individual files. The runtime increase is expected due to the increase in cross-column combinations in Algorithm 2, which depends on #columns and #rows; adding more columns leads to more cross-column

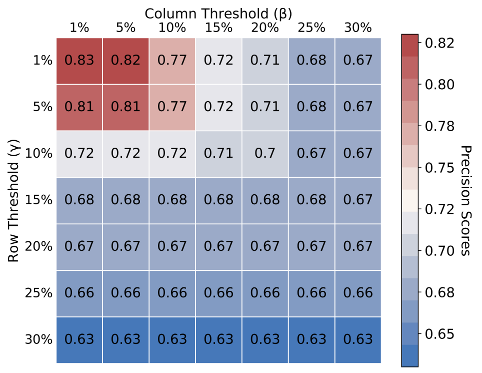


(a) Precision score

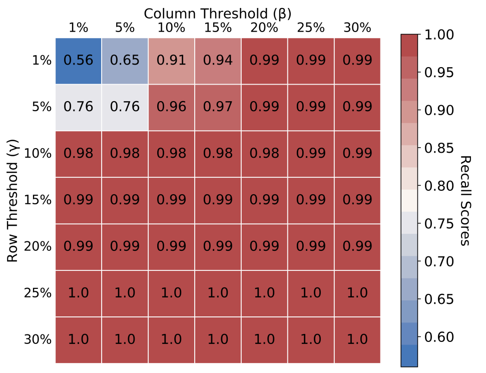


(b) Recall score

Figure 16: Average precision and recall scores for all DataGov dataset files.

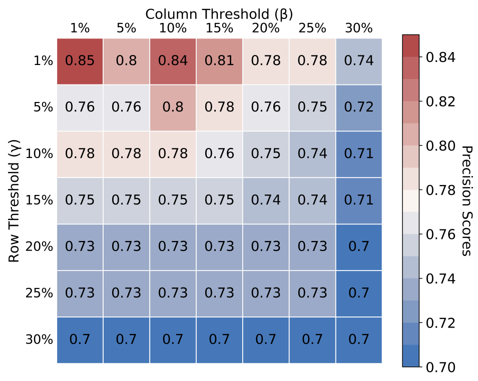


(a) Precision score

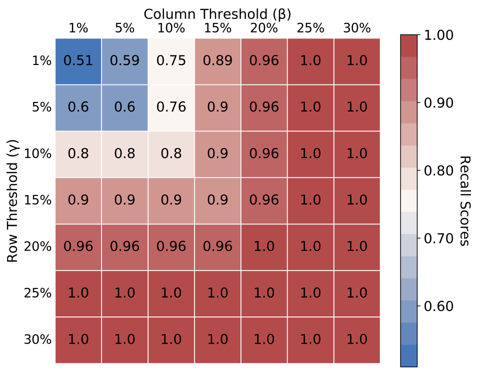


(b) Recall score

Figure 17: Average precision and recall scores for all UKGov dataset files.

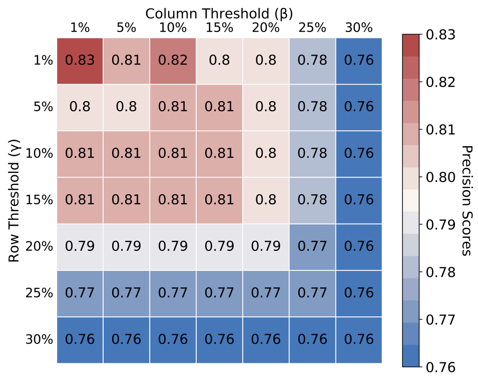


(a) Precision score

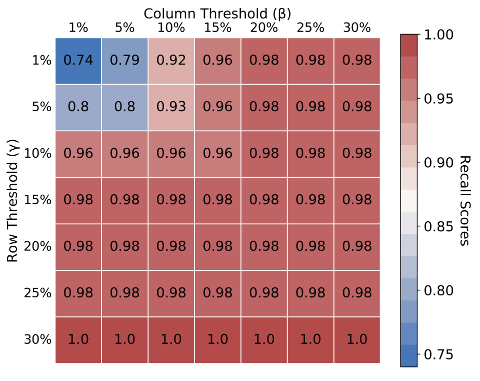


(b) Recall score

Figure 18: Average precision and recall scores for all GitHub dataset files.



(a) Precision score



(b) Recall score

Figure 19: Average precision and recall scores for all Mendeley dataset files.

combinations. Adding more rows can dominate the runtime as these combinations are integrated to construct row patterns, where #rows can influence the row coverage threshold, resulting in less pruning and thus more computation time.

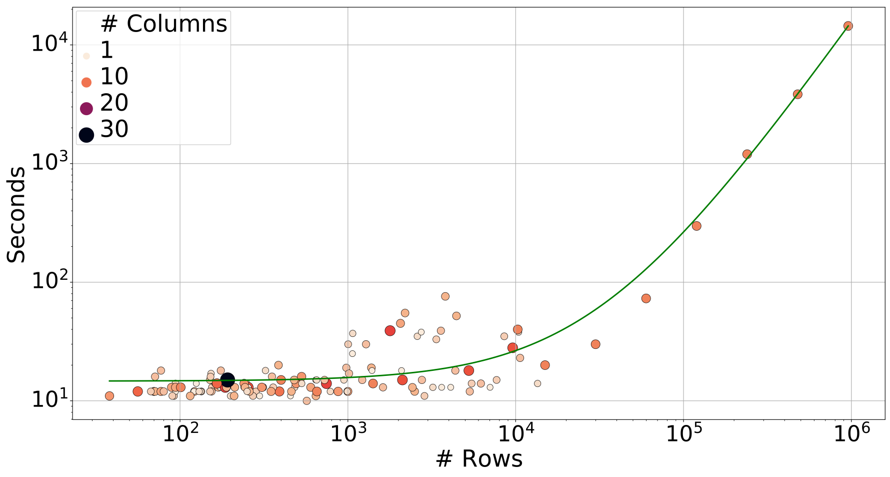


Figure 20: Ill-formed row detection efficiency, with a fitted quadratic curve. The last six files were obtained by extending existing files with duplicate rows.

3.5.2 Comparative Analysis

There is no other approach dealing specifically with the detection of ill-formed rows at the syntactic level. However, FAHES is a syntactic pattern-based approach to detect disguised missing values (DMVs) [79], with which we can simulate our task. The authors categorize DMVs into the following cases: (1) out of range data values, e.g., missing values disguised with a negative value for an attribute with positive values; (2) outliers, e.g., missing values disguised as very large values; (3) strings with repeated characters, e.g., replacing a phone number with 5555555555; (4) values with non-conforming data types, e.g., disguising the missing strings with numerical values; (5) valid values that are randomly distributed within the range of the data. The proposed approach uses a syntactic outlier detection module for Cases 1, 3, and 4 to detect those DMVs that are syntactic outliers or contain special patterns. The approach also uses a numerical outlier detection module for Cases 1 and 2, and follows missing-completely-at-random (MCAR) or missing-at-random (MAR) models [6, 61] for Case 5. For our experiments, we use the FAHES demo [80][17].

[17]`https://github.com/daqcri/Fahes_Demo/`

To compare FAHES with our approach, we consider rows that, according to FAHES, contain disguised missing values as ill-formed, otherwise as well-formed. Since our approach is broader, the files in our datasets include not only the inconsistencies that FAHES deals with but many others, so not every file could be parsed by FAHES during our experiments. Of the files that FAHES successfully parsed, we excluded those with only header and missing values as inconsistencies because FAHES does not consider these cases. Table 10 shows the average precision, average recall, and F-1 score obtained by FAHES and SURAGH with the global threshold setting. The low FAHES scores can be attributed to several factors. Primarily, the tool's design focuses on identifying frequently occurring outliers, assuming that infrequent missing values have minimal impact on analytics. Consequently, the system applies a 10% threshold to remove less frequent values. Additionally, after applying the distinct formula, it is required that the resultant patterns cover at least 1% of the total distinct values. This strict criterion erroneously classified valid but infrequent patterns as errors, thereby adversely impacting precision. Furthermore, the tool's approach of discarding infrequent cases significantly contributed to its low recall. Note that we set the precision score to 1 when FAHES returned no DMVs.

Table 10: FAHES comparison overview

Source	**# files**	**# files used**	**FAHES [79]**			**SURAGH**		
			P	**R**	**F-1**	**P**	**R**	**F-1**
DataGov	40	20	0.35	0.27	0.30	0.87	0.89	0.88
Mendeley	33	3	0.00	0.00	0.00	1.00	1.00	1.00
GitHub	25	11	0.70	0.22	0.33	0.80	0.92	0.86
UKGov	23	19	0.37	0.10	0.16	0.84	0.99	0.91
NYCData	10	8	0.63	0.06	0.11	0.81	0.93	0.87

Another system to which we compared our approach is the multi-class random forest classifier approach STRUDEL for CSV file row classification [49]. STRUDEL divides rows into six semantic classes: metadata, group, header, data, derived, and notes. The approach uses a set of features: content, context, and computational features to model the individual classes to classify rows in a CSV file. STRUDEL is a supervised learning approach, so for our experiments, we trained on the datasets available at the STRUDEL project page[18] and tested on the datasets listed in Section 3.4.1. We chose the STRUDEL datasets for training, because they contain a larger set of files than the datasets we used for SURAGH, and the authors (the author of this thesis is a coauthor) tested their approach on out-of-domain datasets. Moreover, we used the same parameter settings as [49], except for the dialect detection tool, which we replaced with the univocity parser.

[18]`https://hpi.de/naumann/s/strudel`

Table 11: STRUDEL comparison overview

Source	# files	# rows	STRUDEL [49]			SURAGH		
			P	R	F-1	P	R	F-1
DataGov	40	45 728	0.98	0.21	0.34	0.80	0.93	0.86
Mendeley	33	88 729	0.97	0.42	0.58	0.80	0.98	0.88
GitHub	25	19 799	0.93	0.26	0.40	0.78	0.96	0.86
UKGov	23	35 607	0.90	0.22	0.35	0.71	0.99	0.83
NYCData	10	20 687	1.00	0.04	0.08	0.70	0.95	0.81

To compare STRUDEL with our approach, we consider the rows classified as *data* by STRUDEL to be well-formed and the remaining five classes as ill-formed. Table 11 shows the average precision, average recall, and F-1 score obtained by STRUDEL and SURAGH with the global threshold setting. STRUDEL's recall is lower for two main reasons: (i) If there are errors in the data block, STRUDEL cannot detect them due to the scope of the project, as it does not further distinguish erroneous data rows from correct data rows. (ii) In our file collection, some files have metadata and footnote rows with the same number of fields as data rows, making it difficult for STRUDEL to distinguish these non-data rows from data rows that authors also reported in their paper. Note that we set the precision score to 1 when STRUDEL returned all rows as data, thus not misclassifying any outlier-row.

3.5.3 Error Analysis

Not all ill-formed rows exhibit distinct patterns within a file. Conversely, a well-formed row might adhere to a unique pattern, which can lead to confusion for SURAGH, causing it to misclassify it as ill-formed. In the following, we present an analysis of specific detection errors made by SURAGH.

False positive cases

Our global threshold settings achieved an average precision of 76% across all datasets. To investigate instances of false positives in all datasets, we conducted a manual examination of each file. During this analysis, we identified scenarios where SURAGH does not accurately classify a row when a correct pattern is less frequent.

For instance, consider a column that primarily stores employee names, with most names consisting of two tokens (first name and last name). Consequently, the top patterns for this column predominantly consist of two-word combinations. In cases where an employee's name has three tokens, SURAGH may erroneously classify the row as ill-formed, leading to false positives.

Another instance occurs in columns containing values with long decimal numbers, such as 0.0347414562 and $7.6389273017e-005$. Data-driven systems can correctly identify and load these values by recognizing their data type. Therefore, we categorize both patterns as well-formed in our annotations. However, because very few values are represented in scientific notation with 'e', SURAGH prunes this pattern, resulting in false positives.

False negative cases

Our global threshold settings achieved an average recall of 96% across all datasets. However, there are some instances of false negatives.

In cases where columns contain only non-textual values, making it difficult to distinguish the header row from the data row, e.g., numeric headers, such as year, date, etc. Consequently, SURAGH fails to classify the header row as ill-formed, resulting in a false negative.

Another scenario leading to false negatives is when the ill-formed row value patterns are highly frequent, often due to the presence of numerous problematic rows. This is particularly evident in rows with many missing values. It is important to note that we only categorize rows as inconsistent if the file contains non-uniform empty/null values, e.g., a column that contains empty/null values at arbitrary locations. Consequently, the global threshold setting considers these patterns as well-formed, preventing SURAGH from accurately classifying these rows as ill-formed.

3.6 Related Work

Detecting ill-formed rows in CSV files is a novel research problem. However, to improve the user experience in processing data, related work has introduced approaches for implementing automated data preprocessing pipelines. Our research can complement these existing approaches.

Dialect detection: The initial step in extracting data from CSV files involves determining the file's dialect, including the delimiter, quote, and escape characters. Various approaches have been put forward to address this challenge [20, 29, 100]. Döhmen et al. introduced a CSV parsing method that concurrently generates multiple hypotheses regarding the dialect and the table structure. These hypotheses are then ranked based on the quality features of the resulting table [20]. Ge et al. proposed a finite state machine for CSV files to discern the correct dialect [29]. Van den Burg et al. introduced a Python-based approach, CLEVERCSV to detect the dialect of CSV files by leveraging row and type patterns [100]. The authors use pattern scores to favor frequent and long row patterns to find the most suitable dialect. In contrast, our approach relies on the more readily available and adaptable univocity dialect detection system.

File structure detection: Discovering the structure of CSV files is a challenging task that requires distinguishing amongst rows of different types, e.g., data, metadata, header, group header, aggregation, and footnote rows. To address this challenge, several solutions have been introduced in the literature [5, 12, 49]. Among these, Jiang et al. proposed the state-of-the-art STRUDEL approach [49], which classifies CSV file rows into six semantic classes based on three types of features: content, context, and computational features. In Section 3.5.2, we conducted a comprehensive comparative analysis between SURAGH and the STRUDEL approach.

Error detection: Detecting data quality issues has gained significant recognition within the research community [3, 40]. Several techniques for error detection have been proposed [13, 38, 42, 63, 79]. Among these techniques, Qahtan et al. introduced FAHES, a syntactic pattern approach to detect disguised missing values as outliers [79]. The authors classify disguised missing values into five categories and attempt to detect them using various modules, including outlier detection, missing-at-random, or missing-completely-at-random. In Section 3.5.2, we conducted a comparative analysis of their approach with our proposed technique.

Data transformation: Data transformation has posed a persistent research challenge, with one prominent solution being the "transform-by-example" (TBE) method. This method enables users to provide input and output examples, allowing the system to search for consistent programs [4, 36, 50, 51]. While the TBE method has made notable progress, its reliance on users to provide paired input and output examples still presents limitations in its usability. To address this challenge, Jin et al. introduced the "transformation-by-pattern" (TBP) paradigm for data transformation, which operates solely based on input/output data patterns, without the need for paired examples [52]. This approach involves utilizing a source pattern, a target pattern, and a transformation program. Their way of representing the data patterns is similar to our abstraction representation, so our automatically detected patterns might serve as input to this approach in discovering TBP programs.

3.7 Conclusion

Raw data contain ill-formed rows, which have inconsistencies at both column- and row-levels, preventing data from being (correctly) loaded into downstream applications. Identifying such rows in advance reduces errors and user effort in the event of a data loading problem.

In this chapter, we introduced SURAGH– a system to automatically detect ill-formed rows and helps expert users by generating a pattern schema for a file so that they can predetermine how messy the data are and how much data preparation is needed. SURAGH takes an input file from a user and then generates, collects, and constructs syntactic patterns at several levels that help classify ill-formed rows in a file. Additionally, we utilized pruning techniques and implemented weighting criteria and coverage thresholds to select highly specific and high coverage syntactic patterns, optimizing our solution. We collected real-world datasets containing data errors at the syntactic level and performed extensive empirical evaluations. Our approach achieved high precision and recall rates in detecting ill-formed rows across various datasets, including previously unseen data.

After identifying ill-formed rows within a file, the next intriguing challenge is to transform these rows into a standardized format, thereby assisting users in their data preparation efforts. In the following chapter, we introduce our system, TASHEEH, which addresses this transformation task employing syntactic pattern grammar.

Chapter 4

TASHEEH: A STRUCTURAL ERROR CORRECTION SYSTEM

In Chapter 3, we discussed how open data portals contain a plethora of data files, with comma-separated value (CSV) files being particularly popular with users and businesses due to their flexible standard. However, this flexibility comes with much responsibility for data consumers, as many of these files contain various structural problems. With the SURAGH system, we performed classification based on the frequent patterns found within the file and detected erroneous rows. Following the identification of errors (ill-formed rows), the subsequent phase of rectification poses a distinct challenge, as identifying errors differs significantly from correcting them.

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. Additionally, not every ill-formed row requires correction, as some may not contain any data. For instance, rows may be empty or contain metadata, such as preambles, aggregations, or footnotes. Accurately distinguishing between ill-formed rows that require correction and those that should either be removed or stored separately without any transformation presents another challenge in the quest for automation.

In this chapter, we present TASHEEH[19], an integrated system that merges with the SURAGH system. TASHEEH, in collaboration with SURAGH, enhances the identification of erroneous rows by distinguishing between ill-formed rows containing data (*wanted* rows) and non-data rows (*unwanted* rows). Most significantly, it automates the structure standardization within *wanted* rows, ensuring they conform to a consistent format based on the structure observed in well-formed rows.

[19]TASHEEH (TAS-HEEH) is an Urdu word that means correction or rectification.

Our work in this chapter is based on our publication [35], and 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[20].
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.

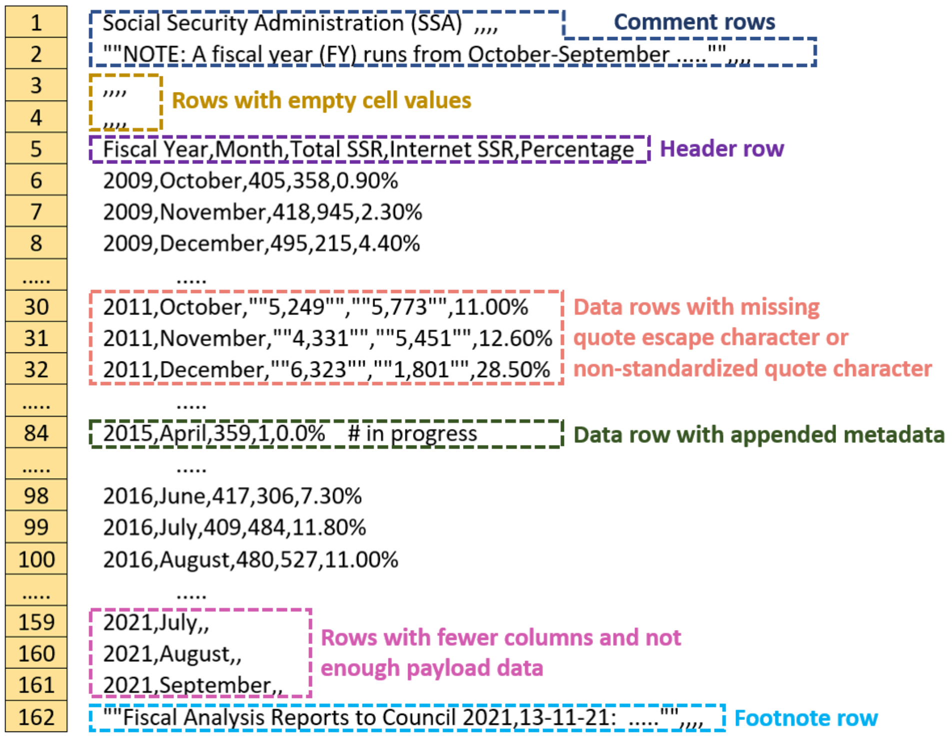
The rest of the chapter is organized as follows: Section 4.1 delves into the underlying rationale of our use-case, highlighting the structural challenges using an example file. Section 4.2 lays the foundation by presenting formal definitions and outlining the problem statement. Section 4.3 illustrates the workflow of TASHEEH and presents the processes of classifying and transforming ill-formed rows. Section 4.4 presents the experimental evaluation of TASHEEH and Section 4.5 presents related work. Finally, Section 4.6 concludes our study.

4.1 Anomalous Row Structures

Recall from Chapter 3, among other challenges, detecting and cleaning "ill-formed" rows in CSV files are difficult problems. Figure 21 shows an example of a raw CSV file taken from a government data portal. We highlight groups of ill-formed rows with different inconsistencies. Among these rows, our goal is to automatically detect those that contain data, which we call *wanted rows* (see Section 4.2 for a formal definition), and automatically clean them by repairing their structural inconsistencies.

To *detect* ill-formed rows, we abstract rows into structural patterns based on a syntactic pattern grammar, using our error-detection system SURAGH. We extend the use of the syntactic pattern grammar with TASHEEH. The goal of TASHEEH is to improve the classification of ill-formed rows by recognizing wanted and unwanted ill-formed rows, as well as automatically *clean* wanted rows structure.

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



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

In the following, we discuss structural challenges using the example file shown in Figure 21.

Example 4 *The file in Figure 21 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 [45] 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, or as* `"""5,249"""`*, as per Rule 2. 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 [95].

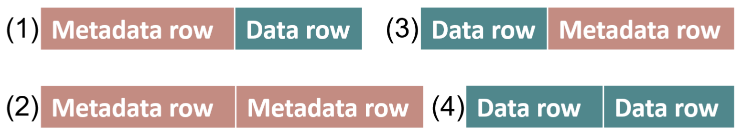

(1) Metadata row Data row
(3) Data row Metadata row
(2) Metadata row Metadata row
(4) Data row Data row

Figure 22: Examples of how data and metadata can appear in the same rows. In our file collection, we observed only combinations (1) and (3).

Example 5 *Another example of structural inconsistency in the file of Figure 21 (row 84) is data and metadata appearing in the same row. We also observed this inconsistency appear in rows in the opposite order as data next-to metadata (see Figure 22). In both combinations, metadata appear mainly in the form of comments, where users leave notes for later 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* new-line 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[21], 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.

4.2 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

[21]`https://www.mendeley.com/`

data from the same row. Formalizing these concepts, we define wanted and unwanted rows as follows:

Definition 8 *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 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 data.

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 the previous chapter using SURAGH, which employs a pattern-based approach [33]. 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 23). 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 23, the output of TASHEEH’s classification process is merged with the results of SURAGH to minimize visual clutter.

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 preprocessing pipeline, leveraging data cleaning tools and libraries [4, 31, 36, 48, 51, 52, 86, 92, 93, 96] that are specifically designed to perform such transformations.

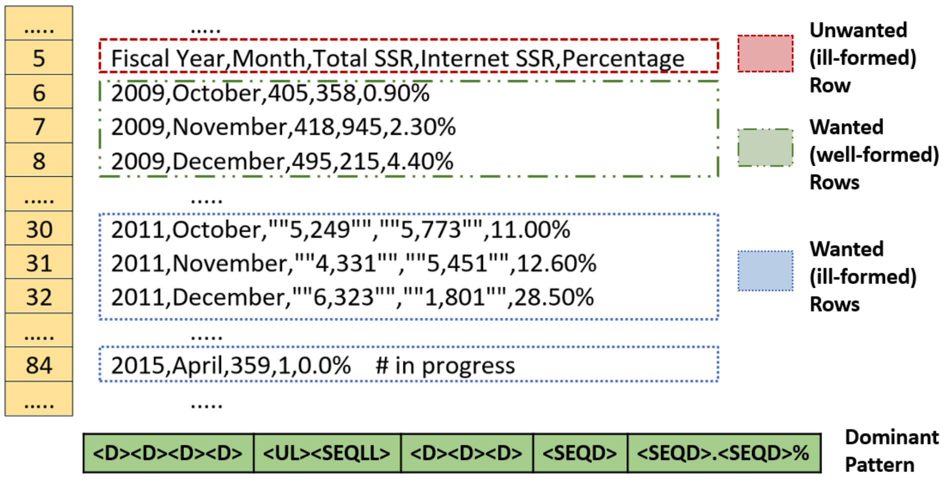

.....
5
6
7
8
.....
30
31
32
.....
84
.....
Fiscal Year,Month,Total SSR,Internet SSR,Percentage
2009,October,405,358,0.90%
2009,November,418,945,2.30%
2009,December,495,215,4.40%
2011,October,""5,249"",""5,773"",11.00%
2011,November,""4,331"",""5,451"",12.60%
2011,December,""6,323"",""1,801"",28.50%
2015,April,359,1,0.0%   # in progress
Unwanted (ill-formed) Row
Wanted (well-formed) Rows
Wanted (ill-formed) Rows
<D><D><D><D>
<UL><SEQLL>
<D><D><D>
<SEQD>
<SEQD>.<SEQD>%
Dominant Pattern

Figure 23: Selected rows of the CSV file of Figure 21 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.

4.3 The TASHEEH System

TASHEEH performs in three phases (see Figure 24). In the first phase, it first uses SURAGH to classify input file rows as ill-formed or well-formed using *dominant* row patterns ($\mathscr{P}_d$). Then, it runs SURAGH incrementally for ill-formed rows to obtain row patterns specifically for those rows; we call these patterns *potential* row patterns ($\mathscr{P}_p$): 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 4.3.1 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 [30]. This pattern sequence alignment helps TASHEEH determine the extent to which ill-formed rows differ structurally from well-formed rows. Section 4.3.2 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 4.3.3 explains the details.

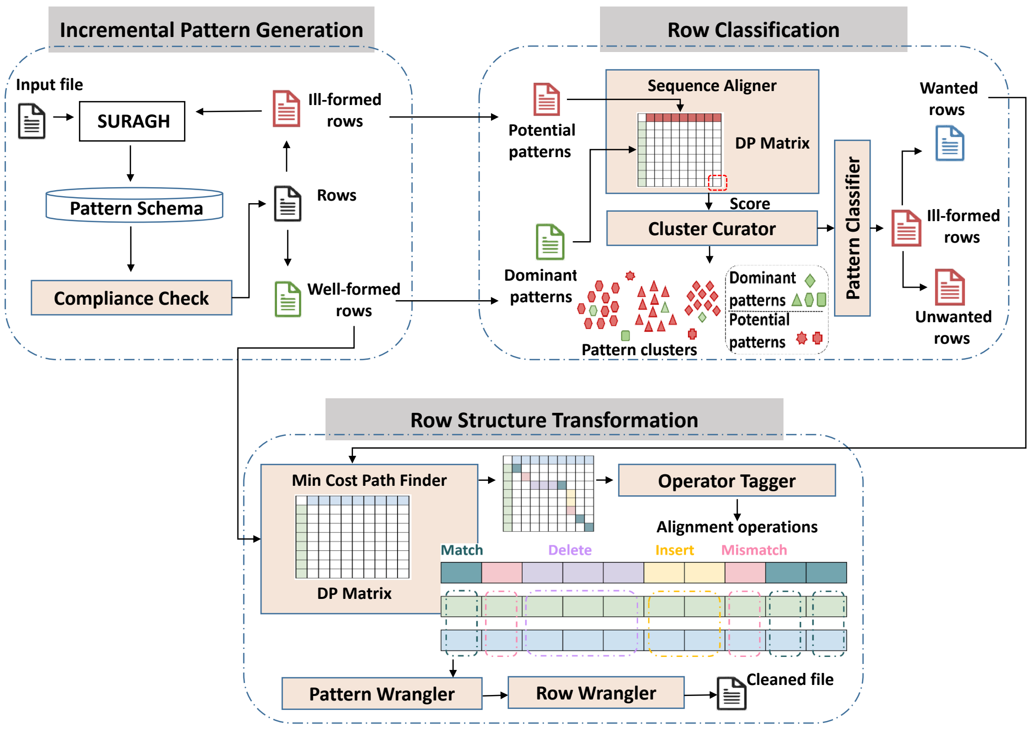


Figure 24: The workflow of TASHEEH

4.3.1 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 reassessing rows previously labeled as ill-formed. As a result, the definition of well-formed dynamically adapts with each iteration, guided by the remaining rows that have yet to be classified. For example, for the input file in Figure 21, three of the potential row patterns along with the dominant row pattern are shown in Table 12.

Table 12: Example set of dominant and potential row patterns (aligned by cell separators that represent delimiters); $\mathscr{P}_d$ corresponds to well-formed rows (Figure 23 $\rightarrow$ rows 6-8), $\mathscr{P}_{p1}$ corresponds to an unwanted row (Figure 23 $\rightarrow$ row 5), $\mathscr{P}_{p2}$ corresponds to wanted rows (Figure 23 $\rightarrow$ rows 30-32), and $\mathscr{P}_{p3}$ corresponds to a wanted row (Figure 23 $\rightarrow$ row 84).

Dominant Row Pattern:

$\mathscr{P}_d$	$\langle D\rangle\langle D\rangle\langle D\rangle\langle D\rangle$	$\langle UL\rangle\langle SEQLL\rangle$	$\langle D\rangle\langle D\rangle\langle D\rangle$	$\langle SEQD\rangle$	$\langle SEQD\rangle$. $\langle SEQD\rangle$%	

Potential Row Patterns:

$\mathscr{P}_{p1}$	`Fiscal Year`	`Month`	`Total SSR`	`Internet SSR`	`Percentage`		
$\mathscr{P}_{p2}$	`2011`	$\langle UL\rangle\langle SEQLL\rangle$	""$\langle D\rangle$	$\langle D\rangle\langle D\rangle\langle D\rangle$ ""	""$\langle D\rangle$	$\langle D\rangle\langle D\rangle\langle D\rangle$ ""	$\langle SEQD\rangle$. $\langle SEQD\rangle$%
$\mathscr{P}_{p3}$	`2015`	$\langle UL\rangle\langle SEQLL\rangle$	`359`	`1`	`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*.

4.3.2 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.

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 [30]. Sequence alignment frameworks arrange the characters of two sequences to maximize the number of matching characters [70], calculating the similarity between the sequences using dynamic programming [7].

Like edit distance frameworks for string matching [104], 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 $\mathscr{P}_d$ and a potential pattern $\mathscr{P}_p$ from Table 12. To calculate their distance, our distance-based alignment framework generates $\mathscr{P}'_d$ and $\mathscr{P}'_p$ with the same number of column patterns so that they can be aligned. To this end, we append column patterns using a special gap character to the shorter of the two sequences. We implemented our framework using a dynamic programming approach to find the alignment with the lowest distance, similar to other sequence alignment distances [70]. To find an alignment between the two patterns $\mathscr{P}_d$, $\mathscr{P}_p$, we instantiate a matrix $\mathscr{M}$ where the position at element i, j represents the minimum cost to transform $\mathscr{P}_p[0,\ldots,j]$ into $\mathscr{P}_d[0,\ldots,i]$. The matrix is initialized with the element at the index $[0,0]$, and then all other costs are filled using Equation (4):

$$\mathscr{M}(\mathscr{P}_d,\mathscr{P}_p)[i][j] = \min\begin{cases} \mathscr{M}[i-1][j]+1, \\ \mathscr{M}[i][j-1]+1, \\ \mathscr{M}[i-1][j-1]+ \\ \quad \mathscr{D}(\mathscr{P}_d[i-1],\mathscr{P}_p[j-1]) \end{cases} \tag{4}$$

Here, the cost of the insertion and deletion are calculated as 1 (first and second lines of Equation (4)). To determine the cost between individual column patterns, we define a pattern distance function $\mathscr{D}$ by enumerating four possible cases, which are summarized in Equation (5). 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 $\langle EV \rangle$, their distance is 1.

3. If α contains the delimiter $\langle DEL \rangle$ 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].

$$\mathscr{D}(\alpha,\beta)=\begin{cases}0, & \text{if } \alpha=\beta\\ 1, & \text{if } \alpha \in \{\text{‘-’},\langle EV\rangle\} \text{ or } \beta \in \{\text{‘-’},\langle EV\rangle\}\\ 1, & \text{if } \alpha=\langle DEL\rangle \text{ and } \beta \notin \{\text{‘-’},\langle EV\rangle\,\langle DEL\rangle\}\\ 1 - & \dfrac{\sum\limits_{i=l,d,s} min(|\alpha_i|,|\beta_i|)}{max(|\alpha|,|\beta|)}, \text{if } \alpha,\beta \notin \{\text{‘-’},\langle EV\rangle,\langle DEL\rangle\}\end{cases} \tag{5}$$

The pattern distance formula is inspired by the string-by-string alignment approach [36], which we adapted to define the distance function $\mathscr{D}$ between syntactic column patterns (α,β) using abstractions [33]. We consider the typical three groups of abstraction classes: letters “*l*”, digits “*d*” and symbols “*s*” (see details in [33]). 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:

$$\mathscr{D}(\mathscr{P}_d,\mathscr{P}_p)=\min_{\mathscr{P}'_d,\mathscr{P}'_p}\frac{1}{|\mathscr{P}'_p|}\sum_{k=1}^{|\mathscr{P}'_p|}\mathscr{D}(\mathscr{P}'_d[k],\mathscr{P}'_p[k]) \tag{6}$$

Here, $\mathscr{P}_d$ and $\mathscr{P}_p$ are the original input row patterns, while $\mathscr{P}'_d$ and $\mathscr{P}'_p$ are the padded row patterns to obtain the same number of columns.

Example 6 *Table 13 demonstrates the pattern-by-pattern distance scores between the dominant pattern $\mathscr{P}_d$ and potential pattern $\mathscr{P}_{p3}$ from Table 12, where the column pattern distance scores were determined through the use of Equation (5). The overall row pattern distance was determined as* $(0+0+0+0+0+0+0+0+0.90)/9=0.10$ *by applying Equation (6).*

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”. Additional details, including information about the global threshold setting, can be found the in the following sections.

Table 13: Column pattern-wise distance scores between row patterns $\mathscr{P}_d$ (Figure 23 $\rightarrow$ rows 6-8) and $\mathscr{P}_{p3}$ (Figure 23 $\rightarrow$ row 84).

Aligned Pattern Pairs		**Distance Score**	**Freq.**
$\mathscr{P}_d$ column patterns	$\mathscr{P}_{p3}$ 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

Cluster curator

In cases with only one dominant pattern, such as in the example file shown in Figure 23 (Page 70), 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 by the cluster curator, and the combination with the lowest distance score is selected.

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 4.4.2 we determined that a threshold value $\theta = 0.3$ yielded the highest F-1 score.

4.3.3 Row Structure Transformation

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

Table 14: 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.

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.

Pattern transformation algebra

Table 14 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.

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 $\mathscr{M}$ from the previous phase. First, we construct a graph on $\mathscr{M}$ 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.

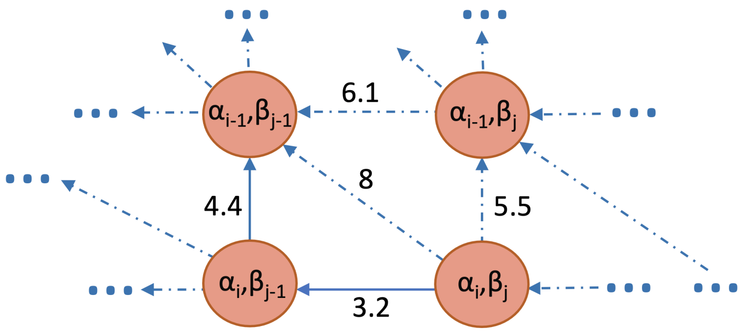


Figure 25: Weighted graph for minimum cost path finder

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 25. 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 15. 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 15). 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 [19] 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

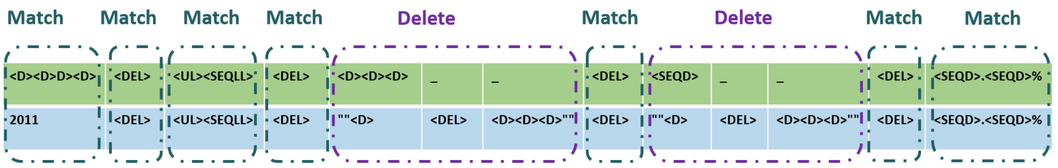


Figure 26: Minimum cost edit path alignment between row patterns $\mathscr{P}_d$ (Figure 23 $\rightarrow$ rows 6-8) and $\mathscr{P}_{p2}$ (Figure 23 $\rightarrow$ rows 30-32) together with marked transformation directions.

Table 15: 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*	<u>Ignore</u>
Mismatch	*Diagonal*	<u>Drop</u>, Extract, Ignore, Replace, Re-delimit, Re-quote, Re-escape, Re-line
Insert	*Vertical*	<u>Pad</u>, Permute
Delete	*Horizontal*	<u>Merge</u>, Drop, Extract, Move, Re-delimit, Re-quote, Re-escape, Re-line

pattern and a wanted pattern, as shown in Figure 26. The figure provides a visual representation of the aligned row patterns ($\mathscr{P}_d$, $\mathscr{P}_{p2}$) 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.

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 3). Once all transformations are complete, it applies the preferred transformations from the queue to the corresponding actual data rows.

Algorithm 3: Pattern Wrangler

```
Input: Dominant pattern $\mathscr{P}_d$, Potential pattern $\mathscr{P}_p$, alignment $A$ between $\mathscr{P}_d, \mathscr{P}_p$
Output: List of Transformations $T$
1 $T \leftarrow []$
2 foreach $0 \leq i \leq |\mathscr{P}_p|$ do
3     $transformations \leftarrow \textsc{ApplicableTransformations}(\mathscr{P}_d[i], \mathscr{P}_p[i], A[i])$
4     $T_c \leftarrow \textsc{GenerateCombinations}(transformations)$
5     $T \leftarrow T \cup \arg\min_{c \in T_c} \Big( \textsc{Distance}\big(\mathscr{P}_d[i],\ c(\mathscr{P}_p[i])\big)\Big)$
6 end
7 return $T$
```

In the following, we explain the alignment operators listed in Table 15 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 26, 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 13 (Page 75) 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 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 4). Note that the system does not impute actual data values.

Delete. If a pattern of the wanted rows has more column patterns than a dominant pattern, the alignment framework inserts gaps into the dominant pattern and marks column patterns

Algorithm 4: Pattern Padding

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

with the "delete" alignment operator in the edit path. The presence of such deletions can be attributed to inconsistencies caused by shifted values 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 26 (Page 78), 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 [45] specifications as described in Section 4.1 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.

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.

4.4 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. In addition, we compare TASHEEH with the SRFN system, which addresses shifted values in CSV files. Finally, we conclude this section with a runtime analysis and a usability case study.

4.4.1 Datasets and Annotation

We extended the datasets previously employed in SURAGH system by incorporating additional files from DataGov, Mendeley, GitHub, and UKGov. The statistics for each of these data sources are summarized in Table 16.

Table 16: 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 manually determined the most specific dominant row pattern(s) for each file, annotated each row as ill-formed or well-formed and wanted or unwanted, and created a ground truth of 200 351 rows across all datasets. Also, we 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[20].

The manual cleaning process involved examining each file's "wanted" rows, with necessary corrections or transformations made to ensure the transformed rows complied with that file's manually labeled dominant row patterns.

4.4.2 Classification Performance Evaluation

This section evaluates TASHEEH's classification (wanted and unwanted rows) component, including finding the best distance threshold to compare patterns.

In accordance with Definition 8 (Page 69), 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.

To assess the effectiveness of the classification component in TASHEEH, it is essential to determine the best distance score threshold, given the substantial differences in distance scores across the various patterns. 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 27. 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.

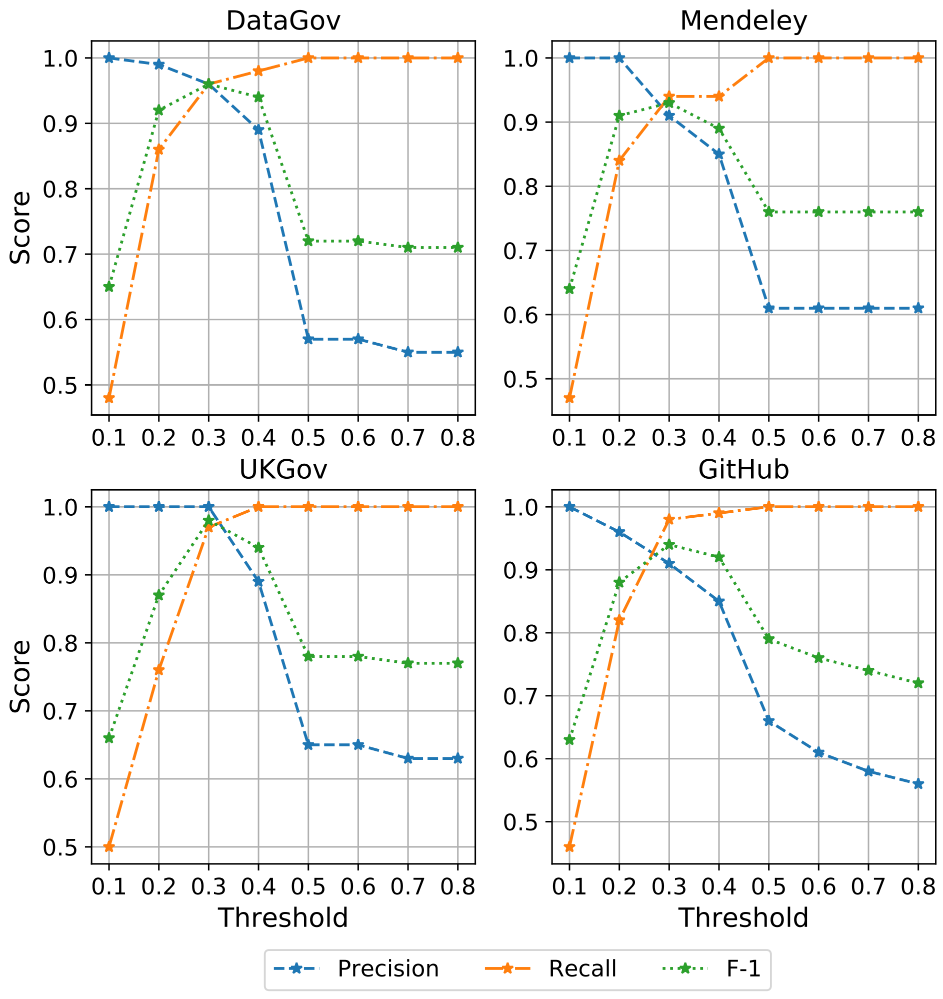


Figure 27: Precision, recall and F-1 measures at different distance score threshold values

The BASELINE classifier was inspired by the ad-hoc approach TABULAR [2], 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 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[22] due to inconsistencies were considered unwanted.

[22]Using `on_bad_lines = 'skip'`

Table 17: Row classification comparison overview

Source	# files	# rows	BASELINE			PANDAS [85]			PYTHEAS [12]			STRUDEL [49]			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**

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 in CSV files [12]. It uses a weighted set of rules to determine whether a row is data or non-data, and the binary results are used to determine the table-top/bottom boundaries. After detecting potential table headers through the search of common header patterns, PYTHEAS applies a set of additional heuristic rules to determine the row classes: context, header, subheader, data, and notes. As PYTHEAS is a supervised learning approach, the authors provided a pre-trained model, 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 and the remaining four classes to be unwanted.

STRUDEL is a multi-class random forest classifier designed for CSV file row classification [49]. In the previous chapter, we conducted a comparison between SURAGH and STRUDEL, as detailed in Section 3.5.2. Here, we provide a concise overview of STRUDEL's approach. It categorizes rows into six semantic classes, metadata, group, header, data, derived, and notes, based on content, context, and computational features. We trained STRUDEL on the datasets available at the STRUDEL project page[23] and tested on the datasets listed in Section 4.4.1. We chose to use the datasets employed by STRUDEL for training because they comprise a large number of files, and the authors (the author of this thesis is a coauthor) of STRUDEL tested their approach on out-of-domain datasets. To ensure comparability with STRUDEL, we kept the same parameter settings as described in [49], with the exception of using the univocity parser[24] instead of STRUDEL's dialect detection tool [100]. 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 17 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.

[23]`https://hpi.de/naumann/s/strudel`
[24]`https://github.com/uniVocity/univocity-parsers`

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.

4.4.3 Transformation Performance Evaluation

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

$$A = \frac{|\textit{correctly cleaned ill-formed rows}|}{|\textit{total ill-formed rows}|} \tag{7}$$

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 discuss in Section 4.5, 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 15 (Page 78).

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 28, 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.

Ill-formed rows transformation evaluation

The evaluation results in Figure 28 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.

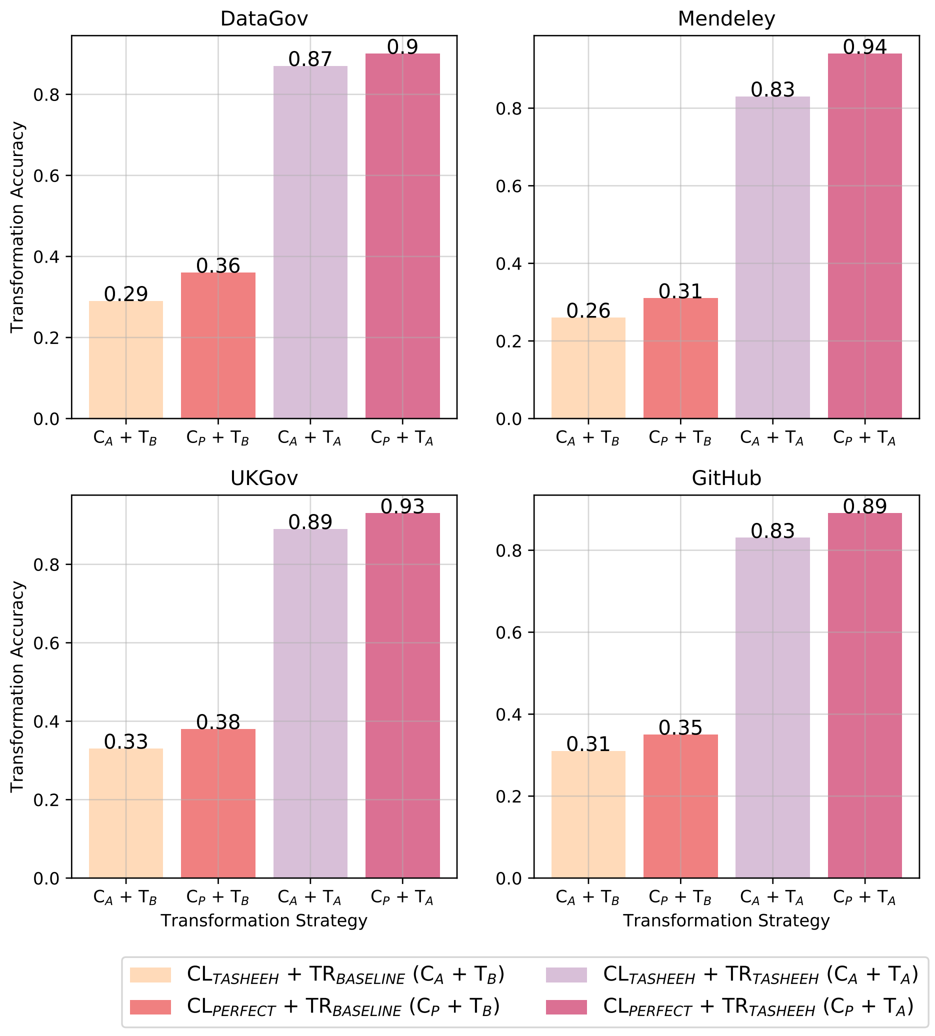


Figure 28: BASELINE and TASHEEH transformation effectiveness with TASHEEH and PERFECT row classifiers

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 number formatting of values, 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. This highlights 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.

SRFN - TASHEEH comparison

SRNF is the only other system that addresses a specific type of structural inconsistencies in data rows by "swapping repair using a fixed set of neighbors" [95]. 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[25]. 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.

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 [66]) revealed several challenges. (1) *Prompt engineering*: The performance of

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

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 [99], 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 greatly depending on the specific task at hand [109]. 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.

4.4.4 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. Figures 29 and 30 depict the runtime of TASHEEH's classification and transformation processes on the files within our dataset.

Figure 31 shows the overall runtime of TASHEEH on the files in our dataset. 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.

4.4.5 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 4.4.1. 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

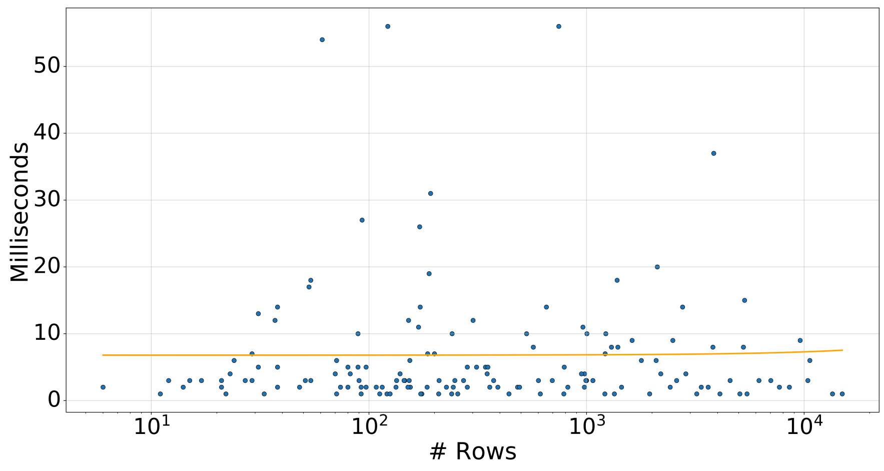


Figure 29: TASHEEH's classification efficiency

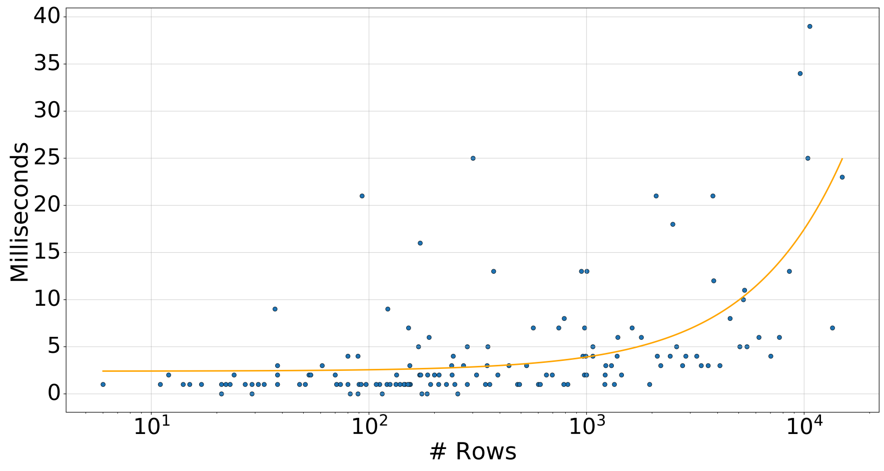


Figure 30: TASHEEH's transformation efficiency

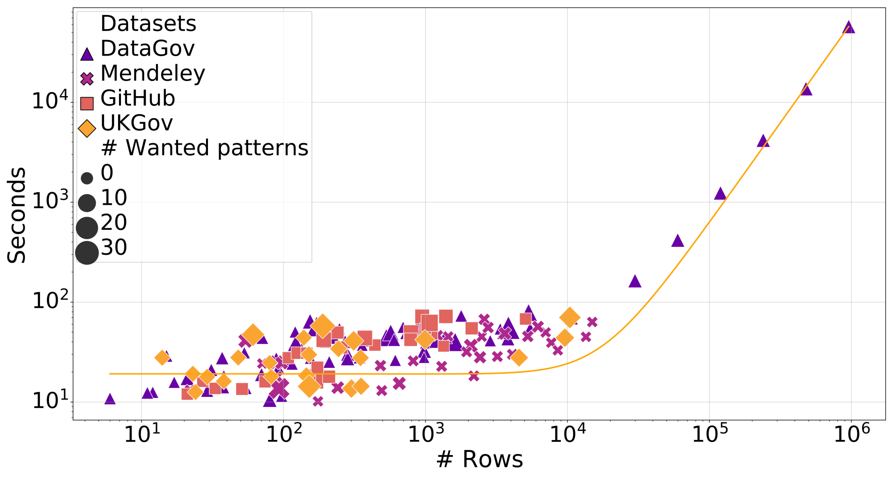


Figure 31: TASHEEH classification and transformation efficiency together with SURAGH, with a fitted quadratic curve. The last six files were obtained by extending existing files with duplicate rows. The size of the marks indicates the number of wanted patterns within the file, i.e., indirectly its degree of inconsistency.

the file-wise time taken for completion and the accuracy of the cleaning rows with the same measure of Section 4.4.

Figure 32 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 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.

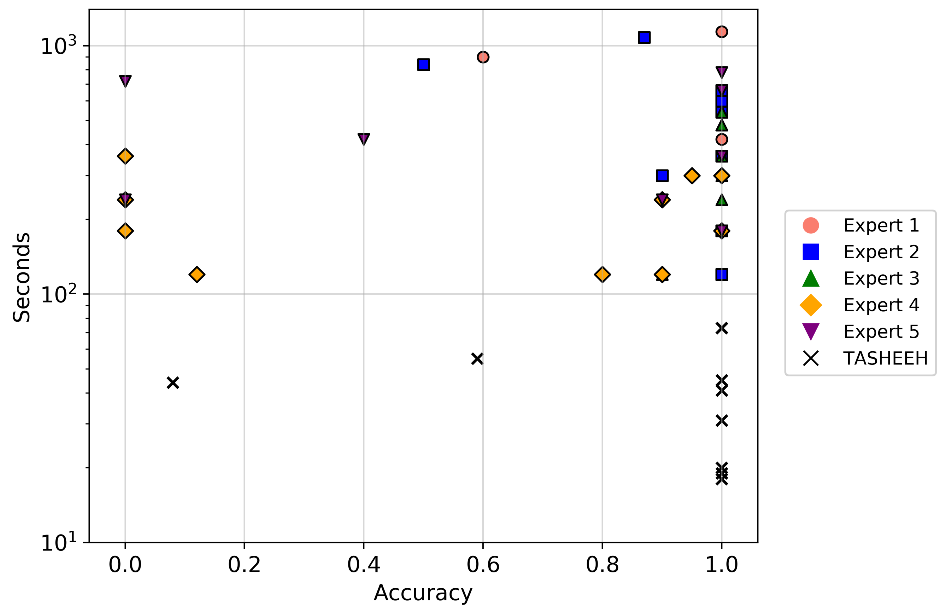


Figure 32: Comparison of manual cleaning with TASHEEH

4.5 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 for table extraction, error detection and correction in structured 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 [12, 14, 21, 26, 55, 57]. In particular, TEGRA [14] (for web lists), TABLESENSE [21] (for spreadsheets), and PYTHEAS [12] (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 4.4.2). 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 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 [64]. 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 [5, 32, 49, 56, 76]. Among these approaches, Jiang et al. proposed the state-of-the-art STRUDEL approach [49]: 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 4.4.2).

File structure preparation: In the context of non-standardized CSV files, various structural inconsistencies can arise when processing data using different tools or parsers [103]. Among these, one notable issue is the occurrence of shifted column values. Sun et al. introduced the SRFN system [95] 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 4.4.3), we compared 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 [81], 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 4.4.3 we provided a brief overview of our experience utilizing these models to address structural inconsistencies.

Despite a dearth of prior work on correcting structural issues in CSV files, many downstream tasks rely on having a structurally sound CSV file as input. In the following, we discuss the related research on these use cases.

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 [4, 36, 37, 50, 51, 91]. 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 [3, 40]. Numerous error detection techniques have been proposed [13, 38, 42, 43, 63, 79, 105], as well as error correction methods [15, 27, 52, 62, 75, 78, 86]. However, these techniques mainly focus on detecting and correcting semantic errors and assume structurally sound data as input. This assumption poses significant challenges when loading data into these systems. Our system addresses these challenges 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.

4.6 Conclusion

In this chapter, we introduced 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 the previous chapter for SURAGH to classify 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 we used during the development and evaluation of 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, TASHEEH can work with any row classifier, as demonstrated by successful integration with the PERFECT row classifier in our experiments. This flexibility of the TASHEEH transformation engine enables easy integration of future improved row classifiers.

In addition to its primary goal of reducing human effort during raw data preparation, TASHEEH functionalities offer promising future directions, e.g., data augmentation, preparation suggestion, and preparation estimation. Chapter 6 provides a more detailed exploration of these potential directions.

Chapter 5

MORPHER: DATA PREPARATION WITH SURAGH AND TASHEEH

In the pursuit of robust and accurate data-driven systems, the critical phase of data preparation often serves as the cornerstone upon which successful outcomes are built. In light of this, in this chapter we present an important component of our research, focusing on the creation and utilization of a user-friendly tool MORPHER, that incorporates our state-of-the-art error detection and correction systems — SURAGH and TASHEEH. While the former excels in the detection of structural errors within CSV files, the latter specializes in rectifying these inconsistencies. The synergy between these two systems presents an invaluable asset in our quest to ensure data quality, laying the foundation for rigorous analysis and informed decision-making.

We implemented both of our systems, SURAGH and TASHEEH, using the Java programming language. The complete source code, along with the datasets, manually created ground truth, and the implementation details are publicly available for both SURAGH[26] and TASHEEH[27].

MORPHER is an interactive system that aims to assist users in detecting and cleaning ill-formed rows in CSV files. Its interface allows users to visualize, for an input file, a classification of ill-formed *wanted* and ill-formed *unwanted* rows with a corresponding cleaned version and provides a seamless export of the final results as both CSV and Microsoft Excel workbook (.xlsx) formats for convenient use.

The content of this chapter draws on the research presented in our publication [34], which serves as a reference for this work.

The subsequent sections of the chapter are structured as follows: Section 5.1 presents an overview of MORPHER. Section 5.2 demonstrates the practical usage of our system through the graphical interface. Finally, Section 5.3 summarizes the contributions of the chapter.

[26]`https://github.com/HPI-Information-Systems/SURAGH`
[27]`https://github.com/HMazharHameed/TASHEEH`

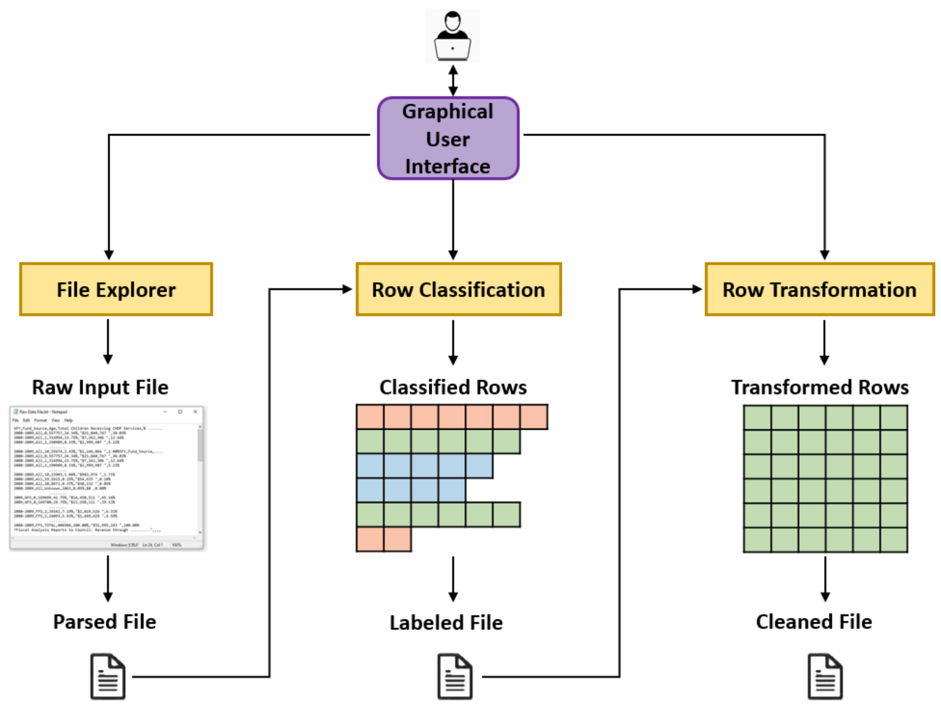


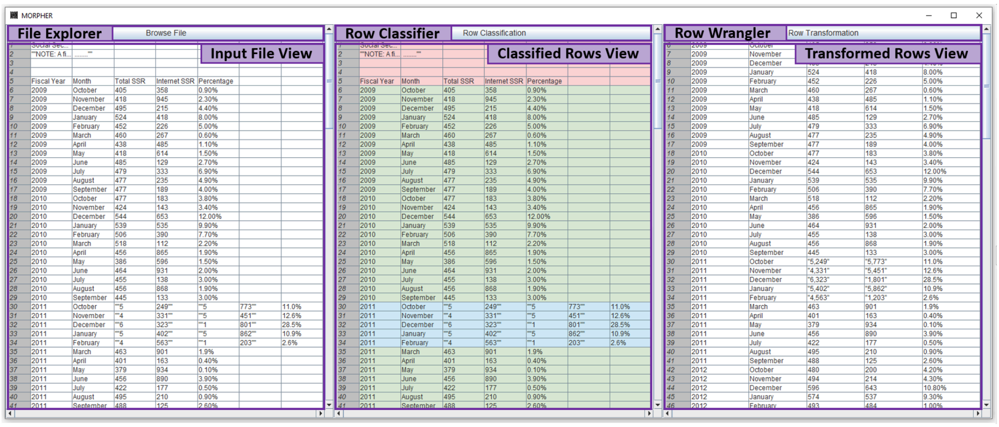
Figure 33: MORPHER overview

5.1 An Overview of MORPHER

MORPHER performs row classification and transformation in three phases. In the first phase, it enables users to browse directories and parse files. Subsequently, in the second phase, MORPHER first utilizes SURAGH to classify input file rows as either ill-formed or well-formed based on the dominant row pattern(s) (see Chapter 3). Additionally, SURAGH generates row patterns for the ill-formed rows, which are referred to as potential row patterns. This process is repeated incrementally until no ill-formed rows remain without a potential pattern. Then, MORPHER leverages TASHEEH to obtain the potential row patterns from the previous phase and further classifies ill-formed rows into wanted and unwanted ones (see Chapter 4). Finally, in the third phase, MORPHER utilizes TASHEEH to transform the wanted rows into well-formed ones using a set of pattern transformations (see Chapter 4). Figure 33 provides an overview of MORPHER's functionalities through a graphical user interface (GUI).

The graphical interface of MORPHER allows for seamless interaction with the results of automated row classification and transformation. Users can navigate through the rows within a file and review their classification. With automatic row transformation, users can clean up the structure of detected ill-formed wanted rows with just a click. An interactive MORPHER demo, accompanied by a demonstration scenario video, can be accessed online[28].

5.2 Self-Service Data Preparation with MORPHER



1	Social Sec...						
2	""NOTE: A fi...	........""					
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%		
9	2009	January	524	418	8.00%		
10	2009	February	452	226	5.00%		
11	2009	March	460	267	0.60%		
12	2009	April	438	485	1.10%		
13	2009	May	418	614	1.50%		
14	2009	June	485	129	2.70%		
15	2009	July	479	333	6.90%		
16	2009	August	477	235	4.90%		
17	2009	September	477	189	4.00%		
18	2010	October	477	183	3.80%		
19	2010	November	424	143	3.40%		
20	2010	December	544	653	12.00%		
21	2010	January	539	535	9.90%		
22	2010	February	506	390	7.70%		
23	2010	March	518	112	2.20%		
24	2010	April	456	865	1.90%		
25	2010	May	386	596	1.50%		
26	2010	June	464	931	2.00%		
27	2010	July	455	138	3.00%		
28	2010	August	456	868	1.90%		
29	2010	September	445	133	3.00%		
30	2011	October	""5	249""	""5	773""	11.0%
31	2011	November	""4	331""	""5	451""	12.6%
32	2011	December	""6	323""	""1	801""	28.5%
33	2011	January	""5	402""	""5	862""	10.9%
34	2011	February	""4	563""	""1	203""	2.6%
35	2011	March	463	901	1.9%		
36	2011	April	401	163	0.40%		
37	2011	May	379	934	0.10%		
38	2011	June	456	890	3.90%		
39	2011	July	422	177	0.50%		
40	2011	August	495	210	0.90%		
41	2011	September	488	125	2.60%		

1	Social Sec...						
2	""NOTE: A fi...	........""					
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%		
9	2009	January	524	418	8.00%		
10	2009	February	452	226	5.00%		
11	2009	March	460	267	0.60%		
12	2009	April	438	485	1.10%		
13	2009	May	418	614	1.50%		
14	2009	June	485	129	2.70%		
15	2009	July	479	333	6.90%		
16	2009	August	477	235	4.90%		
17	2009	September	477	189	4.00%		
18	2010	October	477	183	3.80%		
19	2010	November	424	143	3.40%		
20	2010	December	544	653	12.00%		
21	2010	January	539	535	9.90%		
22	2010	February	506	390	7.70%		
23	2010	March	518	112	2.20%		
24	2010	April	456	865	1.90%		
25	2010	May	386	596	1.50%		
26	2010	June	464	931	2.00%		
27	2010	July	455	138	3.00%		
28	2010	August	456	868	1.90%		
29	2010	September	445	133	3.00%		
30	2011	October	""5	249""	""5	773""	11.0%
31	2011	November	""4	331""	""5	451""	12.6%
32	2011	December	""6	323""	""1	801""	28.5%
33	2011	January	""5	402""	""5	862""	10.9%
34	2011	February	""4	563""	""1	203""	2.6%
35	2011	March	463	901	1.9%		
36	2011	April	401	163	0.40%		
37	2011	May	379	934	0.10%		
38	2011	June	456	890	3.90%		
39	2011	July	422	177	0.50%		
40	2011	August	495	210	0.90%		
41	2011	September	488	125	2.60%		

6	2009	October			
7	2009	November			
8	2009	December			
9	2009	January	524	418	8.00%
10	2009	February	452	226	5.00%
11	2009	March	460	267	0.60%
12	2009	April	438	485	1.10%
13	2009	May	418	614	1.50%
14	2009	June	485	129	2.70%
15	2009	July	479	333	6.90%
16	2009	August	477	235	4.90%
17	2009	September	477	189	4.00%
18	2010	October	477	183	3.80%
19	2010	November	424	143	3.40%
20	2010	December	544	653	12.00%
21	2010	January	539	535	9.90%
22	2010	February	506	390	7.70%
23	2010	March	518	112	2.20%
24	2010	April	456	865	1.90%
25	2010	May	386	596	1.50%
26	2010	June	464	931	2.00%
27	2010	July	455	138	3.00%
28	2010	August	456	868	1.90%
29	2010	September	445	133	3.00%
30	2011	October	"5,249"	"5,773"	11.0%
31	2011	November	"4,331"	"5,451"	12.6%
32	2011	December	"6,323"	"1,801"	28.5%
33	2011	January	"5,402"	"5,862"	10.9%
34	2011	February	"4,563"	"1,203"	2.6%
35	2011	March	463	901	1.9%
36	2011	April	401	163	0.40%
37	2011	May	379	934	0.10%
38	2011	June	456	890	3.90%
39	2011	July	422	177	0.50%
40	2011	August	495	210	0.90%
41	2011	September	488	125	2.60%
42	2012	October	480	200	4.20%
43	2012	November	494	214	4.30%
44	2012	December	596	643	10.80%
45	2012	January	574	537	9.30%
46	2012	February	493	484	1.00%

Figure 34: MORPHER's desktop-based user interface

We present the simple but effective MORPHER GUI through a demonstration scenario shown in Figure 34. To explore and interact with the system, users can download[28] MORPHER as a desktop application, along with a set of 148 exemplary raw CSV files obtained from four different open data sources.

We present a use-case in which a data scientist analyzes a company's growth and gains insights from the statistics in a CSV file. To begin the analysis, the scientist might need to load the file into a database connected to the analytics platform. Unfortunately, the file contains ill-formed rows that disrupt the file ingestion process. After encountering a halt during the file loading, the data scientist begins a manual inspection to narrow down the causes. However, due to the sheer amount of data and the complexity of the file, the data scientist may have overlooked some issues or needed help finding them amidst the wall of characters. The data scientist may have initially identified one or a few ill-formed rows, but cannot assume those are the only issues in the file. While some ill-formed rows may be easy to spot by simply scrolling through a file, such as those containing preambles or footnotes, others are more complicated to recognize, such as rows containing cell values with user-specific dialect details (non-standardized) or those containing additional metadata, making it challenging for even expert users to identify them manually. To ensure accurate analytics

[28]`https://github.com/HMazharHameed/MORPHER`

and reliable results, thoroughly inspecting the entire file to identify all problematic rows is crucial.

To overcome these challenges, the data scientist utilizes the functionalities of MORPHER, our unsupervised tool designed to detect and clean ill-formed rows in a CSV file automatically. The tool's GUI includes a ***file explorer***, which facilitates navigation and viewing of the input file. Using the GUI, with one click the data scientist runs MORPHER's automatic ***row classifier***, which assigns each row a label as either well-formed (i.e., clean), ill-formed but wanted (i.e., erroneous but can be cleaned), or ill-formed unwanted (i.e., non-data row). Figure 34 displays the output of the row classification module on the tool's GUI, where each row is color-coded based on its classification label for ease of interpretation.

After classification, the data scientist executes the final stage of MORPHER's pipeline: ***row wrangler***, automatically cleaning the ill-formed wanted rows with a single click. The transformed and standardized output file, as shown in Figure 34, can then be exported in both CSV and Excel formats, providing the data scientist with a handy resource for their intended task. Moreover, users have the convenience of executing the entire process through the command line interface.

The ability to efficiently identify and clean ill-formed rows streamlines the data processing pipeline and allows data scientists and machine learning engineers to spend more time on higher-level tasks, such as modeling and analytics.

5.3 Conclusion

MORPHER is an innovative tool that addresses a common data preparation challenge that data scientists face when working with CSV files – the presence of ill-formed rows. Through its intuitive GUI, MORPHER simplifies identifying and cleaning ill-formed rows, allowing data scientists to navigate and parse their files more efficiently. MORPHER automates a crucial step in data preparation, freeing up time and resources to be better spent during the later stages of a data processing pipeline. Overall, MORPHER represents a valuable contribution to the field of data science and has the potential to benefit a wide range of researchers and practitioners.

Chapter 6

CONCLUSION

Data play an integral role in contemporary society, permeating various facets of our daily lives, from professional environments and commercial operations to fields as diverse as sports, healthcare, aviation, agriculture, and marine biology. The exploitation of data trends not only accelerates progress in these domains but also aids in forecasting advancements and mitigating potential crises. As with any valuable resource, optimizing the utilization of data becomes critical, prompting the exploration of diverse data processing solutions tailored to the available resources.

Amidst this landscape, the preprocessing of file-based data has emerged as a widely embraced challenge, predominantly due to the prevalent availability of online data in file-based formats. Notably, open data portals serve as repositories housing a plethora of data files, with comma-separated value (CSV) files standing out as particularly favored by users and businesses owing to their adaptable standard and ease of use. However, the flexibility inherent in these files necessitates a significant responsibility for data consumers. Many files present structural issues, such as varying cell counts across data rows, diverse value formats within the same column, and discrepancies in quoted fields due to user specifications, among other challenges. Consequently, ingesting them into a host system, such as a database or an analytics platform, often requires prior data preparation steps.

6.1 Summary

Effectively ingesting files relies on accurate parsing. To achieve this, in this thesis, we emphasized the structure of a file – the set of characters necessary for precise parsing of data from the file. These characters are integral components of a file, playing a pivotal role in correctly organizing and interpreting data within that file. Driven by the need, we focused on comprehending the structure of CSV files and developed automated solutions to identify and rectify any potential structural inconsistencies that may arise within these files.

We began with an extensive survey of commercially available data preparation tools detailed in Chapter 2. In this survey, we not only collected features aimed at improving data quality, but also thoroughly examined the preprocessing challenges. The exploration was intended to encourage the community to enhance available features and devise more innovative solutions, addressing a wide range of inconsistencies. Throughout our survey, a recurring observation emerged: despite the data preparation and cleaning tools, these tools often necessitated a preprocessed file as their primary prerequisite, even though their main objective was to prepare data. These findings led to the development of our automatic structure detection and correction pipeline.

In Chapter 3, we introduced SURAGH, a pattern-based structural error detection system designed to identify rows containing structural inconsistencies (referred to as *ill-formed* rows) that may obstruct file parsing. With the SURAGH system, we performed classification based on the frequent patterns found within the file and detected erroneous rows. SURAGH creates patterns for individual cell values based on their content, encompassing elements, such as numbers, letters, and special characters. The aim is to abstract information effectively without compromising specificity; an excessive level of abstraction would result in mere strings, while retaining literal values would overly expand the search space. Once these patterns are generated, they are accumulated per column and subsequently merged to formulate row patterns that collectively represent entire rows within the input file. Finally, row patterns that exhibited dominance across a majority of instances were identified and retained, while others were omitted. These dominant patterns contribute to forming the pattern schema of a file, facilitating the identification of non-conforming (ill-formed) and conforming (well-formed) rows.

Subsequently, in Chapter 4, we presented TASHEEH, offering structural error correction through pattern transformation. Following the identification of rows with structural errors, our next objective was to rectify them. However, before initiating this rectification process, our initial step involved understanding which rows required transformation. For instance, we aimed not to clean empty rows or to impute values, and similarly, we aimed to avoid transforming footnote or comment rows. To determine which ill-formed rows needed transformation, we refined the classification within TASHEEH by further classifying ill-formed rows into *wanted* and *unwanted* rows. We then leveraged the pattern schema as the gold standard structure for the input file and transformed the structure of the *wanted* rows using our novel pattern transformation algebra.

We empirically demonstrated that both SURAGH and TASHEEH offer an end-to-end solution, enabling the seamless ingestion of files. To evaluate the effectiveness of our pipeline, we collected data from four open data sources: DataGov, Mendeley, GitHub, and UKGov. We manually annotated each row, establishing a ground truth dataset of over 200 000 rows distributed across 148 files. Our results showed that SURAGH achieved an average precision of 77% and an average recall of 97% in identifying ill-formed rows. On the other hand, TASHEEH demonstrated an average precision of 95% and an average recall of 96% in identifying ill-formed wanted rows across all files. Notably, TASHEEH automatically generated accurate transformations for 86% of ill-formed rows across all files, thus automatically recovering much data that could otherwise not be ingested. We have made both the code and annotated data for our entire data preparation pipeline publicly accessible.

Finally, in Chapter 5, we introduced MORPHER —a user-friendly tool that integrates the functionalities of both SURAGH and TASHEEH. MORPHER, featuring a graphical interface, enables seamless interaction with automated row classification and transformation results. Similar to SURAGH and TASHEEH, the artifacts for MORPHER are available online, accompanied by guidelines for users to customize its features as needed.

6.2 Outlook

While our end-to-end data preparation pipeline demonstrates remarkable performance, exploring the pattern-based approach developed in SURAGH and TASHEEH opens up numerous intriguing avenues for future research and enhancements. In the following discussion, we briefly discuss these potential opportunities and explore the directions for future investigations.

Semantic knowledge: The pattern language currently operates based on the concept of syntactic features inherent in the data values. For example, it understands how digits are arranged within values across columns, how strings are formed, which special characters are used, and how they are distributed across the data. This approach allows us to comprehend the structural aspects of input data, enabling us to identify structural deformities. While this grammar is highly effective for detecting structural inconsistencies, its capability to interpret information at the semantic level remains an area for further development. For instance, it cannot distinguish whether columns containing digits with special characters represent credit card numbers, phone numbers, or student registration IDs. Expanding this approach to include semantic roles—such as recognizing column headers like city names, postal codes, and phone numbers—holds great potential for identifying and correcting semantic errors. We believe that extending this grammar to encompass semantic understanding while retaining structural information could be a highly interesting approach, providing a robust pipeline for addressing both structural and semantic errors.

Pattern-based machine learning: Machine learning has achieved significant breakthroughs over the past few years, transforming numerous industries. A promising direction for future development is to utilize the pattern language as a foundation for training machine learning models. By leveraging the generated row patterns as input data, we can streamline the training process, enabling models to learn from structured and representative features. This approach would enhance the model's ability to identify and correct structural errors.

Moreover, pattern-based training could significantly reduce the need for extensive manual data labeling and individual row analysis, which are both time-consuming and error-prone. Automating this process would make it feasible to apply error detection and correction to much larger datasets, thereby improving overall scalability.

Data augmentation and data pollution: Data augmentation is a vital technique widely used in various fields, including machine learning, where it involves generating additional training examples from existing data to enhance model performance and generalization. By creating diverse and representative examples through data augmentation, machine learning models can adapt to a wider range of scenarios. In contrast, data pollution involves introducing errors or misleading information into the training dataset to improve model robustness. This process prepares models to handle real-world imperfections more effectively. An intriguing future direction is to utilize row patterns derived from well-formed and ill-formed rows for both data augmentation and data pollution. Well-formed row patterns can be used to generate additional training examples, enriching the dataset and improving model performance. Conversely, ill-formed row patterns can be strategically used to pollute data, enhancing the model's ability to manage and correct errors. These pattern-based strategies offer innovative approaches to boost model robustness and performance across diverse datasets and domains.

Preparation estimation: Another intriguing direction to explore is determining the effort required for file structure preparation. The distance measure proposed by our pipeline quantifies the degree of inconsistency in ill-formed rows, allowing users to estimate the effort needed for data preparation more accurately. Further investigation in this area could yield significant benefits, such as more precise time and cost estimates for data-driven projects, while also informing strategies to automate data cleaning pipelines and optimize resource allocation. To advance this research, it is crucial to collect extensive, representative datasets of both raw files and their prepared versions, and to conduct thorough user studies to establish reliable ground truth. These efforts will be essential in developing predictive models that accurately forecast preparation efforts, ultimately improving resource management and strategic planning for future data preparation tasks.

As we conclude this thesis, we revisit the expansive and intricate field of data preparation, where we continue to encounter a broad spectrum of challenges and opportunities. This spans from the initial collection of raw data to the creation of standardized, deployment-ready datasets. As we navigate this complex landscape, the aspiration of developing a comprehensive data preparation system becomes both a substantial and multifaceted endeavor. Such a system would not merely streamline individual tasks but would integrate a series of interconnected operations into a cohesive framework. This framework would encompass critical elements including data lineage tracking, automation of preparation pipelines, estimation and pipeline management, optimization techniques, and a collaborative marketplace for shared data preparation solutions.

This thesis contributes meaningfully to this vision by introducing an unsupervised structural preparation pipeline tailored for effective data ingestion. While this pipeline showcases robust functionality independently, it is also designed to integrate seamlessly into a broader, more comprehensive framework. Through this advancement, we progress toward achieving an integrated data preparation ecosystem that not only simplifies but also enhances the entire preparation process, thereby laying the foundation for more efficient and scalable solutions in the future.

References

[1] Trifacta end user data preparation. https://www.trifacta.com/wp-content/uploads/2018/02/End-User-Data-Preparation-Market-Study-2018.pdf. (last accessed September 19th, 2019).

[2] A. H. Abba and M. Hassan. Design and implementation of a csv validation system. In *Proceedings of the International Conference on Applications in Information Technology*, pages 111–116, 2018.

[3] Z. Abedjan, X. Chu, D. Deng, R. C. Fernandez, I. F. Ilyas, M. Ouzzani, P. Papotti, M. Stonebraker, and N. Tang. Detecting data errors: Where are we and what needs to be done? *PVLDB*, 9(12):993–1004, 2016.

[4] Z. Abedjan, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, and M. Stonebraker. DataXformer: A robust transformation discovery system. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 1134–1145. IEEE, 2016.

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

[6] P. D. Allison. *Missing data*. Sage publications, Thousand Oaks, CA, 2001.

[7] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.

[8] K. Bhageshpur. Data is the new oil–and that's a good thing. https://www.forbes.com/sites/forbestechcouncil/2019/11/15/data-is-the-new-oil-and-thats-a-good-thing, 2019.

[9] C. A. Charu. *Outlier Analysis*. Springer, 2013.

[10] N. Chepurko, R. Marcus, E. Zgraggen, R. C. Fernandez, T. Kraska, and D. R. Karger. ARDA: automatic relational data augmentation for machine learning. *PVLDB*, 13(9):1373–1387, 2020.

[11] B. Chopra, A. Fariha, S. Gulwani, A. Z. Henley, D. Perelman, M. Raza, S. Shi, D. Simmons, and A. Tiwari. Cowrangler: Recommender system for data-wrangling scripts. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 147–150, 2023.

[12] C. Christodoulakis, E. B. Munson, M. Gabel, A. D. Brown, and R. J. Miller. Pytheas: pattern-based table discovery in csv files. *PVLDB*, 13(12):2075–2089, 2020.

[13] X. Chu, I. F. Ilyas, and P. Papotti. Discovering denial constraints. *PVLDB*, 6(13): 1498–1509, 2013.

[14] X. Chu, Y. He, K. Chakrabarti, and K. Ganjam. Tegra: Table extraction by global record alignment. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 1713–1728, 2015.

[15] X. Chu, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, N. Tang, and Y. 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.

[16] G. Convertino and A. Echenique. Self-service data preparation and analysis by business users: New needs, skills, and tools. In *Proceedings of the CHI Conference Extended Abstracts on Human Factors in Computing Systems*, pages 1075–1083. ACM, 2017.

[17] M. Dallachiesa, A. Ebaid, A. Eldawy, A. Elmagarmid, I. F. Ilyas, M. Ouzzani, and N. Tang. Nadeef: a commodity data cleaning system. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 541–552. ACM, 2013.

[18] Y. Diao, K. Dimitriadou, Z. Li, W. Liu, O. Papaemmanouil, K. Peng, and L. Peng. Aide: an automatic user navigation system for interactive data exploration. *PVLDB*, 8(12):1964–1967, 2015.

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

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

[21] H. Dong, S. Liu, S. Han, Z. Fu, and D. 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.

[22] X. L. Dong and D. Srivastava. Big data integration. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 1245–1248. IEEE, 2013.

[23] EBNF. Iso/iec 14977:1996(e), extended bnf. `https://www.iso.org/standard/26153.html`, 1996. (last accessed May 25th, 2021).

[24] T. Economist. The world's most valuable resource is no longer oil, but data. `https://www.economist.com/leaders/2017/05/06/the-worlds-most-valuable-resource-is-no-longer-oil-but-data`, 2017.

[25] J. Ehrlich, M. Roick, L. Schulze, J. Zwiener, T. Papenbrock, and F. Naumann. Holistic data profiling: Simultaneous discovery of various metadata. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, pages 305–316, 2016.

[26] H. Elmeleegy, J. Madhavan, and A. Halevy. Harvesting relational tables from lists on the web. *PVLDB*, 2(1):1078–1089, 2009.

[27] A. Fariha, A. Tiwari, A. Meliou, A. Radhakrishna, and S. Gulwani. Coco: Interactive exploration of conformance constraints for data understanding and data cleaning. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 2706–2710, 2021.

[28] Forbes. 175 zettabytes by 2025. `https://www.forbes.com/sites/tomcoughlin/2018/11/27/175-zettabytes-by-2025/?sh=257ff2675459`, 2018. (last accessed April 3rd, 2023).

[29] C. Ge, Y. Li, E. Eilebrecht, B. Chandramouli, and D. 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.

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

[31] I. Google. Openrefine, 2022. URL `www.openrefine.org`. (last accessed August 30th, 2022).

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

[33] M. Hameed, G. Vitagliano, L. Jiang, and F. 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.

[34] M. Hameed, G. Vitagliano, and F. Naumann. MORPHER: structural transformation of ill-formed rows. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*, pages 5051–5055, 2023.

[35] M. Hameed, G. Vitagliano, F. Panse, and F. Naumann. TASHEEH: Repairing row-structure in raw csv files. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, 2024 (to appear).

[36] Y. He, X. Chu, K. Ganjam, Y. Zheng, V. Narasayya, and S. Chaudhuri. Transform-data-by-example (tde) an extensible search engine for data transformations. *PVLDB*, 11(10):1165–1177, 2018.

[37] J. Heer, J. M. Hellerstein, and S. Kandel. Predictive interaction for data transformation. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*. Citeseer, 2015.

[38] A. Heidari, J. McGrath, I. F. Ilyas, and T. Rekatsinas. Holodetect: Few-shot learning for error detection. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 829–846, 2019.

[39] A. Helal, M. Helali, K. Ammar, and E. Mansour. A demonstration of kglac: A data discovery and enrichment platform for data science. *PVLDB*, 14(12):2675–2678, 2021.

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

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

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

[43] Z. Huang and Y. He. Auto-detect: Data-driven error detection in tables. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 1377–1392, 2018.

[44] IDC. International data corporation. `www.idc.com`, 2023. (last accessed April 3rd, 2023).

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

[46] I. F. Ilyas and F. Naumann. Data errors: Symptoms, causes and origins. *IEEE Data Engineering Bulletin*, 45.

[47] I. F. Ilyas, X. Chu, et al. Trends in cleaning relational data: Consistency and deduplication. *Foundations and Trends® in Databases*, 5(4):281–393, 2015.

[48] T. Inc. Trifacta data engineering cloud, 2022. URL `www.trifacta.com`. (last accessed August 30th, 2022).

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

[50] Z. Jin, M. R. Anderson, M. Cafarella, and H. Jagadish. Foofah: Transforming data by example. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 683–698. ACM, 2017.

[51] Z. Jin, M. Cafarella, H. Jagadish, S. Kandel, M. Minar, and J. M. Hellerstein. Clx: Towards verifiable pbe data transformation. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, pages 265–276, 2019.

[52] Z. Jin, Y. He, and S. Chauduri. Auto-transform: learning-to-transform by patterns. *PVLDB*, 13(12):2368–2381, 2020.

[53] M. Joglekar, H. Garcia-Molina, and A. G. Parameswaran. Interactive data exploration with smart drill-down (extended version). *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, (1):1–1, 2017.

[54] S. Kandel, A. Paepcke, J. M. Hellerstein, and J. Heer. Wrangler: interactive visual specification of data transformation scripts. In *Proceedings of the International Conference on Human Factors in Computing Systems (CHI)*, pages 3363–3372, 2011.

[55] E. Koci, M. Thiele, W. Lehner, and O. Romero. Table recognition in spreadsheets via a graph representation. In *2018 13th IAPR International Workshop on Document Analysis Systems (DAS)*, pages 139–144. IEEE, 2018.

[56] E. Koci, M. Thiele, O. Romero, and W. Lehner. 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*, pages 78–100. Springer, 2019.

[57] E. Koci, M. Thiele, O. Romero, and W. 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.

[58] S. Krishnan, J. Wang, E. Wu, M. J. Franklin, and K. Goldberg. Activeclean: Interactive data cleaning for statistical modeling. *PVLDB*, 9(12):948–959, 2016.

[59] M. Lenzerini. Data integration: A theoretical perspective. In *Proceedings of the Symposium on Principles of Database Systems (PODS)*, pages 233–246. ACM, 2002.

[60] Y. Li, H. Sun, B. Dong, and H. W. Wang. Cost-efficient data acquisition on online data marketplaces for correlation analysis. *PVLDB*, 12(4):362–375, 2018.

[61] R. J. Little and D. B. Rubin. *Statistical analysis with missing data*, volume 793. John Wiley & Sons, 2019.

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

[63] M. Mahdavi, Z. Abedjan, R. Castro Fernandez, S. Madden, M. Ouzzani, M. Stonebraker, and N. Tang. Raha: A configuration-free error detection system. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 865–882, 2019.

[64] J. Mitlöhner, S. Neumaier, J. Umbrich, and A. Polleres. Characteristics of open data csv files. In *Proceedings of the International Conference on Open and Big Data (OBD)*, pages 72–79. IEEE, 2016.

[65] G. Nagy, S. Seth, and D. W. Embley. End-to-end conversion of HTML tables for populating a relational database. In *Proceedings of the IAPR International Workshop on Document Analysis Systems*, pages 222–226, 2014.

[66] A. Narayan, I. Chami, L. J. Orr, and C. Ré. Can foundation models wrangle your data? *PVLDB*, 16(4):738–746, 2022.

[67] F. Nargesian, E. Zhu, K. Q. Pu, and R. J. Miller. Table union search on open data. *PVLDB*, 11(7):813–825, 2018.

[68] F. Naumann. Data profiling revisited. *SIGMOD Record*, 42(4):40–49, 2014.

[69] F. Naumann and M. Herschel. An introduction to duplicate detection. *Synthesis Lectures on Data Management*, 2(1):1–87, 2010.

[70] S. B. Needleman and C. 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.

[71] P. Oliveira, F. Rodrigues, P. Henriques, and H. Galhardas. A taxonomy of data quality problems. In *2nd Int. Workshop on Data and Information Quality*, pages 219–233, 2005.

[72] S. Palkar, F. Abuzaid, P. Bailis, and M. Zaharia. Filter before you parse: Faster analytics on raw data with sparser. *PVLDB*, 11(11):1576–1589, 2018.

[73] M. Palmer. Data is the new oil. `https://ana.blogs.com/maestros/2006/11/dataisthenew.html`, 2006. (last accessed April 3rd, 2023).

[74] T. Papenbrock, T. Bergmann, M. Finke, J. Zwiener, and F. Naumann. Data profiling with Metanome. *PVLDB*, 8(12):1860–1863, 2015.

[75] J. Peng, D. Shen, N. Tang, T. Liu, Y. Kou, T. Nie, H. Cui, and G. Yu. Self-supervised and interpretable data cleaning with sequence generative adversarial networks. *PVLDB*, 16(3):433–446, 2022.

[76] D. Pinto, A. McCallum, X. Wei, and W. B. Croft. Table extraction using conditional random fields. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 235–242, 2003.

[77] G. Press. Cleaning data: Most time-consuming, least enjoyable data science task. *Forbes*, Mar. 2016.

[78] A. Qahtan, N. Tang, M. Ouzzani, Y. Cao, and M. Stonebraker. Pattern functional dependencies for data cleaning. *PVLDB*, 13(5):684–697, 2020.

[79] A. A. Qahtan, A. Elmagarmid, R. Castro Fernandez, M. Ouzzani, and N. 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.

[80] A. A. Qahtan, A. K. Elmagarmid, M. Ouzzani, and N. Tang. Fahes: Detecting disguised missing values. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 1609–1612, 2018.

[81] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. Improving language understanding by generative pre-training. 2018.

[82] E. Rahm and H. H. Do. Data cleaning: Problems and current approaches. *IEEE Data Engineering Bulletin*, 23(4):3–13, 2000.

[83] V. Raman and J. M. Hellerstein. Potter's wheel: An interactive data cleaning system. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, 2001.

[84] T. Rattenbury, J. M. Hellerstein, J. Heer, S. Kandel, and C. Carreras. *Principles of data wrangling: Practical techniques for data preparation*. O'Reilly Media, Inc., 2017.

[85] J. Reback, jbrockmendel, W. McKinney, J. V. den Bossche, M. Roeschke, T. Augspurger, S. Hawkins, P. Cloud, gfyoung, Sinhrks, P. Hoefler, A. Klein, T. Petersen, J. Tratner, C. She, W. Ayd, S. Naveh, J. Darbyshire, R. Shadrach, M. Garcia, J. Schendel, A. Hayden, D. Saxton, M. E. Gorelli, F. Li, T. Wörtwein, M. Zeitlin, V. Jancauskas, A. McMaster, and T. Li. pandas-dev/pandas: Pandas 1.4.3, June 2022. URL `https://doi.org/10.5281/zenodo.6702671`.

[86] T. Rekatsinas, X. Chu, I. F. Ilyas, and C. Ré. Holoclean: Holistic data repairs with probabilistic inference. *PVLDB*, 10(11):1190–1201, 2017.

[87] S. Schelter, D. Lange, P. Schmidt, M. Celikel, F. Biessmann, and A. Grafberger. Automating large-scale data quality verification. *PVLDB*, 11(12):1781–1794, 2018.

[88] T. Sellam and M. Kersten. Ziggy: Characterizing query results for data explorers. *PVLDB*, 9(13):1473–1476, 2016.

[89] V. Shah and A. 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*, pages 1–4, 2019.

[90] T. Siddiqui, A. Kim, J. Lee, K. Karahalios, and A. Parameswaran. Effortless data exploration with zenvisage: an expressive and interactive visual analytics system. *PVLDB*, 10(4):457–468, 2016.

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

[92] R. Singh and S. Gulwani. Learning semantic string transformations from examples. *PVLDB*, 5(8):740—-751, 2012.

[93] S. Song, A. Zhang, L. Chen, and J. Wang. Enriching data imputation with extensive similarity neighbors. *PVLDB*, 8(11):1286–1297, 2015.

[94] M. Stonebraker and I. F. Ilyas. Data integration: The current status and the way forward. *IEEE Data Engineering Bulletin*, 41(2):3–9, 2018.

[95] Y. Sun, S. Song, C. Wang, and J. Wang. Swapping repair for misplaced attribute values. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 721–732. IEEE, 2020.

[96] L. Tableau Software. Tableau, 2022. URL `www.tableau.com`. (last accessed August 30th, 2022).

[97] N. Tang, J. Fan, F. Li, J. Tu, X. Du, G. Li, S. Madden, and M. Ouzzani. Rpt: Relational pre-trained transformer is almost all you need towards democratizing data preparation. *PVLDB*, 14(8):1254–1261, 2021.

[98] I. G. Terrizzano, P. M. Schwarz, M. Roth, and J. 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.

[99] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample. Llama: Open and efficient foundation language models. *CoRR*, abs/2302.13971, 2023.

[100] G. J. van den Burg, A. Nazábal, and C. Sutton. Wrangling messy csv files by detecting row and type patterns. *Data Mining and Knowledge Discovery*, 33(6):1799–1820, 2019.

[101] G. Vitagliano, L. Jiang, and F. Naumann. Detecting layout templates in complex multiregion files. *PVLDB*, 15(3):646–658, 2021.

[102] G. Vitagliano, L. Reisener, L. Jiang, M. Hameed, and F. Naumann. Mondrian: Spreadsheet layout detection. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 2361–2364, 2022.

[103] G. Vitagliano, M. Hameed, L. Jiang, L. Reisener, E. Wu, and F. Naumann. Pollock: A data loading benchmark. *PVLDB*, 16(8):1870–1882, 2023.

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

[105] P. Wang and Y. He. Uni-detect: A unified approach to automated error detection in tables. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 811–828, 2019.

[106] P. Wang, Y. He, R. Shea, J. Wang, and E. Wu. Deeper: A data enrichment system powered by deep web. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 1801–1804. ACM, 2018.

[107] S. C. Weller and A. K. Romney. *Systematic data collection*, volume 10. Sage publications, 1988.

[108] H. Wickham. Tidy data. *Journal of statistical software*, 59(10):1–23, 2014.

[109] C. Zhang, C. Zhang, C. Li, Y. Qiao, S. Zheng, S. K. Dam, M. Zhang, J. U. Kim, S. T. Kim, J. Choi, et al. One small step for generative ai, one giant leap for agi: A complete survey on chatgpt in aigc era. *CoRR*, abs/2304.06488, 2023.

Selbstständigkeitserklärung

Ich erkläre hiermit, dass

- ich die vorliegende Dissertationsschrift selbständig und ohne unerlaubte Hilfe angefertigt sowie nur die angegebene Literatur verwendet habe,
- die Dissertation keiner anderen Hochschule in gleicher oder ähnlicher Form vorgelegt wurde,
- mir die Promotionsordnung der Digital Engineering Fakultät der Universität Potsdam vom 27. November 2019 bekannt ist.

Mazhar Hameed – 18. December 2023